THE
TRANSPUTER
DATABOOK







INMOS Limited 1000 Aztec West Almondsbury Bristol BS12 4SQ UK Telephone (0454) 616616 Telex 444723

INMOS Japan K.K.
4th Floor No 1 Kowa Bldg
11-41 Akasaka 1-chome
Minato-ku
Tokyo 107
Japan
Telephone 03-505 2840
Telex J29507 TEI JPN
Fax 03-505 2844

INMOS Corporation PO Box 16000 Colorado Springs CO 80935 USA Telephone (719) 630 4000 Telex (Easy Link) 629 44 936

INMOS SARL Immeuble Monaco 7 rue Le Corbusier SILIC 219 94518 Rungis Cedex France Telephone (1) 46.87.22.01 Telex 201222 INMOS GmbH Danziger Strasse 2 8057 Eching Munich West Germany Telephone (089) 319 10 28 Telex 522645

### LOCAL U.S. SALES OFFICES

INMOS Corporation 200 E Sandpointe Suite 650 Santa Ana CA 92707 Telephone(714) 957 6018

INMOS Corpoation 2620 Augustine Drive Suite 100 Santa Clara CA 95054 Telephone (408) 727 7771

INMOS Corporation 12400 Whitewater Drive Suite 120 Minnetonka MN 55343 Telephone (612) 932 7121 INMOS Corporation 6025-G Atlantic Blvd Norcross GA 30071 Telephone (404) 242 7444

INMOS Corporation 5 Burlington Woods Drive Suite 201 Burlington MA 01803 Telephone (617) 229 2550

INMOS Corporation 10200 E Girard Avenue Suite B239 Denver CO 80231 Telephone (303) 368 0561 INMOS Corporation 14643 Dallas Parkway Suite 730 Dallas TX 75240 Telephone (214) 490 9522

INMOS Corporation 9861 Broken Land Parkway Suite 320 Columbia MD 21046 Telephone (301) 995 6952

INMOS Corporation PO Box 272 Fishkill NY 12524 Telephone (914) 897 2422

### **INMOS Databook Series**

Transputer Databook Military Micro-products Databook

Transputer Support Databook: Development and Sub-systems

Memory Databook Graphics Databook Digital Signal Processing Databook

Transputer Applications Notebook: Architecture and Software Transputer Applications Notebook: Systems and Performance

Copyright ©INMOS Limited 1989

INMOS reserves the right to make changes in specifications at any time and without notice. The information furnished by INMOS in this publication is believed to be accurate; however, no responsibility is assumed for its use, nor for any infringement of patents or other rights of third parties resulting from its use. No licence is granted under any patents, trademarks or other rights of INMOS.

, Inmos, IMS and occam are trademarks of the INMOS Group of Companies.

INMOS is a member of the SGS-THOMSON Microelectronics Group.

INMOS document number: 72 TRN 203 01



## TRANSPUTER DATABOOK

### **Contents overview**

1	INMOS	1
2	Transputer architecture	5
3	Transputer overview	27
4	IMS T805 engineering data	47
5	IMS T801 engineering data	127
6	IMS T800 engineering data	189
7	IMS T425 engineering data	261
8	IMS T414 engineering data	333
9	IMS T222 engineering data	399
10	IMS T225 preview	453
11	IMS M212 preview	463
12	IMS C004 engineering data	479
13	IMS C011 engineering data	503
14	IMS C012 engineering data	529
A	Quality and Reliability	551
В	Index	557

### Contents

	Preface			xix
	Notation and	nomencla	ature	xx
1	INMOS			1
	1 Introd	duction		2
	1.1	Manuf	acturing	2
	1.2	Assen	nbly	2
	1.3	Test		2
	1.4		y and Reliability	2
	1.5	Militar		2
	1.6		Developments	3
		1.6.1	Research and Development	3
		1.6.2	Process Developments	3
	<b>T</b>			
2	Transputer	archited	cture	5
	1			
	1 Introd	duction Overv	low	6
	1.1	Overv	Transputers and occam	7
	1.2	Systol	m design rationale	
	1.2	1.2.1	Programming	8
		1.2.2	Hardware	9
		1.2.3		9
	1.3		ms architecture rationale	9
		1.3.1	Point to point communication links	9
		1.3.2	Local memory	10
	1.4		nunication	10
		.m model		12
	2.1	Overv		12
	2.2		n overview	13
		2.2.1	Processes	13
			Assignment	13
			Input	13
			Output	13
		2.2.2	Constructions	14
			Sequence Parallel	14 14
			Communication	15
			Conditional	15
			Alternation	15
			Loop	16
			Selection	16
			Replication	16
		2.2.3	Types	17
		2.2.4	Declarations, arrays and subscripts	17
		2.2.5	Procedures	18
		2.2.6	Functions	18
		2.2.7	Expressions	18

			2.2.8 Timer	19
			2.2.9 Peripheral access	19
		2.3	Configuration	20
			PLACED PAR	20
			PRI PAR	20
			2.3.1 INMOS standard links	20
	3	Error	handling	21
	4		ram development	22
		4.1	Logical behaviour	22
		4.2	Performance measurement	22
		4.3	Separate compilation of OCCAM and other languages	22
		4.4	Memory map and placement	23
	5	Physi	ical architecture	24
		5.1	INMOS serial links	24
			5.1.1 Overview	24
			5.1.2 Link electrical specification	24
		5.2	System services	24
			5.2.1 Powering up and down, running and stopping	24
			5.2.2 Clock distribution	25
		5.3	Bootstrapping from ROM or from a link	25
***************************************		5.4	Peripheral interfacing	25
3	Trar	sputer	overview	27
	1	Introd	duction	28
	2	The t	ransputer: basic architecture and concepts	29
		2.1	A programmable device	29
		2.2	occam	29
		2.3	VLSI technology	29
		2.4	Simplified processor with micro-coded scheduler	30
	3	Trans	sputer internal architecture	31
		3.1	Sequential processing	32
		3.2	Instructions	32
		<del></del>	3.2.1 Direct functions	33
			3.2.2 Prefix functions	33
			3.2.3 Indirect functions	34
			3.2.4 Efficiency of encoding	34
		3.3	Support for concurrency	34
		3.4	Communications	36
			3.4.1 Internal channel communication	36
			3.4.2 External channel communication	38
			3.4.3 Communication links	39
		3.5	Timer	40
		3.6	Alternative	40
		3.7	Floating point instructions	40
			3.7.1 Optimising use of the stack	41
			3.7.2 Concurrent operation of FPU and CPU	41
		3.8	Floating point unit design	42

		3.9	Graphic	s capability	43
			3.9.1	Example - drawing coloured text	43
	4	Conc	lusion		45
	1840	TOOL -			
4	IIVIS	1805 e	ngineerin	g data	47
	1	Introd	duction		48
	2	Pin d	esignations	3	51
	3	Proce			52
		3.1	Register		52
		3.2	Instructi		53
				Direct functions	53
				Prefix functions	53
				Indirect functions	54
				Expression evaluation	54 54
		3.3		Efficiency of encoding less and concurrency	55
		3.4	Priority	es and concurrency	56
		3.5		nications	56
		3.6	Block m		57
		3.7	Timers		57
			11111010		
	4	Instru	ction set s	summary	59
		4.1		duling points	60
		4.2		structions	61
		4.3	Debugg	ing support	61
		4.4	Floating	point errors	61
	5	Float	ing point u	nit	69
	6		m services		71
		6.1	Power		71
		6.2		s, CapMinus	71
		6.3	Clockin	and Calando O	71
		6.4		eedSelect0-2	72
		6.5 6.6	Reset Bootstra	2N	73 73
	·	6.7	Peek an		75
		6.8	Analyse		75
		6.9	Error, E		76
	7	Memo	orv		77
	····				
	8	Exter	nal memor	y interface	79
		8.1	Pin fund		80
			8.1.1	MemAD2-31	80
			8.1.2	notMemRd	80
			8.1.3	MemnotWrD0	
			8.1.4	notMemWrB0-3	80
			8.1.5	notMemS0-4	80

			8.1.6 MemWait	80
			8.1.7 MemnotRfD1	80
	**************************************		8.1.8 notMemRf	80
			8.1.9 RefreshPending	80
			8.1.10 MemReq, MemGranted	81
			8.1.11 MemConfig	81
			8.1.12 ProcClockOut	82
		8.2	Read cycle	83
		8.3	Write cycle	88
		8.4	Wait	89
		8.5	Memory refresh	91
		8.6	Direct memory access	94
		8.7	Memory configuration	96
			8.7.1 Internal configuration	96
			8.7.2 External configuration	98
	9	Events	3	103
	10	Links		105
		Liiks		100
	11	Electri	ical specifications	108
		11.1	DC electrical characteristics	108
		11.2	Equivalent circuits	109
		11.3	AC timing characteristics	110
		11.4	Power rating	112
	12	Perfor	mance	113
		12.1	Performance overview	113
		12.2	Fast multiply, TIMES	115
		12.3	Arithmetic	116
		12.4	Floating point operations	117
			12.4.1 Floating point functions	117
			12.4.2 Special purpose functions and procedures	118
		12.5	Effect of external memory	118
		12.6	Interrupt latency	119
	13	Packa	ge specifications	120
		13.1	84 pin grid array package	120
		13.2	84 pin PLCC J-bend package	122
		13.3	84 lead quad cerpack package	124
	14	Orderi	ng	126
5	IMS	T801 en	ngineering data	127
	1	Introdi		128
		mirodi	uction	·
	2	Pin de	esignations	131

3	Proce		132
	3.1	Registers	132
	3.2	Instructions	133
		3.2.1 Direct functions	133
		3.2.2 Prefix functions	133
		3.2.3 Indirect functions	134
		3.2.4 Expression evaluation	134
		3.2.5 Efficiency of encoding	134
	3.3	Processes and concurrency	135
	3.4	Priority	136
	3.5	Communications	136
	3.6	Block move	137
	3.7	Timers	137
4	Inetru	ction set summary	139
<b></b>	4.1	Descheduling points	140
	4.2	Error instructions	141
	4.3	Debugging support	141
	4.4	Floating point errors	141
	7.7	Troating point errors	141
5	Floatir	ng point unit	149
	1 104111	ng point aint	170
6	System	m services	151
	6.1	Power	151
	6.2	CapPlus, CapMinus	151
	6.3	Clockin	151
	6.4	ProcSpeedSelect0-2	152
	6.5	Reset	153
	6.6	Bootstrap	153
	6.7	Peek and poke	155
	6.8	Analyse	155
	6.9	ErrorOut	156
			and the second
7	Memo	ory	157
8	Extern	nal memory interface	159
	8.1	Pin functions	160
		8.1.1 MemA2-31	160
		8.1.2 MemD0-31	160
		8.1.3 notMemCE	160
		8.1.4 notMemWrB0-3	161
		8.1.5 MemWait	161
		8.1.6 MemReq, MemGranted	161
		8.1.7 ProcClockOut	162
	8.2	Read cycle	163
	8.3	Write cycle	164
	8.4	Wait	165
	8.5	Direct memory access	167
	F		400
9	Event	3	169
10	Links		171
10	LIIIKS		171

	11	Electrical specifications				
		11.1	DC electrical characteristics	174		
		11.2	Equivalent circuits	175		
		11.3	AC timing characteristics	176		
		11.4	Power rating	177		
	12	Perfo	rmance	179		
		12.1	Performance overview	179		
		12.2	Fast multiply, TIMES	181		
		12.3	Arithmetic	182		
		12.4	Floating point operations	183		
			12.4.1 Floating point functions	183		
				184		
		12.5		184		
		12.6	Interrupt latency	185		
	13	Packa	age specifications	186		
		13.1	100 pin grid array package	186		
	14	Order	ring	188		
6	IMS	T800 ei	ngineering data	189		
	1	Introd	duction	190		
	2	Pin d	esignations	192		
	3	Proce	DC electrical characteristics  Equivalent circuits  AC timing characteristics  Power rating  ormance  Performance overview  Fast multiply, TIMES  Arithmetic  Floating point operations  12.4.1 Floating point functions  12.4.2 Special purpose functions and procedures  Effect of external memory  Interrupt latency  kage specifications  100 pin grid array package			
		3.1	Registers	193		
		3.2	Instructions	194		
				194		
				194		
				195		
	·			195		
	····			195		
		3.3		196		
		3.4		197		
		3.5		197		
		3.6		198		
		3.7	Timers	198		
	4			200		
		4.1		201		
		4.2		202		
		4.3	Floating point errors	202		
	5	Floati	ing point unit	209		

6	Systen	n services	211
	6.1	Power	211
	6.2	CapPlus, CapMinus	211
	6.3	Clockin	211
	6.4	ProcSpeedSelect0-2	212
	6.5	Reset	213
	6.6	Bootstrap	213
	6.7	Peek and poke	215
	6.8	Analyse	215
	6.9	Error, Errorin	216
7	Memo	ry	217
8		al memory interface	219
	8.1	ProcClockOut	219
	8.2	Tstates	219
	8.3	Internal access	220
	8.4	MemAD2-31	221
	8.5	MemnotWrD0	221
	8.6	MemnotRfD1	221
	8.7	notMemRd	221
	8.8	notMemS0-4	221
	8.9	notMemWrB0-3	225
	8.10	MemConfig	228
		8.10.1 Internal configuration	228
		8.10.2 External configuration	230
	8.11	notMemRf	235
	8.12	MemWait	236
	8.13	MemReq, MemGranted	238
9	Events		240
	WHAT CO		
10	Links		241
11		cal specifications	244
	11.1	DC electrical characteristics	244
	11.2	Equivalent circuits	245
	11.3	AC timing characteristics	246
	11.4	Power rating	248
12		mance	249
	12.1	Performance overview	249
	12.2	Fast multiply, TIMES	251
	12.3	Arithmetic	252
	12.4	Floating point operations	253
		12.4.1 Floating point functions	253
	10.5	12.4.2 Special purpose functions and procedures	254
	12.5 12.6	Effect of external memory	254 255
	1∠.0	Interrupt latency	255

	13	Packa	ige specifi		256
		13.1		grid array package	256
		13.2	84 lead	quad cerpack package	258
	14	Order	ring		260
7	IMS	T425 e	ngineerii	ng data	261
	1	Introd	duction		262
	2	Pin d	esignation	IS	264
					005
	3	Proce			265
		3.1	Registe Instruc		265
		3.2	3.2.1	Direct functions	266 266
			3.2.1	Prefix functions	266
			3.2.3	Indirect functions	267
			3.2.3	Expression evaluation	267
			3.2.5	Efficiency of encoding	267
		3.3		ses and concurrency	268
		3.4	Priority		269
		3.5		unications	269
un		3.6	Block		270
		3.7	Timers		270
		3.7	11111013		210
	4	Instru	uction set	summary	272
	•	4.1		eduling points	273
		4.2		nstructions	274
		4.3	Debug	ging support	274
				3 - 11	
	5	Syste	m service	S	280
		5.1	Power		280
		5.2	CapPlu	s, CapMinus	280
		5.3	Clockir		280
	<u></u>	5.4	ProcSp	eedSelect0-2	281
		5.5	Reset		282
		5.6	Bootsti	rap	282
		5.7	Peek a	nd poke	284
		5.8	Analys	e	284
		5.9	Error, E	Errorin	285
	6	Memo	ory		286
	7			ry interface	288
		7.1	ProcCl		288
		7.2	Tstates		288
		7.3		Laccess	289
		7.4	MemA		290
		7.5	Memno		290
		7.6	Memno		290
		7.7	notMen	nRd	290

		7.8	notMemS0-4	290
		7.9	notMemWrB0-3	294
		7.10	MemConfig	297
			7.10.1 Internal configuration	297
			7.10.2 External configuration	299
		7.11	RefreshPending	304
		7.12	notMemRf	305
		7.13	MemWait	306
		7.14	MemReq, MemGranted	308
	8	Events	S	310
	9	Links		312
	10	Flectr	ical specifications	315
		10.1	DC electrical characteristics	315
		10.2	Equivalent circuits	316
		10.3	AC timing characteristics	317
		10.4	Power rating	319
	11	Porfor	mance	320
		11.1	Performance overview	320
		11.2	Fast multiply, TIMES	322
		11.3	Arithmetic	322
		11.4	Floating point operations	323
		11.7	11.4.1 Special purpose functions and procedures	324
		11.5	Effect of external memory	324
		11.6	Interrupt latency	325
		11.0	menapt laterioy	
	12	Packa	ge specifications	326
		12.1	84 pin grid array package	326
		12.2	84 pin PLCC J-bend package	328
		12.3	84 lead quad cerpack package	330
	13	Orderi	ing	332
8	IMS	T414 er	ngineering data	333
	1	Introd	uction	334
	2	Pin de	esignations	336
	3	Proces	CCOL	337
	<u> </u>	3.1	Registers	337
		3.2	Instructions	338
		U.L	3.2.1 Direct functions	338
			3.2.2 Prefix functions	338
			3.2.3 Indirect functions	339
		······································	3.2.4 Expression evaluation	339
			3.2.5 Efficiency of encoding	339
		3.3	Processes and concurrency	340
		3.4	Priority	341
		2.5	Communications	2/1

 	3.6	Timers	342
 4	Instru	ction set summary	343
	4.1	Descheduling points	344
 	4.2	Error instructions	344
 	7.4	Error mattactions	
 5	Systen	m services	349
	5.1	Power	349
 	5.2	CapPlus, CapMinus	349
	5.3	Clockin	349
	5.4	Reset	351
***************************************	5.5	Bootstrap	351
	5.6	Peek and poke	353
	5.7	Analyse	353
 	5.8	Error	354
 6	Memo	M.	355
 	Wento	ıy	333
 7	Extern	nal memory interface	357
	7.1	ProcClockOut	357
 	7.2	Tstates	357
 	7.3	Internal access	358
 	7.4	MemAD2-31	359
 	7.5	MemnotWrD0	359
 ·	7.6	MemnotRfD1	359
	7.7	notMemRd	359
 ······································	7.8	notMemS0-4	359
	7.9	notMemWrB0-3	363
	7.10	MemConfig	366
		7.10.1 Internal configuration	366
		7.10.2 External configuration	368
	7.11	notMemRf	373
	7.12	MemWait	374
	7.13	MemReq, MemGranted	376
 8	Events		378
 	Lvents		370
9	Links		379
 10		ical specifications	382
 	10.1	DC electrical characteristics	382
 	10.2	Equivalent circuits	383
 	10.3	AC timing characteristics	384
 ***************************************	10.4	Power rating	386
 11	Perfori	mance	387
 	11.1	Performance overview	387
 	11.2	Fast multiply, TIMES	389
 <del></del>	11.3	Arithmetic	389
 	11.4	Floating point operations	390
 	11.5	Effect of external memory	391
 	11.6	Interrupt latency	392

	12	Packa	ge specifications	393
		12.1	84 pin grid array package	393
		12.2	84 pin PLCC J-bend package	395
	13	Orderi	ing	397
	13	Orden	ing	
9	IMS	T222 en	ngineering data	399
				400
	1	Introd	uction	400
	2	Pin de	esignations	402
	3	Proces		403
		3.1	Registers	403
		3.2	Instructions	404
			3.2.1 Direct functions	404
			3.2.2 Prefix functions	404
	***************************************		3.2.3 Indirect functions	405
			3.2.4 Expression evaluation	405
			3.2.5 Efficiency of encoding	405
		3.3	Processes and concurrency	406
		3.4	Priority	407
		3.5	Communications	407
		3.6	Timers	408
	4	Instru	ction set summary	409
		4.1	Descheduling points	410
		4.2	Error instructions	410
	5	System	m services	415
		5.1	Power	415
		5.2	CapPlus, CapMinus	415
		5.3	Clockin	415
		5.4	Reset	416
		5.5	Bootstrap	416
		5.6	Peek and poke	418
		5.7	Analyse	418
		5.8	Error	419
				100
	6	Memo	ory	420
	7	Extern	nal memory interface	422
		7.1	ProcClockOut	422
		7.2	Tstates	423
		7.3	Internal access	423
		7.4	Mem A0-15	423
		7.5	MemD0-15	423
		7.6	notMemWrB0-1	424
		7.7	notMemCE	426
		7.8	MemBAcc	428
		7.9	MemWait	429
		7.10	MemReq, MemGranted	431

	8	Events	3		433
	9	Links			434
	10			ifications	437
		10.1		ctrical characteristics	437
		10.2		lent circuits	438
		10.3		ing characteristics	439
		10.4	Power	rating	441
	11	Perfor	mance		442
		11.1		mance overview	442
		11.2		ultiply, TIMES	444
		11.3	Arithm		444
		11.4		g point operations	445
		11.5		of external memory	446
		11.6		pt latency	447
	12		ge specif	fications	448
		12.1	68 pin	grid array package	448
		12.2	ьв ріп	PLCC J-bend package	450
	13	Orderi	ng		452
10	IMS	IMS T225 preview			
<del></del>					453
	1	Introd	uction		454
	2	Pin de	signatio	ns	456
	3	Instru	ction set	summary	457
	4	Packa	ge speci	fications	459
		4.1		grid array package	459
		4.2	68 pin	PLCC J-bend package	460
	5	Orderi	ng		461
11	IMS M212 preview 4				
	1	Introd	uction		464
		1.1		212 peripheral processor	465
		1.1	1.1.1	Central processor	465
			1.1.2	Peripheral interface	465
			1.1.3	Disk controller	465
			1.1.4	Links	466
			1.1.5	Memory system	466
			1.1.6	Error handling	466
				<u> </u>	
	2	Operation			467
		2.1	Mode		467
		2.2	Mode :	2	468

	3	Appli	ications	469	
	4	Pack	age specifications	473	
		4.1	68 pin grid array package	473	
		4.2	68 pin PLCC J-bend package	475	
	5	Orde	ring	477	
	3	Orde	illig	4//	
12	IMS	C004 e	engineering data	479	
····	1	Intro	duction	480	
		Dia d	I a lama Alaus	404	
***************************************	2	Pin a	designations	481	
	3		em services	482	
		3.1	Power	482	
		3.2	CapPlus, CapMinus	482	
		3.3	Clockin	482	
		3.4	Reset	484	
	4	Links	3	485	
	5	Curito	h implementation	489	
	<u> </u>	SWILL	ch implementation	409	
	6		ications	490	
		6.1	Link switching	490	
		6.2	Multiple IMS C004 control	490	
		6.3	Bidirectional exchange	490	
		6.4	Bus systems	490	
	7	Elect	rical specifications	494	
		7.1	DC electrical characteristics	494	
		7.2	Equivalent circuits	495	
		7.3	AC timing characteristics	496	
		7.4	Power rating	497	
	8	Packa	age specifications	498	
		8.1	84 pin grid array package	498	
		8.2	84 lead quad cerpack package	500	
	9	Orde	ring	502	
13	IMS C011 engineering data				
	mo our originating data				
	1	Intro	duction	504	
	2	Pin d	designations	505	

	3	Syste	m service	S	506
		3.1	Power		506
		3.2	CapMir		506
		3.3	Clockir		506
		3.4	Separa	telQ	507
		3.5	Reset		508
	4	Links			509
	5	Mode	1 paralle	I interface	512
		5.1	Input p		512
		5.2	Output	port	513
	6	Mode	2 Paralle	l interface	514
		6.1	D0-7		514
		6.2	notCS		514
		6.3	RnotW		514
		6.4	RS0-1		514
			6.4.1	Input Data Register	514
			6.4.2	Input Status Register	517
			6.4.3	Output Data Register	517
			6.4.4	Output Status Register	517
		6.5	InputIn		517
		6.6	Output		518
		6.7	Data re		518
		6.8	Data w	rite	518
	7	Electi	rical speci	ifications	519
		7.1		ctrical characteristics	519
		7.2		lent circuits	520
		7.3		ing characteristics	521
	·	7.4	Power	rating	523
	8	Packa	age specif		524
		8.1	28 pin	plastic dual-in-line package	524
		8.2	28 pin	ceramic dual-in-line package	525
		8.3		SOIC package	526
		8.4	Pinout		527
·	9	Orde	ring		528
14	IMC	C012 o	ngineeri	na data	529
14	TIVIO	CUIZE	ngineen	ng data	529
	1	Introd	duction		530
·····	2	Pin d	esignation	ns	531
		0			
	3	3.1	m service	<u> </u>	532 532
		3.1	Power	2110	532
		3.2	CapMir Clocklr		532
		3.4	Reset	1	534

	4	Links		535
				· · · · · · · · · · · · · · · · · · ·
	5	Paralle	I interface	538
		5.1	D0-7	538
		5.2	notCS	538
		5.3	RnotW	538
		5.4	RS0-1	538
			5.4.1 Input Data Register	538
			5.4.2 Input Status Register	541
			5.4.3 Output Data Register	541
			5.4.4 Output Status Register	541
		5.5	Inputint	541
		5.6	OutputInt	542
		5.7	Data read	542
		5.8	Data write	542
	6	Flectri	cal specifications	543
		6.1	DC electrical characteristics	543
		6.2	Equivalent circuits	544
		6.3	AC timing characteristics	545
		6.4	Power rating	547
	7		ge specifications	548
		7.1	24 pin plastic dual-in-line package	548
		7.2	Pinout	549
	8	Orderi	ηα	550
A	Qualit	y and	Reliability	551
	A		y and Reliability	552
		A.1	Total quality control (TQC) and reliability programme	552
		A.2	Quality and reliability in design	552
		A.3	Document control	553
		A.4	New product qualification	553
		A.5	Product monitoring programme	553
		A.6	Production testing and quality monitoring procedure	554
			A.6.1 Reliability testing	554
			A.6.2 Production testing	554
			A.6.3 Quality monitoring procedure	555
В	Index			557
	- IIIGOA			
	В	Index		559

### **Preface**

This databook describes the architecture of the transputer family of products and details some of the devices which make up that family. Items described include the 32 bit and 16 bit transputer products IMS T805, IMS T801, IMS T800, IMS T425, IMS T414, IMS T222 and IMS T225; the peripheral controller IMS M212; and the communications devices IMS C004, IMS C011 and IMS C012. For details of the military version of a device refer to *The Military Micro-products Databook* which is available as a separate publication.

The databook first describes the transputer architecture and general features of transputer family devices. It then continues with the various product datasheets.

A transputer is a single VLSI device with processor, memory and communications links for direct connection to other transputers. Concurrent systems can be constructed from a collection of transputers operating concurrently and communicating through links. The transputer can be used as a building block for concurrent processing systems, with OCCAM as the associated design formalism.

Current transputer products include the 16 bit IMS T222, the 32 bit IMS T414 and IMS T425, and the IMS T800, IMS T801 and IMS T805 which are 32 bit transputers with an integral high speed floating point processor. A product preview of the IMS T225, which is a 16 bit transputer with debugger support, is also included.

The IMS M212 is an intelligent peripheral controller. It contains a 16 bit processor, on-chip memory and communications links. It contains hardware and interface logic to control disk drives and can be used as a programmable disk controller or as a general purpose peripheral interface.

The INMOS serial communication link is a high speed system interconnect which provides full duplex communication between members of the transputer family. It can also be used as a general purpose interconnect even where transputers are not used. The IMS C011 and IMS C012 link adaptors are communications devices enabling the INMOS serial communication link to be connected to parallel data ports and microprocessor buses. The IMS C004 is a programmable link switch. It provides a full crossbar switch between 32 link inputs and 32 link outputs.

The transputer development system referred to in this databook comprises an integrated editor, compiler and debugging system which enables transputers to be programmed in OCCAM and in industry standard languages, for example, C, Fortran, Pascal. The *Transputer Development System Manual* is supplied with the transputer development system and is available as a separate publication.

Other information relevant to all transputer products is contained in the OCCAM Reference Manual, supplied with INMOS software products and available as a separate publication. If more detail on the machine level operation is required, refer to Transputer Instruction Set - A Compiler Writers' Guide, which is available as a separate publication.

Various application and technical notes are also available from INMOS.

Software and hardware examples given in this databook are outline design studies and are included to illustrate various ways in which transputers can be used. The examples are not intended to provide accurate application designs.

In addition to transputer devices, the INMOS product range also includes graphics products, digital signal processing devices and memory devices. For further information concerning INMOS products, please contact your local INMOS sales outlet.

### Notation and nomenclature

The nomenclature and notation in general use throughout this databook is described below.

### Significance

The bits in a byte are numbered 0 to 7, with bit 0 least significant. The bytes in words are numbered from 0, with byte 0 least significant. In general, wherever a value is treated as a number of component values, the components are numbered in order of increasing numerical significance, with the least significant component numbered 0. Where values are stored in memory, the least significant component value is stored at the lowest (most negative) address.

Similarly, components of arrays are numbered starting from 0 and stored in memory with component 0 at the lowest address.

Transputer memory is byte addressed, with words aligned on four-byte boundaries for 32 bit devices and on two-byte boundaries for 16 bit devices.

Hexadecimal values are prefixed with #, as in #1DF.

Where a byte is transmitted serially, it is always transmitted least significant bit (0) first. In general, wherever a value is transmitted as a number of component values, the least significant component is transmitted first. Where an array is transmitted serially, component 0 is transmitted first. Consequently, block transfers to and from memory are performed starting with the lowest (most negative) address and ending with the highest (most positive) one.

In diagrams, the least significant component of a value is to the right hand side of the diagram. Component 0 of an array is at the bottom of a diagram, as are the most negative memory locations.

### Signal naming conventions

Signal names identifying individual pins of a transputer chip have been chosen to avoid being cryptic, giving as much information as possible. The majority of transputer signals are active high. Those which are active low have names commencing with **not**; names such as **RnotW** imply that the first component of the name refers to its active high state and the second to its active low state. Capitals are used to introduce new components of a name, as in **ProcClockOut**.

All transputer signals described in the text of this databook are printed in **bold**. Registers and flags internal to a device are printed in *italics*, as are instruction operation codes. *Italics* are also used for emphasis. OCCAM program notation is printed in a **fixed space teletype** style.

### References

The databook is divided into several *chapters*, each chapter having a number of *sections* and *subsections*. Figures and tables have reference numbers tied to relevant sections of a particular chapter of the databook. Unless otherwise stated, all references refer to those within the current chapter of the databook.

### Transputer product numbers

All INMOS products, both memories and transputers, have a part number of the general form

### IMS abbbc-xyyz

Field  $\bf a$  identifies the product group. This is a digit for memory products and a letter for other devices, the particular letter indicating the type of product (table 1). Field  $\bf bbb$  identifies the product within that group and field  $\bf c$  is its revision code. Field  $\bf x$  denotes the package type, whilst field  $\bf y$  indicates speed variants etc. The final field  $\bf z$  indicates to which specification the component is qualified; standard, military etc. Where appropriate some identifiers may be omitted, depending on the device.

A typical product part would be IMS T800C-G20S.

Table 1 INMOS products

IMS 1	Static RAM products
IMS A	Digital signal processors
IMS B	PC boards and modular hardware
IMS C	Communications adaptors
IMS D	Development system
IMS G	Graphics products
IMS L	Literature
IMS M	Peripheral control transputers
IMS P	occam programming system
IMS S	Software product
IMS T	Transputers



### **INMOS**

INMOS is a recognised leader in the development and design of high-performance integrated circuits and is a pioneer in the field of parallel processing. The company manufactures components designed to satisfy the most demanding of current processing applications and also provide an upgrade path for future applications. Current designs and development will meet the requirements of systems in the next decade. Computing requirements essentially include high-performance, flexibility and simplicity of use. These characteristics are central to the design of all INMOS products.

INMOS has a consistent record of innovation over a wide product range and supplies components to system manufacturing companies in the United States, Europe, Japan and the Far East. As developers of the Transputer, a unique microprocessor concept with a revolutionary architecture, and the OCCAM parallel processing language, INMOS has established the standards for the future exploitation of the power of parallel processing. INMOS products include a range of transputer products in addition to a highly successful range of high-performance graphics devices, an innovative and successful range of high-performance digital signal processing (DSP) devices and a broad range of fast static RAMs, an area in which it has achieved a greater than 10% market share.

The corporate headquarters, product design team and worldwide sales and marketing management are based at Bristol, UK.

INMOS is constantly upgrading, improving and developing its product range and is committed to maintaining a global position of innovation and leadership.

### 1.1 Manufacturing

INMOS products are manufactured at the INMOS Newport, Duffryn facility which began operations in 1983. This is an 8000 square metre building with a 3000 square metre cleanroom operating to Class 10 environment in the work areas.

To produce high performance products, where each microchip may consist of up to 400,000 transistors, INMOS uses advanced manufacturing equipment. Wafer steppers, plasma etchers and ion implanters form the basis of fabrication.

### 1.2 Assembly

Sub-contractors in Korea, Taiwan, Hong Kong and the UK are used to assemble devices.

### 1.3 Test

The final testing of commercial products is carried out at the INMOS Newport, Coed Rhedyn facility. Military final testing takes place at Colorado Springs.

### 1.4 Quality and Reliability

Stringent controls of quality and reliability provide the customer with early failure rates of less than 1000 ppm and long term reliability rates of better than 100 FITs (one FIT is one failure per 1000 million hours). Requirements for military products are even more stringent.

### 1.5 Military

Various INMOS products are already available in military versions processed in full compliance with MIL-STD-883C. Further military programmes are currently in progress.

### 1.6 Future Developments

### 1.6.1 Research and Development

INMOS has achieved technical success based on a position of innovation and leadership in products and process technology in conjunction with substantial research and development investment. This investment has averaged 18% of revenues since inception and it is anticipated that future investment will be increased.

### 1.6.2 Process Developments

One aspect of the work of the Technology Development Group at Newport is to scale the present 1.2 micron technology to 1.0 micron for products to be manufactured in 1989/90. In addition, work is in progress on the development of 0.8 micron CMOS technology.

1 INMOS



# transputer architecture

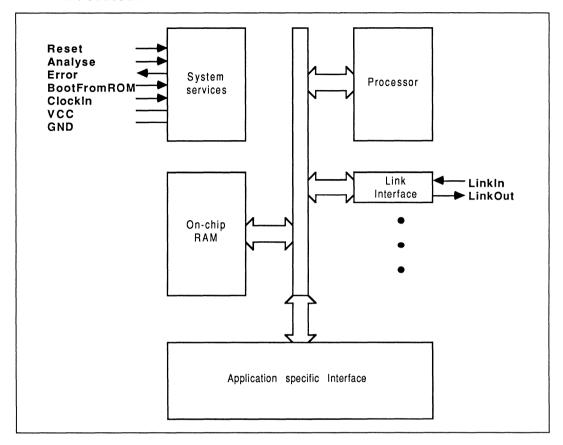


Figure 1.1 Transputer architecture

### 1.1 Overview

A transputer is a microcomputer with its own local memory and with links for connecting one transputer to another transputer.

The transputer architecture defines a family of programmable VLSI components. The definition of the architecture falls naturally into the *logical* aspects which define how a system of interconnected transputers is designed and programmed, and the *physical* aspects which define how transputers, as VLSI components, are interconnected and controlled.

A typical member of the transputer product family is a single chip containing processor, memory, and communication links which provide point to point connection between transputers. In addition, each transputer product contains special circuitry and interfaces adapting it to a particular use. For example, a peripheral control transputer, such as a graphics or disk controller, has interfaces tailored to the requirements of a specific device.

A transputer can be used in a single processor system or in networks to build high performance concurrent systems. A network of transputers and peripheral controllers is easily constructed using point-to-point communication.

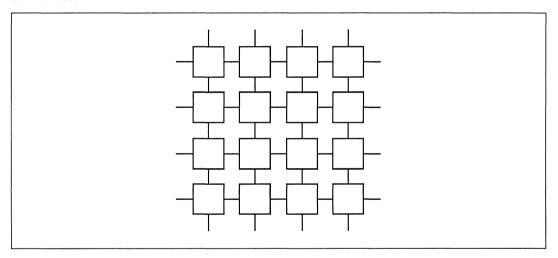


Figure 1.2 Transputer network

### Transputers and occam

Transputers can be programmed in most high level languages, and are designed to ensure that compiled programs will be efficient. Where it is required to exploit concurrency, but still to use standard languages, occam can be used as a harness to link modules written in the selected languages.

To gain most benefit from the transputer architecture, the whole system can be programmed in OCCam (pages 12, 29). This provides all the advantages of a high level language, the maximum program efficiency and the ability to use the special features of the transputer.

OCCAM provides a framework for designing concurrent systems using transputers in just the same way that boolean algebra provides a framework for designing electronic systems from logic gates. The system designer's task is eased because of the architectural relationship between OCCAM and the transputer. A program running in a transputer is formally equivalent to an OCCAM process, so that a network of transputers can be described directly as an OCCAM program.

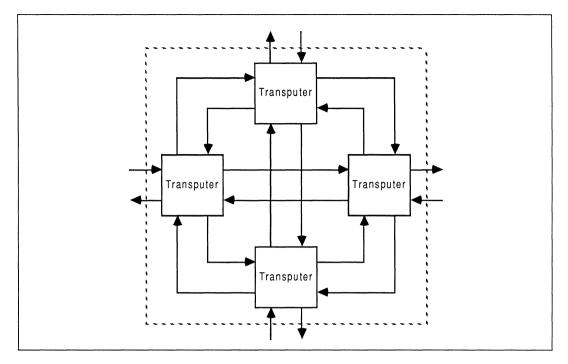


Figure 1.3 A node of four transputers

### 1.2 System design rationale

The transputer architecture simplifies system design by the use of processes as standard software and hardware building blocks.

An entire system can be designed and programmed in OCCam, from system configuration down to low level I/O and real time interrupts.

### 1.2.1 Programming

The software building block is the process. A system is designed in terms of an interconnected set of processes. Each process can be regarded as an independent unit of design. It communicates with other processes along point-to-point channels. Its internal design is hidden, and it is completely specified by the messages it sends and receives. Communication between processes is synchronized, removing the need for any separate synchronisation mechanism.

Internally, each process can be designed as a set of communicating processes. The system design is therefore hierarchically structured. At any level of design, the designer is concerned only with a small and manageable set of processes.

occam is based on these concepts, and provides the definition of the transputer architecture from the logical point of view (pages 12, 29).

### 1.2.2 Hardware

Processes can be implemented in hardware. A transputer, executing an OCCam program, is a hardware process. The process can be independently designed and compiled. Its internal structure is hidden and it communicates and synchronizes with other transputers via its links, which implement OCCam channels.

Other hardware implementations of the process are possible. For example, a transputer with a different instruction set may be used to provide a different cost/performance trade-off. Alternatively, an implementation of the process may be designed in terms of hard-wired logic for enhanced performance.

The ability to specify a hard-wired function as an OCCAM process provides the architectural framework for transputers with specialized capabilities (e.g., graphics). The required function (e.g., a graphics drawing and display engine) is defined as an OCCAM process, and implemented in hardware with a standard OCCAM channel interface. It can be simulated by an OCCAM implementation, which in turn can be used to test the application on a development system.

### 1.2.3 Programmable components

A transputer can be programmed to perform a specialized function, and be regarded as a 'black box' thereafter. Some processes can be hard-wired for enhanced performance.

A system, perhaps constructed on a single chip, can be built from a combination of software processes, preprogrammed transputers and hardware processes. Such a system can, itself, be regarded as a component in a larger system.

The architecture has been designed to permit a network of programmable components to have any desired topology, limited only by the number of links on each transputer. The architecture minimizes the constraints on the size of such a system, and the hierarchical structuring provided by occam simplifies the task of system design and programming.

The result is to provide new orders of magnitude of performance for any given application, which can now exploit the concurrency provided by a large number of programmable components.

### 1.3 Systems architecture rationale

### 1.3.1 Point to point communication links

The transputer architecture simplifies system design by using point to point communication links. Every member of the transputer family has one or more standard links, each of which can be connected to a link of some other component. This allows transputer networks of arbitrary size and topology to be constructed.

Point to point communication links have many advantages over multi-processor buses:

There is no contention for the communication mechanism, regardless of the number of transputers in the system.

There is no capacitive load penalty as transputers are added to a system.

The communications bandwidth does not saturate as the size of the system increases. Rather, the larger the number of transputers in the system, the higher the total communications bandwidth of the system. However large the system, all the connections between transputers can be short and local.

### 1.3.2 Local memory

Each transputer in a system uses its own local memory. Overall memory bandwidth is proportional to the number of transputers in the system, in contrast to a large global memory, where the additional processors must share the memory bandwidth.

Because memory interfaces are not shared, and are separate from the communications interfaces, they can be individually optimized on different transputer products to provide high bandwidth with the minimum of external components.

### 1.4 Communication

To provide synchronised communication, each message must be acknowledged. Consequently, a link requires at least one signal wire in each direction.

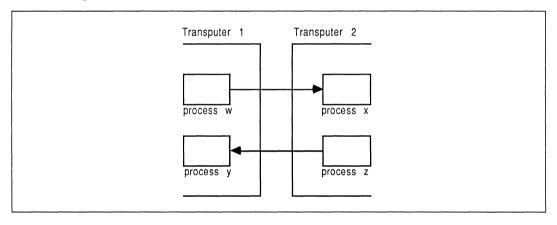


Figure 1.4 Links communicating between processes

A link between two transputers is implemented by connecting a link interface on one transputer to a link interface on the other transputer by two one-directional signal lines, along which data is transmitted serially.

The two signal wires of the link can be used to provide two OCCam channels, one in each direction. This requires a simple protocol. Each signal line carries data and control information.

The link protocol provides the synchronized communication of OCCam. The use of a protocol providing for the transmission of an arbitrary sequence of bytes allows transputers of different word length to be connected.

Each message is transmitted as a sequence of single byte communications, requiring only the presence of a single byte buffer in the receiving transputer to ensure that no information is lost. Each byte is transmitted as a start bit followed by a one bit followed by the eight data bits followed by a stop bit. After transmitting a data byte, the sender waits until an acknowledge is received; this consists of a start bit followed by a zero bit. The acknowledge signifies both that a process was able to receive the acknowledged byte, and that the receiving link is able to receive another byte. The sending link reschedules the sending process only after the acknowledge for the final byte of the message has been received.

Data bytes and acknowledges are multiplexed down each signal line. An acknowledge can be transmitted as soon as reception of a data byte starts (if there is room to buffer another one). Consequently transmission may be continuous, with no delays between data bytes.

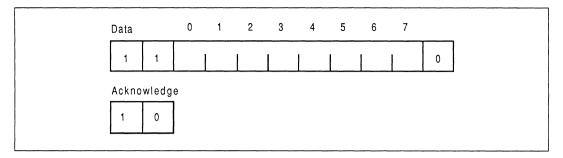


Figure 1.5 Link protocol

The links are designed to make the engineering of transputer systems straightforward. Board layout of two wire connections is easy to design and area efficient. All transputers will support a standard communications frequency of 10 Mbits/sec, regardless of processor performance. Thus transputers of different performance can be directly connected and future transputer systems will directly communicate with those of today.

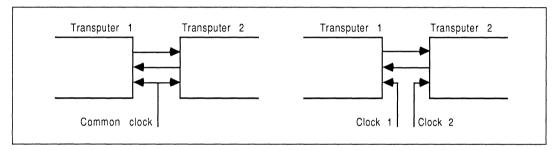


Figure 1.6 Clocking transputers

Link communication is not sensitive to clock phase. Thus, communication can be achieved between independently clocked systems as long as the communications frequency is the same.

The transputer family includes a number of link adaptor devices which provide a means of interfacing transputer links to non-transputer devices.

### 2 occam model

The programming model for transputers is defined by OCCAM (page 29). The purpose of this section is to describe how to access and control the resources of transputers using OCCAM. A more detailed description is available in the OCCAM programming manual and the transputer development system manual (provided with the development system).

The transputer development system will enable transputers to be programmed in other industry standard languages. Where it is required to exploit concurrency, but still to use standard languages, occam can be used as a harness to link modules written in the selected languages.

### 2.1 Overview

In OCCam processes are connected to form concurrent systems. Each process can be regarded as a black box with internal state, which can communicate with other processes using point to point communication channels. Processes can be used to represent the behaviour of many things, for example, a logic gate, a microprocessor, a machine tool or an office.

The processes themselves are finite. Each process starts, performs a number of actions and then terminates. An action may be a set of sequential processes performed one after another, as in a conventional programming language, or a set of parallel processes to be performed at the same time as one another. Since a process is itself composed of processes, some of which may be executed in parallel, a process may contain any amount of internal concurrency, and this may change with time as processes start and terminate.

Ultimately, all processes are constructed from three primitive processes - assignment, input and output. An assignment computes the value of an expression and sets a variable to the value. Input and output are used for communicating between processes. A pair of concurrent processes communicate using a one way channel connecting the two processes. One process outputs a message to the channel and the other process inputs the message from the channel.

The key concept is that communication is synchronized and unbuffered. If a channel is used for input in one process, and output in another, communication takes place when both processes are ready. The value to be output is copied from the outputting process to the inputting process, and the inputting and outputting processes then proceed. Thus communication between processes is like the handshake method of communication used in hardware systems.

Since a process may have internal concurrency, it may have many input channels and output channels performing communication at the same time.

Every transputer implements the OCCam concepts of concurrency and communication. As a result, OCCam can be used to program an individual transputer or to program a network of transputers. When OCCam is used to program an individual transputer, the transputer shares its time between the concurrent processes and channel communication is implemented by moving data within the memory. When OCCam is used to program a network of transputers, each transputer executes the process allocated to it. Communication between OCCam processes on different transputers is implemented directly by transputer links. Thus the same OCCam program can be implemented on a variety of transputer configurations, with one configuration optimized for cost, another for performance, or another for an appropriate balance of cost and performance.

The transputer and OCCAM were designed together. All transputers include special instructions and hardware to provide maximum performance and optimal implementations of the OCCAM model of concurrency and communications.

All transputer instruction sets are designed to enable simple, direct and efficient compilation of occam. Programming of I/O, interrupts and timing is standard on all transputers and conforms to the occam model.

Different transputer variants may have different instruction sets, depending on the desired balance of cost, performance, internal concurrency and special hardware. The OCCAM level interface will, however, remain standard across all products.

occam model 13

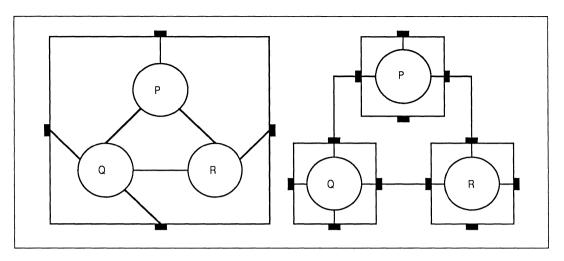


Figure 2.1 Mapping processes onto one or several transputers

# 2.2 occam overview

#### 2.2.1 Processes

After it starts execution, a process performs a number of actions, and then either stops or terminates. Each action may be an assignment, an input, or an output. An assignment changes the value of a variable, an input receives a value from a channel, and an output sends a value to a channel.

At any time between its start and termination, a process may be ready to communicate on one or more of its channels. Each channel provides a one way connection between two concurrent processes; one of the processes may only output to the channel, and the other may only input from it.

# Assignment

An assignment is indicated by the symbol :=. The example

$$v := e$$

sets the value of the variable  $\mathbf{v}$  to the value of the expression  $\mathbf{e}$  and then terminates, for example:  $\mathbf{x} := \mathbf{0}$  sets  $\mathbf{x}$  to zero, and  $\mathbf{x} := \mathbf{x} + \mathbf{1}$  increases the value of  $\mathbf{x}$  by 1.

#### Input

An input is indicated by the symbol? The example

с? х

inputs a value from the channel c, assigns it to the variable x and then terminates.

# Output

An output is indicated by the symbol! The example

c ! e

outputs the value of the expression e to the channel c.

# 2.2.2 Constructions

A number of processes can be combined to form a construct. A construct is itself a process and can therefore be used as a component of another construct. Each component process of a construct is written two spaces further from the left hand margin, to indicate that it is part of the construct. There are four classes of constructs namely the sequential, parallel, conditional and the alternative construct.

# Sequence

A sequential construct is represented by

```
SEQ
P1
P2
P3
```

The component processes P1, P2, P3 ... are executed one after another. Each component process starts after the previous one terminates and the construct terminates after the last component process terminates. For example

```
SEQ
c1 ? x
x := x + 1
c2 ! x
```

inputs a value, adds one to it, and then outputs the result.

Sequential constructs in OCCAM are similar to programs written in conventional programming languages. Note, however, that they provide the performance and efficiency equivalent to that of an assembler for a conventional microprocessor.

#### Parallel

A parallel construct is represented by

```
PAR
P1
P2
P3
```

The component processes **P1**, **P2**, **P3** ... are executed together, and are called concurrent processes. The construct terminates after all of the component processes have terminated, for example:

```
PAR
c1 ? x
c2 ! y
```

allows the communications on channels c1 and c2 to take place together.

The parallel construct is unique to OCCam. It provides a straightforward way of writing programs which directly reflects the concurrency inherent in real systems. The implementation of parallelism on a single transputer is highly optimized so as to incur minimal process scheduling overhead.

2 occam model 15

#### Communication

Concurrent processes communicate only by using channels, and communication is synchronized. If a channel is used for input in one process, and output in another, communication takes place when both the inputting and the outputting processes are ready. The value to be output is copied from the outputting process to the inputting process, and the processes then proceed.

Communication between processes on a single transputer is via memory-to-memory data transfer. Between processes on different transputers it is via standard links. In either case the OCCAM program is identical.

#### Conditional

A conditional construct

```
IF
condition1
P1
condition2
P2
```

means that **P1** is executed if **condition1** is true, otherwise **P2** is executed if **condition2** is true, and so on. Only one of the processes is executed, and then the construct terminates, for example:

```
IF

x = 0

y := y + 1

x <> 0

SKIP
```

increases y only if the value of x is 0.

# Alternation

An alternative construct

```
ALT
input1
P1
input2
P2
input3
P3
```

waits until one of input1, input2, input3 ... is ready. If input1 first becomes ready, input1 is performed, and then process P1 is executed. Similarly, if input2 first becomes ready, input2 is performed, and then process P2 is executed. Only one of the inputs is performed, then its corresponding process is executed and then the construct terminates, for example:

```
ALT
count ? signal
counter := counter + 1
total ? signal
SEQ
out ! counter
counter := 0
```

either inputs a signal from the channel **count**, and increases the variable **counter** by 1, or alternatively inputs from the channel **total**, outputs the current value of the counter, then resets it to zero.

The **ALT** construct provides a formal language method of handling external and internal events that must be handled by assembly level interrupt programming in conventional microprocessors.

# Loop

repeatedly executes the process P until the value of the condition is false, for example:

WHILE 
$$(x - 5) > 0$$
  
 $x := x - 5$ 

leaves x holding the value of (x remainder 5) if x were positive.

# Selection

A selection construct

means that P1 is executed if s has the same value as n, otherwise P2 is executed if s has the same value as n or n0, and so on, for example:

```
CASE direction
    up
        x := x + 1
    down
        x := x - 1
```

increases the value of x if **direction** is equal to **up**, otherwise if **direction** is equal to **down** the value of x is decreased.

# Replication

A replicator is used with a SEQ, PAR, IF or ALT construction to replicate the component process a number of times. For example, a replicator can be used with SEQ to provide a conventional loop.

causes the process  $\mathbf{P}$  to be executed  $\mathbf{n}$  times.

A replicator may be used with PAR to construct an array of concurrent processes.

constructs an array of n similar processes P0, P1, ..., Pn-1. The index i takes the values 0, 1, ..., n-1, in P0, P1, ..., Pn-1 respectively.

2 occam model 17

# 2.2.3 Types

Every variable, expression and value has a type, which may be a primitive type, array type, record type or variant type. The type defines the length and interpretation of data.

All implementations provide the primitive types shown in table 2.1.

Table 2.1 Types

p	
CHAN OF protocol	Each communication channel provides communication between two concurrent processes. Each channel is of a type which allows communication of data according to the specified protocol.
TIMER	Each timer provides a clock which can be used by any number of concurrent processes.
BOOL	The values of type BOOL are true and false.
BYTE	The values of type BYTE are unsigned numbers $\mathbf{n}$ in the range $0 <= \mathbf{n} < 256$ .
INT	Signed integers $\mathbf{n}$ in the range $-2^{31} <= \mathbf{n} < 2^{31}$ .
INT16	Signed integers <b>n</b> in the range $-2^{15} <= \mathbf{n} < 2^{15}$ .
INT32	Signed integers <b>n</b> in the range $-2^{31} <= \mathbf{n} < 2^{31}$ .
INT64	Signed integers <b>n</b> in the range $-2^{63} <= \mathbf{n} < 2^{63}$ .
REAL32	Floating point numbers stored using a sign bit, 8 bit exponent and 23 bit fraction in ANSI/IEEE Standard 754-1985 representation. <sup>1</sup>
REAL64	Floating point numbers stored using a sign bit, 11 bit exponent and 52 bit fraction in ANSI/IEEE Standard 754-1985 representation. <sup>1</sup>

# 2.2.4 Declarations, arrays and subscripts

A declaration **T x** declares **x** as a new channel, variable, timer or array of type **T**, for example:

declares x as an integer variable for use in process P.

Array types are constructed from component types. For example  $[ n ] \mathbf{T}$  is an array type constructed from  $\mathbf{n}$  components of type  $\mathbf{T}$ .

A component of an array may be selected by subscription, for example  $\mathbf{v}[\mathbf{e}]$  selects the  $\mathbf{e}$ 'th component of  $\mathbf{v}$ .

A set of components of an array may be selected by subscription, for example [ $\mathbf{v}$  FROM  $\mathbf{e}$  FOR  $\mathbf{c}$ ] selects the  $\mathbf{c}$  components  $\mathbf{v}[\mathbf{e}]$ ,  $\mathbf{v}[\mathbf{e}+\mathbf{1}]$ , ...  $\mathbf{v}[\mathbf{e}+\mathbf{c}-\mathbf{1}]$ . A set of components of an array may be assigned, input or output.

<sup>&</sup>lt;sup>1</sup>IEEE Standard for Binary Floating-Point arithmetic ANSI/IEEE Std 754-1985

# 2.2.5 Procedures

A process may be given a name, for example:

```
PROC square (INT n)
    n := n * n
:
```

defines the procedure square. The name may be used as an instance of the process, for example:

```
square (x)
```

is equivalent to

```
n IS x:
n := n * n
```

# 2.2.6 Functions

A function can be defined in the same way as a procedure. For example:

```
INT FUNCTION factorial (VAL INT n)
  INT product:
  VALOF
   IF
        n >= 0
        SEQ
        product := 1
        SEQ i = 1 FOR n
        product := product * i
  RESULT product
.
```

defines the function factorial, which may appear in expressions such as

```
m := factorial (6)
```

# 2.2.7 Expressions

An expression is constructed from the operators given in table 2.2, from variables, numbers, the truth values **TRUE** and **FALSE**, and the brackets ( and ).

Operator	Operand types	Description
+ - * / REM	integer, real	arithmetic operators
PLUS MINUS TIMES AFTER	integer	modulo arithmetic
= <>	any primitive	relational operators
> < >= <=	integer, real	relational operators
AND OR NOT	boolean	boolean operators
/\ \/ >< ~	integers	bitwise operators: and, or, xor, not
<< >>	integer	shift operators

Table 2.2 Operators

2 OCCAM model 19

For example, the expression

$$(5 + 7) / 2$$

evaluates to 6, and the expression

evaluates to #D (the character # introduces a hexadecimal constant).

A string is represented as a sequence of ASCII characters, enclosed in double quotation marks ". If the string has n characters, then it is an array of type [n]BYTE.

#### 2.2.8 Timer

All transputers incorporate a timer. The implementation directly supports the OCCAM model of time. Each process can have its own independent timer, which can be used for internal measurement or for real time scheduling.

A timer input sets a variable to a value of type **INT** representing the time. The value is derived from a clock, which changes at regular intervals, for example:

sets the variable v to the current value of a free running clock, declared as the timer tim.

A delayed input takes the following form

A delayed input is unable to proceed until the value of the timer satisfies (*timer* **AFTER** *e*). The comparison performed is a modulo comparison. This provides the effect that, starting at any point in the timer's cycle, the previous half cycle of the timer is considered as being before the current time, and the next half cycle is considered as being after the current time.

# 2.2.9 Peripheral access

The implementation of OCCAM provides for peripheral access by extending the input and output primitives with a port input/output mechanism. A port is used like an OCCAM channel, but has the effect of transferring information to and from a block of addresses associated with a peripheral.

Ports behave like OCCam channels in that only one process may input from a port, and only one process may output to a port. Thus ports provide a secure method of accessing external memory mapped status registers etc.

Note that there is no synchronization mechanism associated with port input and output. Any timing constraints which result from the use of asynchronous external hardware will have to be programmed explicitly. For example, a value read by a port input may depend upon the time at which the input was executed, and inputting at an invalid time would produce unusable data.

During applications development it is recommended that the peripheral is modelled by an OCCAM process connected via channels.

# 2.3 Configuration

occam programs may be configured for execution on one or many transputers. The transputer development system provides the necessary tools for correctly distributing a program configured for many transputers.

Configuration does not affect the logical behaviour of a program (see section four, Program development). However, it does enable the program to be arranged to ensure that performance requirements are met.

#### PLACED PAR

A parallel construct may be configured for a network of transputers by using the **PLACED PAR** construct. Each component process (termed a placement) is executed by a separate transputer. The variables and timers used in a placement must be declared within each placement process.

#### PRI PAR

On any individual transputer, the outermost parallel construct may be configured to prioritize its components. Each process is executed at a separate priority. The first process has the highest priority, the last process has the lowest priority. Lower priority components may only proceed when all higher priority components are unable to proceed.

# 2.3.1 INMOS standard links

Each link provides one channel in each direction between two transputers.

A channel (which must already have been declared) is associated with a link by a channel association, for example:

# PLACE LinkOInput AT 4 :

# 3 Error handling

Errors in OCCam programs are either detected by the compiler or can be handled at runtime in one of three ways.

- 1 Cause the process to **STOP** allowing other processes to continue.
- 2 Cause the whole system to halt.
- 3 Have an arbitrary (undefined) effect.

The OCCAM process STOP starts but never terminates. In method 1, an errant process stops and in particular cannot communicate erroneous data to other processes. Other processes will continue to execute until they become dependent on data from the stopped process. It is therefore possible, for example, to write a process which uses a timeout to warn of a stopped process, or to construct a redundant system in which several processes performing the same task are used to enable the system to continue after one of them has failed.

Method 1 is the preferred method of executing a program.

Method 2 is useful for program development and can be used to bring transputers to an immediate halt, preventing execution of further instructions. The transputer **Error** output can be used to inform the transputer development system that such an error has occurred. No variable local to the process can be overwritten with erroneous data, facilitating analysis of the program and data which gave rise to the error.

Method 3 is useful only for optimising programs which are known to be correct!

When a system has stopped or halted as a result of an error, the state of all transputers in the system can be analysed using the transputer development system.

For languages other than OCCam, the transputer provides facilities for handling individual errors by software.

# 4 Program development

The development of programs for multiple processor systems can involve experimentation. In some cases, the most effective configuration is not always clear until a substantial amount of work has been done. For this reason, it is desirable that most of the design and programming can be completed before hardware construction is started.

# 4.1 Logical behaviour

An important property of OCCAM in this context is that it provides a clear notion of 'logical behaviour'; this relates to those aspects of a program not affected by real time effects.

It is guaranteed that the logical behaviour of a program is not altered by the way in which the processes are mapped onto processors, or by the speed of processing and communication. Consequently a program ultimately intended for a network of transputers can be compiled, executed and tested on a single computer used for program development.

Even if the application uses only a single transputer, the program can be designed as a set of concurrent processes which could run on a number of transputers. This design style follows the best traditions of structured programming; the processes operate completely independently on their own variables except where they explicitly interact, via channels. The set of concurrent processes can run on a single transputer or, for a higher performance product, the processes can be partitioned amongst a number of transputers.

It is necessary to ensure, on the development system, that the logical behaviour satisfies the application requirements. The only ways in which one execution of a program can differ from another in functional terms result from dependencies upon input data and the selection of components of an **ALT**. Thus a simple method of ensuring that the application can be distributed to achieve any desired performance is to design the program to behave 'correctly' regardless of input data and **ALT** selection.

#### 4.2 Performance measurement

Performance information is useful to gauge overall throughput of an application, and has to be considered carefully in applications with real time constraints.

Prior to running in the target environment, an OCCam program should be relatively mature, and indeed should be correct except for interactions which do not obey the OCCam synchronization rules. These are precisely the external interactions of the program where the world will not wait to communicate with an OCCam process which is not ready. Thus the set of interactions that need to be tested within the target environment are well identified.

Because, in OCCAM, every program is a process, it is extremely easy to add monitor processes or simulation processes to represent parts of the real time environment, and then to simulate and monitor the anticipated real time interactions. The OCCAM concept of time and its implementation in the transputer is important. Every process can have an independent timer enabling, for example, all the real time interactions to be modelled by separate processes and any time dependent features to be simulated.

# 4.3 Separate compilation of OCCAM and other languages

A program portion which is separately compiled, and possibly written in a language other than OCCAM, may be executed on a single transputer.

If the program is written in OCCAM, then it takes the form of a single PROC, with only channel parameters. If the program is written in a language other than OCCAM, then a run-time system is provided which provides input/output to OCCAM channels.

Such separately compiled program portions are linked together by a framework of channels, termed a harness. The harness is written in occam. It includes all configuration information, and in particular specifies the transputer configuration in which the separately compiled program portion is executed.

Transputers are designed to allow efficient implementations of high level languages, such as C, Pascal and Fortran. Such languages will be available in addition to OCCAM.

At runtime, a program written in such a language is treated as a single OCCAM process. Facilities are provided in the implementations of these languages to allow such a program to communicate on OCCAM channels. It can thus communicate with other such programs, or with programs written in OCCAM. These programs may reside on the same transputer, in which case the channels are implemented in store, or may reside on different transputers, in which case the channels are implemented by transputer links.

It is therefore possible to implement OCCam processes in conventional high level languages, and arrange for them to communicate. It is possible for different parts of the same application to be implemented in different high level languages.

The standard input and output facilities provided within these languages are implemented by a well-defined protocol of communications on OCCAM channels.

The development system provides facilities for management of separately compiled OCCAM.

# 4.4 Memory map and placement

The low level memory model is of a signed address space.

Memory is byte addressed, the lowest addressed byte occupying the least significant byte position within the word.

The implementation of OCCam supports the allocation of the code and data areas of an OCCam process to specific areas of memory. Such a process must be a separately compiled PROC, and must not reference any variables and timers other than those declared within it.

# 5 Physical architecture

# 5.1 INMOS serial links

# 5.1.1 Overview

All transputers have several links. The link protocol and electrical characteristics form a standard for all INMOS transputer and peripheral products.

All transputers support a standard link communications frequency of 10 Mbits/sec. Some devices also support other data rates. Maintaining a standard communications frequency means that devices of mixed performance and type can intercommunicate easily.

Each link consists of two unidirectional signal wires carrying both data and control bits. The link signals are TTL compatible so that their range can be easily extended by inserting buffers.

The INMOS communication links provide for communication between devices on the same printed circuit board or between printed circuit boards via a back plane. They are intended to be used in electrically quiet environments in the same way as logic signals between TTL gates.

The number of links, and any communication speeds in addition to the standard speed of 10 Mbits/sec, are given in the **product data** for each product.

#### 5.1.2 Link electrical specification

The quiescent state of the link signals is low, for a zero. The link input signals and output signals are standard TTL compatible signals.

For correct functioning of the links the specifications for maximum variation in clock frequency between two transputers joined by a link and maximum capacitive load must be met. Each transputer product also has specified the maximum permissible variation in delay in buffering, and minimum permissible edge gradients. Details of these specifications are provided in the product data.

Provided that these specifications are met then any buffering employed may introduce an arbitrary delay into a link signal without affecting its correct operation.

# 5.2 System services

# 5.2.1 Powering up and down, running and stopping

At all times the specification of input voltages with respect to the GND and VCC pins must be met. This includes the times when the VCC pins are ramping to 5 V, and also while they are ramping from 5 V down to 0 V.

The system services comprise the clocks, power, and signals used for initialization.

The specification includes minimum times that **VCC** must be within specification, the input clock must be oscillating, and the **Reset** signal must be high before **Reset** goes low. These specifications ensure that internal clocks and logic have settled before the transputer starts.

When the transputer is reset the memory interface is initialised (if present and configurable).

The processor and INMOS serial links start after reset. The transputer obeys a bootstrap program which can either be in off-chip ROM or can be received from one of the links. How to specify where the bootstrap program is taken from depends upon the type of transputer being used. The program will normally load up a larger program either from ROM or from a peripheral such as a disk.

During power down, as during power up, the input and output pins must remain within specification with respect to both GND and VCC.

A software error, such as arithmetic overflow, array bounds violation or divide by zero, causes an error flag to be set in the transputer processor. The flag is directly connected to the **Error** pin. Both the flag and the pin can be ignored, or the transputer stopped. Stopping the transputer on an error means that the error cannot cause further corruption.

As well as containing the error in this way it is possible to determine the state of the transputer and its memory at the time the error occurred.

# 5.2.2 Clock distribution

All transputers operate from a standard 5MHz input clock. High speed clocks are derived internally from the low frequency input to avoid the problems of distributing high frequency clocks. Within limits the mark-to-space ratio, the voltage levels and the transition times are immaterial. The limits on these are given in the product data for each product. The asynchronous data reception of the links means that differences in the clock phase between chips is unimportant.

The important characteristic of the transputer's input clock is its stability, such as is provided by a crystal oscillator. An R-C oscillator is inadequate. The edges of the clock should be monotonic (without kinks), and should not undershoot below -0.5 V.

# 5.3 Bootstrapping from ROM or from a link

The program which is executed after reset can either reside in ROM in the transputer's address space or it can be loaded via any one of the transputer's INMOS serial links.

The transputer bootstraps from ROM by transferring control to the top two bytes in memory, which will invariably contain a backward jump into ROM.

If bootstrapping from a link, the transputer bootstraps from the first link to receive a message. The first byte of the message is the count of the number of bytes of program which follow. The program is loaded into memory starting at a product dependent location *MemStart*, and then control is transferred to this address.

Messages subsequently arriving on other links are not acknowledged until the transputer processor obeys a process which inputs from them. The loading of a network of transputers is controlled by the transputer development system, which ensures that the first message each transputer receives is the bootstrap program.

# 5.4 Peripheral interfacing

All transputers contain one or more INMOS serial links. Certain transputer products also have other application specific interfaces. The peripheral control transputers contain specialized interfaces to control a specific peripheral or peripheral family.

In general, a transputer based application will comprise a number of transputers which communicate using INMOS links. There are three methods of communicating with peripherals.

The first is by employing peripheral control transputers (eg for graphics or disks), in which the transputer chip connects directly to the peripheral concerned (figure 5.1). The interface to the peripheral is implemented by special purpose hardware within the transputer. The application software in the transputer is implemented as an OCCAM process, and controls the interface via OCCAM channels linking the processor to the special purpose hardware.

The second method is by employing link adaptors (figure 5.2). These devices convert between a link and a specialized interface. The link adaptor is connected to the link of an appropriate transputer, which contains the application designer's peripheral device handler implemented as an OCCAM process.

The third method is by memory mapping the peripheral onto the memory bus of a transputer (figure 5.3). The peripheral is controlled by memory accesses issued as a result of **PORT** inputs and outputs. The application designer's peripheral device handler provides a standard OCCAM channel interface to the rest of

the application.

The first transputers implement an event pin which provides a simple means for an external peripheral to request attention from a transputer.

In all three methods, the peripheral driver interfaces to the rest of the application via occam channels. Consequently, a peripheral device can be simulated by an occam process. This enables testing of all aspects of a transputer system before the construction of hardware.

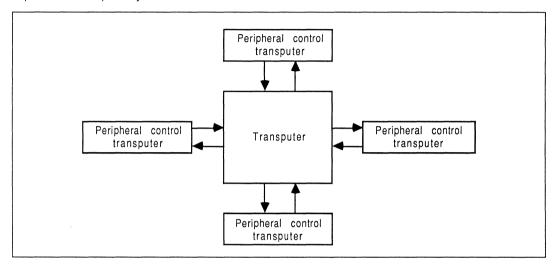


Figure 5.1 Transputer with peripheral control transputers

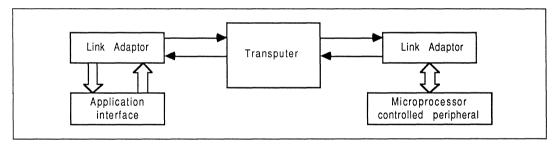


Figure 5.2 Transputer with link adaptors

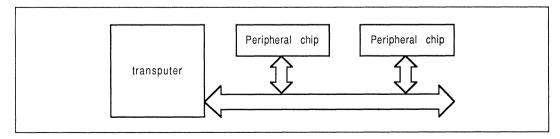


Figure 5.3 Memory mapped peripherals



# transputer overview

# 1 Introduction

The INMOS transputer family is a range of system components each of which combines processing, memory and interconnect in a single VLSI chip. A concurrent system can be constructed from a collection of transputers which operate concurrently and communicate through serial communication links. Such systems can be designed and programmed in OCCAM, a language based on communicating processes, and in other industry standard languages. Transputers have been sucessfully used in application areas ranging from embedded systems to supercomputers.

The first member of the family, the IMS T414 32-bit transputer, was introduced in September 1985, and has enabled concurrency to be applied in a wide variety of applications such as simulation, robot control, image synthesis, and digital signal processing. Many computationally intensive applications can exploit large arrays of transputers; the system performance depending on the number of transputers, the speed of inter-transputer communication and the performance of each transputer processor.

The power of transputer based systems lies in the smoothly scaleable performance offered by adding more transputers. The transputer embodies the concepts required for effective parallel processing.

Further transputer products are continually being developed which increase the memory, processing performance and communications performance. An important example is the floating point transputer first introduced in 1987.

# 2 The transputer: basic architecture and concepts

# 2.1 A programmable device

The transputer is a component designed to exploit the potential of VLSI. This technology allows large numbers of *identical* devices to be manufactured cheaply. For this reason, it is attractive to implement a concurrent system using a number of identical components, each of which is customised by an appropriate program. The transputer is, therefore, a VLSI device with a processor, memory to store the program executed by the processor, and communication links for direct connection to other transputers. Transputer systems can be designed and programmed using OCCAM which allows an application to be described as a collection of processes which operate concurrently and communicate through channels. The transputer can therefore be used as a building block for concurrent processing systems, with OCCAM as the associated design formalism.

# 2.2 occam

occam enables a system to be described as a collection of concurrent processes, which communicate with each other and with peripheral devices through channels. occam programs are built from three primitive processes:

v := e	assign expression <b>e</b> to variable <b>v</b>
c ! e	output expression e to channel c
c?v	input from channel c to variable v

The primitive processes are combined to form constructs:

SEQuential components executed one after another components executed together another component first ready is executed

A construct is itself a process, and may be used as a component of another construct.

Conventional sequential programs can be expressed with variables and assignments, combined in sequential constructs. IF and WHILE constructs are also provided.

Concurrent programs can be expressed with channels, inputs and outputs, which are combined in parallel and alternative constructs.

Each occam channel provides a communication path between two concurrent processes. Communication is synchronised and takes place when both the inputting process and the outputting process are ready. The data to be output is then copied from the outputting process to the inputting process, and both processes continue.

An alternative process may be ready for input from any one of a number of channels. In this case, the input is taken from the channel which is first used for output by another process.

# 2.3 VLSI technology

One important property of VLSI technology is that communication between devices is very much slower than communication within a device. In a computer, almost every operation that the processor performs involves the use of memory. For this reason a transputer includes both processor and memory in the same integrated circuit device.

In any system constructed from integrated circuit devices, much of the physical bulk arises from connections between devices. The size of the package for an integrated circuit is determined more by the number of connection pins than by the size of the device itself. In addition, connections between devices provided by paths on a circuit board consume a considerable amount of space.

The speed of communication between electronic devices is optimised by the use of one-directional signal wires, each connecting two devices. If many devices are connected by a shared bus, electrical problems of driving the bus require that the speed is reduced. Also, additional control logic and wiring are required to control sharing of the bus.

To provide maximum speed with minimal wiring, the transputer uses point-to-point serial communication links for direct connection to other transputers. The protocols used on the transputer links are discussed later.

# 2.4 Simplified processor with micro-coded scheduler

The most effective implementation of simple programs by a programmable computer is provided by a sequential processor. Consequently, the transputer has a fairly conventional microcoded processor. There is a small core of about 32 instructions which are used to implement simple sequential programs. In addition there are other, more specialised groups of instructions which provide facilities such as long arithmetic and process scheduling.

As a process executed by a transputer may itself consist of a number of concurrent processes the transputer has to support the OCCAM programming model internally. The transputer, therefore, has a microcoded scheduler which shares the processor time between the concurrent processes. The scheduler provides two priority levels; any high priority process which can run will do so in preference to any low priority process.

# 3 Transputer internal architecture

Internally, a transputer consists of a memory, processor and communications system connected via a 32-bit bus. The bus also connects to the external memory interface, enabling additional local memory to be used. The processor, memory and communications system each occupy about 25% of the total silicon area, the remainder being used for power distribution, clock generators and external connections.

The floating point transputers each have an on-chip floating point unit. The small size and high performance of this unit come from a design which takes careful note of silicon economics. This contrasts starkly with conventional co-processors, where the floating point unit typically occupies more area than a complete microprocessor, and requires a second chip.

The block diagram 3.1 indicates the way in which the major blocks of the transputer are interconnected.

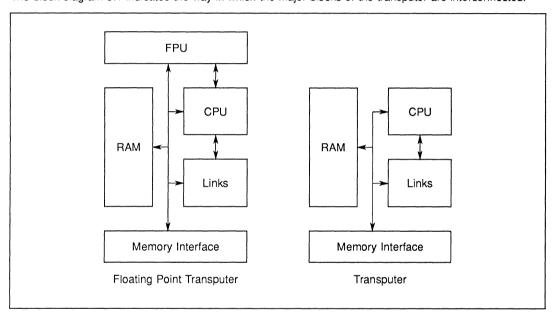


Figure 3.1 Transputer interconnections

The CPU of the transputers contains three registers (A, B and C) used for integer and address arithmetic, which form a hardware stack. Loading a value into the stack pushes B into C, and A into B, before loading A. Storing a value from A pops B into A and C into B. Similarly, the FPU includes a three register floating-point evaluation stack, containing the AF, BF, and CF registers. When values are loaded onto, or stored from the stack the AF, BF and CF registers push and pop in the same way as the A, B and C registers.

The addresses of floating point values are formed on the CPU stack, and values are transferred between the addressed memory locations and the FPU stack under the control of the CPU. As the CPU stack is used only to hold the addresses of floating point values, the wordlength of the CPU is independent of that of the FPU. Consequently, it would be possible to use the same FPU together with a 16-bit CPU.

The transputer scheduler provides two priority levels. The FPU register stack is duplicated so that when the floating point transputer switches from low to high priority none of the state in the floating point unit is written to memory. This results in a worst-case interrupt response of about 3  $\mu$ s. Furthermore, the duplication of the register stack enables floating point arithmetic to be used in an interrupt routine without any performance penalty.

# 3.1 Sequential processing

The design of the transputer processor exploits the availability of fast on-chip memory by having only a small number of registers; the CPU contains six registers which are used in the execution of a sequential process. The small number of registers, together with the simplicity of the instruction set enables the processor to have relatively simple (and fast) data-paths and control logic.

The six registers are:

The workspace pointer which points to an area of store where local variables are kept.

The instruction pointer which points to the next instruction to be executed.

The operand register which is used in the formation of instruction operands.

The A, B and C registers which form an evaluation stack, and are the sources and destinations for most arithmetic and logical operations. Loading a value into the stack pushes B into C, and A into B, before loading A. Storing a value from A, pops B into A and C into B.

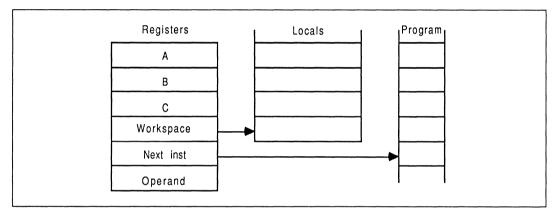


Figure 3.2 Registers

Expressions are evaluated on the evaluation stack, and instructions refer to the stack implicitly. For example, the *add* instruction adds the top two values in the stack and places the result on the top of the stack. The use of a stack removes the need for instructions to respecify the location of their operands. Statistics gathered from a large number of programs show that three registers provide an effective balance between code compactness and implementation complexity.

No hardware mechanism is provided to detect that more than three values have been loaded onto the stack. It is easy for the compiler to ensure that this never happens.

# 3.2 Instructions

It was a design decision that the transputer should be programmed in a high-level language. The instruction set has, therefore, been designed for simple and efficient compilation. It contains a relatively small number of instructions, all with the same format, chosen to give a compact representation of the operations most frequently occuring in programs. The instruction set is independent of the processor wordlength, allowing the same microcode to be used for transputers with different wordlengths. Each instruction consists of a single byte divided into two 4-bit parts. The four most significant bits of the byte are a function code, and the four least significant bits are a data value.

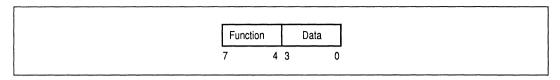


Figure 3.3 Instruction format

#### 3.2.1 Direct functions

The representation provides for sixteen functions, each with a data value ranging from 0 to 15. Thirteen of these are used to encode the most important functions performed by any computer. These include:

load constant	add constant	
load local	store local	load local pointer
load non-local	store non-local	·
jump	conditional jump	call

The most common operations in a program are the loading of small literal values, and the loading and storing of one of a small number of variables. The *load constant* instruction enables values between 0 and 15 to be loaded with a single byte instruction. The *load local* and *store local* instructions access locations in memory relative to the workspace pointer. The first 16 locations can be accessed using a single byte instruction.

The *load non-local* and *store non-local* instructions behave similarly, except that they access locations in memory relative to the A register. Compact sequences of these instructions allow efficient access to data structures, and provide for simple implementations of the static links or displays used in the implementation of block structured programming languages such as OCCam.

# 3.2.2 Prefix functions

Two more of the function codes are used to allow the operand of any instruction to be extended in length. These are:

prefix negative prefix

All instructions are executed by loading the four data bits into the least significant four bits of the operand register, which is then used as the the instruction's operand. All instructions except the prefix instructions end by clearing the operand register, ready for the next instruction.

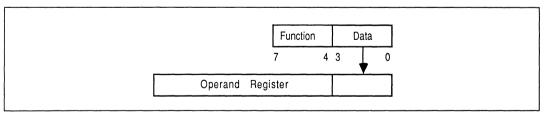


Figure 3.4 Instruction operand register

The *prefix* instruction loads its four data bits into the operand register, and then shifts the operand register up four places. The *negative prefix* instruction is similar, except that it complements the operand register before shifting it up. Consequently operands can be extended to any length up to the length of the operand register by a sequence of prefix instructions. In particular, operands in the range -256 to 255 can be represented using one prefix instruction.

The use of prefix instructions has certain beneficial consequences. Firstly, they are decoded and executed in the same way as every other instruction, which simplifies and speeds instruction decoding. Secondly, they simplify language compilation, by providing a completely uniform way of allowing any instruction to take an operand of any size. Thirdly, they allow operands to be represented in a form independent of the processor wordlength.

# 3.2.3 Indirect functions

The remaining function code, *operate*, causes its operand to be interpreted as an operation on the values held in the evaluation stack. This allows up to 16 such operations to be encoded in a single byte instruction. However, the prefix instructions can be used to extend the operand of an *operate* instruction just like any other. The instruction representation therefore provides for an indefinite number of operations.

The encoding of the indirect functions is chosen so that the most frequently occuring operations are represented without the use of a prefix instruction. These include arithmetic, logical and comparison operations such as

add exclusive or greater than

Less frequently occuring operations have encodings which require a single prefix operation (the transputer instruction set is not large enough to require more than 512 operations to be encoded!).

The IMS T800 has additional instructions which load into, operate on, and store from, the floating point register stack. It also contains new instructions which support colour graphics, pattern recognition and the implementation of error correcting codes. These instructions have been added whilst retaining the existing IMS T414 instruction set. This has been possible because of the extensible instruction encoding used in transputers.

# 3.2.4 Efficiency of encoding

Measurements show that about 70% of executed instructions are encoded in a single byte (ie without the use of prefix instructions). Many of these instructions, such as *load constant* and *add* require just one processor cycle.

The instruction representation gives a more compact representation of high level language programs than more conventional instruction sets. Since a program requires less store to represent it, less of the memory bandwidth is taken up with fetching instructions. Furthermore, as memory is word accessed the processor will receive several instructions for every fetch.

Short instructions also improve the effectiveness of instruction prefetch, which in turn improves processor performance. There is an extra word of prefetch buffer so that the processor rarely has to wait for an instruction fetch before proceeding. Since the buffer is short, there is little time penalty when a jump instruction causes the buffer contents to be discarded.

# 3.3 Support for concurrency

The processor provides efficient support for the OCCAM model of concurrency and communication. It has a microcoded scheduler which enables any number of concurrent processes to be executed together, sharing the processor time. This removes the need for a software kernel. The processor does not need to support the dynamic allocation of storage as the OCCAM compiler is able to perform the allocation of space to concurrent processes.

At any time, a concurrent process may be

active - being executed

on a list waiting to be executed

inactive - ready to input

ready to output

waiting until a specified time

The scheduler operates in such a way that inactive processes do not consume any processor time. The active processes waiting to be executed are held on a list. This is a linked list of process workspaces, implemented using two registers, one of which points to the first process on the list, the other to the last. In figure 3.5, S is executing, and P, Q and R are active, awaiting execution.

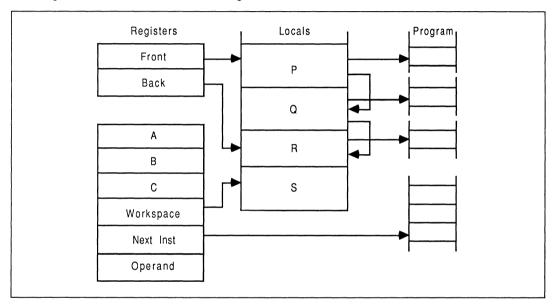


Figure 3.5 Linked process list

A process is executed until it is unable to proceed because it is waiting to input or output, or waiting for the timer. Whenever a process is unable to proceed, its instruction pointer is saved in its workspace and the next process is taken from the list. Actual process switch times are very small as little state needs to be saved; it is not necessary to save the evaluation stack on rescheduling.

The processor provides a number of special operations to support the process model. These include

start process end process

When a parallel construct is executed, *start process* instructions are used to create the necessary concurrent processes. A *start process* instruction creates a new process by adding a new workspace to the end of the scheduling list, enabling the new concurrent process to be executed together with the ones already being executed.

The correct termination of a parallel construct is assured by use of the *end process* instruction. This uses a workspace location as a counter of the components of the parallel construct which have still to terminate. The counter is initialised to the number of components before the processes are 'started'. Each component ends with an *end process* instruction which decrements and tests the counter. For all but the last component, the counter is non zero and the component is descheduled. For the last component, the counter is zero and the component continues.

# 3.4 Communications

Communication between processes is achieved by means of channels. OCCAM communication is point-to-point, synchronised and unbuffered. As a result, a channel needs no process queue, no message queue and no message buffer.

A channel between two processes executing on the same transputer is implemented by a single word in memory; a channel between processes executing on different transputers is implemented by point-to-point links. The processor provides a number of operations to support message passing, the most important being

input message output message

The *input message* and *output message* instructions use the address of the channel to determine whether the channel is internal or external. This means that the same instruction sequence can be used for both hard and soft channels, allowing a process to be written and compiled without knowledge of where its channels are connected.

As in the occam model, communication takes place when both the inputting and outputting processes are ready. Consequently, the process which first becomes ready must wait until the second one is also ready.

A process performs an input or output by loading the evaluation stack with a pointer to a message, the address of a channel, and a count of the number of bytes to be transferred, and then executing an *input message* or an *output message* instruction.

# 3.4.1 Internal channel communication

At any time, an internal channel (a single word in memory) either holds the identity of a process, or holds the special value *empty*. The channel is initialised to *empty* before it is used.

When a message is passed using the channel, the identity of the first process to become ready is stored in the channel, and the processor starts to execute the next process from the scheduling list. When the second process to use the channel becomes ready, the message is copied, the waiting process is added to the scheduling list, and the channel reset to its initial state. It does not matter whether the inputting or the outputting process becomes ready first.

In figure 3.6, a process P is about to execute an output instruction on an 'empty' channel C. The evaluation stack holds a pointer to a message, the address of channel C, and a count of the number of bytes in the message.

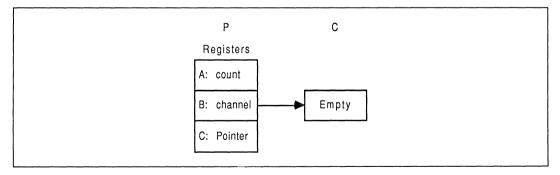


Figure 3.6 Output to empty channel

After executing the output instruction, the channel C holds the address of the workspace of P, and the address of the message to be transferred is stored in the workspace of P. P is descheduled, and the process starts to execute the next process from the scheduling list.

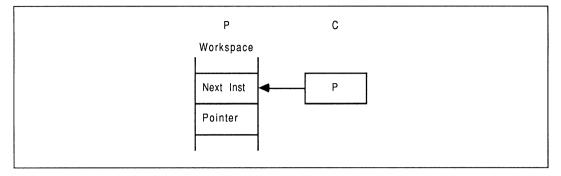


Figure 3.7

The channel C and the process P remain in this state until a second process, Q executes an output instruction on the channel.

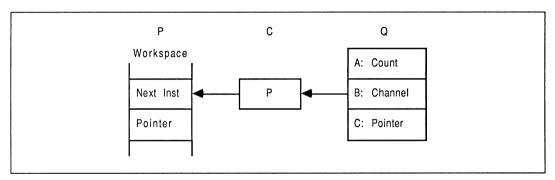


Figure 3.8

The message is copied, the waiting process P is added to the scheduling list, and the channel C is reset to its initial 'empty' state.

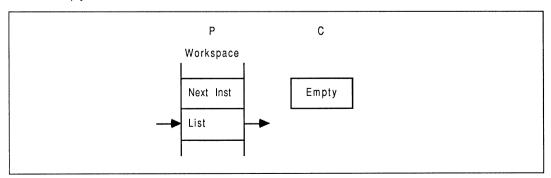


Figure 3.9

#### 3.4.2 External channel communication

When a message is passed via an external channel the processor delegates to an autonomous link interface the job of transferring the message and deschedules the process. When the message has been transferred the link interface causes the processor to reschedule the waiting process. This allows the processor to continue the execution of other processes whilst the external message transfer is taking place.

Each link interface uses three registers:

- a pointer to a process workspace
- a pointer to a message
- a count of bytes in the message

In figure 3.10 processes P and Q executed by different transputers communicate using a channel C implemented by a link connecting two transputers. P outputs, and Q inputs.

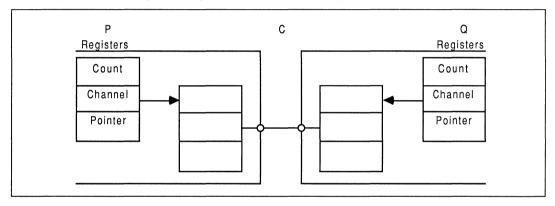


Figure 3.10 Communication between transputers

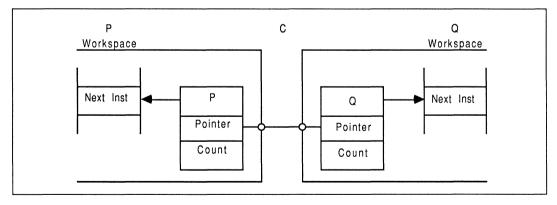


Figure 3.11

When P executes its output instruction, the registers in the link interface of the transputer executing P are initialised, and P is descheduled. Similarly, when Q executes its input instruction, the registers in the link interface of the process executing Q are initialised, and Q is descheduled (figure 3.11).

The message is now copied through the link, after which the workspaces of P and Q are returned to the corresponding scheduling lists (figure 3.12). The protocol used on P and Q ensures that it does not matter which of P and Q first becomes ready.

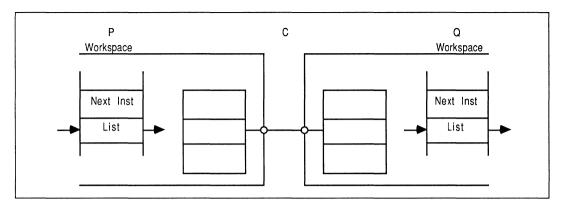


Figure 3.12

#### 3.4.3 Communication links

A link between two transputers is implemented by connecting a link interface on one transputer to a link interface on the other transputer by two one-directional signal wires, along which data is transmitted serially. The two wires provide two OCCam channels, one in each direction. This requires a simple protocol to multiplex data and control information. Messages are transmitted as a sequence of bytes, each of which must be acknowledged before the next is transmitted. A byte of data is transmitted as a start bit followed by a one bit followed by eight bits of data followed by a stop bit. An acknowledgement is transmitted as a start bit followed by a stop bit. An acknowledgement indicates both that a process was able to receive the data byte and that it is able to buffer another byte.

The protocol permits an acknowledgement to be generated as soon as the receiver has identified a data packet. In this way the acknowledgement can be received by the transmitter before all of the data packet has been transmitted and the transmitter can transmit the next data packet immediately. Some transputers do not implement this overlapping and achieve a data rate of 0.8 Mbytes/sec using a link to transfer data in one direction. However, by implementing the overlapping and including sufficient buffering in the link hardware, the rate can be more than doubled to achieve 1.8 Mbytes/sec in one direction, and 2.4 Mbytes/sec when the link carries data in both directions. The diagram below shows the signals that would be observed on the two link wires when a data packet is overlapped with an acknowledgement.

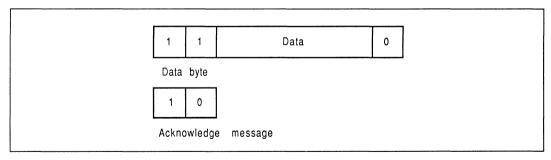


Figure 3.13 Link data and acknowledge formats

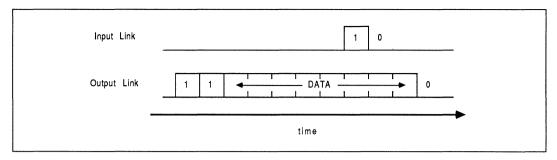


Figure 3.14 Overlapped link acknowledge

# 3.5 Timer

The transputer has a clock which 'ticks' every microsecond. The current value of the processor clock can be read by executing a *read timer* instruction.

A process can arrange to perform a *timer input*, in which case it will become ready to execute after a specified time has been reached.

The *timer input* instruction requires a time to be specified. If this time is in the 'past' (i.e. *ClockReg* **AFTER** *SpecifiedTime*) then the instruction has no effect. If the time is in the 'future' (i.e. *SpecifiedTime* **AFTER** *Clockreg* or *SpecifiedTime* = *ClockReg*) then the process is descheduled. When the specified time is reached the process is scheduled again.

#### 3.6 Alternative

The occam alternative construct enables a process to wait for input from any one of a number of channels, or until a specific time occurs. This requires special instructions, as the normal *input* instruction deschedules a process until a specific channel becomes ready, or until a specific time is reached. The instructions are:

enable channel disable channel enable timer disable timer alternative wait

The alternative is implemented by 'enabling' the channel input or timer input specified in each of its components. The 'alternative wait' is then used to deschedule the process if none of the channel or timer inputs is ready; the process will be re-scheduled when any one of them becomes ready. The channel and timer inputs are then 'disabled'. The 'disable' instructions are also designed to select the component of the alternative to be executed; the first component found to be ready is executed.

# 3.7 Floating point instructions

The core of the floating point instruction set was established fairly early in the design of the floating point transputer. This core includes simple load, store and arithmetic instructions. Examination of statistics derived from FORTRAN programs suggested that the addition of some more complex instructions would improve performance and code density. Proposed changes to the instruction set were assesed by examining their effect on a number of numerical programs. For each proposed instruction set, a compiler was constructed, the programs compiled with it, and the resulting code then run on a simulator. The resulting instruction set is now described.

In the floating point transputer operands are transferred between the transputer's memory and the floating point evaluation stack by means of floating point load and store instructions. There are two groups of such instructions, one for single length numbers, one for double length. In the description of the load and store instructions which follow only the double length instructions are described. However, there are single length

instructions which correspond with each of the double length instructions.

The address of a floating point operand is computed on the CPU's stack and the operand is then loaded, from the addressed memory location, onto the FPU's stack. Operands in the floating point stack are tagged with their length. The operand's tag will be set when the operand is loaded or is computed. The tags allow the number of instructions needed for floating point operations to be reduced; there is no need, for example, to have both *floating add single* and *floating add double* instructions; a single *floating add* will suffice.

# 3.7.1 Optimising use of the stack

The depth of the register stacks in the CPU and FPU is carefully chosen. Floating point expressions commonly have embedded address calculations, as the operands of floating point operators are often elements of one dimensional or two dimensional arrays. The CPU stack is deep enough to allow most integer calculations and address calculations to be performed within it. Similarly, the depth of the FPU stack allows most floating point expressions to be evaluated within it, employing the CPU stack to form addresses for the operands.

No hardware is used to deal with stack overflow. A compiler can easily examine expressions and introduce temporary variables in memory to avoid stack overflow. The number of such temporary variables can be minimised by careful choice of the evaluation order; an algorithm to perform this optimisation is given in the Prentice Hall publication *Transputer Instruction Set - A Compiler Writers' Guide*. The algorithm is used to optimise the use of the integer stack of the transputer CPU.

# 3.7.2 Concurrent operation of FPU and CPU

In the floating point transputer the FPU operates concurrently with the CPU. This means that it is possible to perform an address calculation in the CPU whilst the FPU performs a floating point calculation. This can lead to significant performance improvements in real applications which access arrays heavily. This aspect of the floating point transputer's performance was carefully assessed, partly through examination of the 'Livermore Loops' (refer to *The Livermore Fortran Kernels: A Computer Test of the Numerical Performance Range*). These are a collection of small kernels designed to represent the types of calculation performed on supercomputers. They are of interest because they contain constructs which occur in real programs which are not represented in such programs as the Whetstone benchmark. In particular, they contain accesses to two and three-dimensional arrays, operations where the concurrency within the floating point transputer is used to good effect. In some cases the compiler is able to choose the order of performing address calculations so as to maximise overlapping; this involves a modification of the algorithm mentioned earlier.

As a simple example of overlapping consider the implementation of Livermore Loop 7. The occam program for loop 7 is as follows:

```
-- LIVERMORE LOOP 7

SEQ k = 0 FOR n

x[k] := u[k] + ((( r*(z[k] + (r*y[k]))) +

(t*((u[k+3] + (r*(u[k+2] + (r*u[k+1]))))))) +

(t*((u[k+6] + (r*(u[k+5] + (r*u[k+4])))))))
```

The first stage in the computation of this is to load the value y[k]. This requires a sequence of four instructions. A further three instructions cause x to be loaded and the FPU multiply to be initiated.

Although the floating point multiplication takes several cycles to complete, the CPU is able to continue executing instructions whilst the FPU performs the multiplication. Thus the CPU can execute the next segment of code which computes the address of z[k] whilst the FPU perfroms the multiplication.

Finally, the value  $\mathbf{z}[\mathbf{k}]$  is pushed onto the floating point stack and added to the previously computed subexpression  $\mathbf{z} \star \mathbf{y}[\mathbf{k}]$ . It is not until value  $\mathbf{z}[\mathbf{k}]$  is loaded that the CPU needs to synchronise with the FPU.

The computation of the remainder of the expression proceeds in the same way, and the FPU never has to wait for the CPU to perform an address calculation.

# 3.8 Floating point unit design

In designing a concurrent systems component such as a transputer, it is important to maximise the performance obtained from a given area of silicon; many components can be used together to deliver more performance. This contrasts with the design of a conventional co-processor where the aim is to maximise the performance of a single processor by the use of a large area of silicon. As a result, in designing the floating point transputer, the performance benefits of silicon hungry devices such as barrel shifters and flash multipliers were carefully examined.

A flash multiplier is too large to fit on chip together with the processor, and would therefore necessitate the use of a separate co-processor chip. The introduction of a co-processor interface to a separate chip slows down the rate at which operands can be transferred to and from the floating point unit. Higher performance can, therefore, be obtained from a slow multiplier on the same chip as the processor than from a fast one on a separate chip. This leads to an important conclusion: a separate co-processor chip is not appropriate for scalar floating point arithmetic. A separate co-processor would be effective where a large amount of work can be handed to the co-processor by transferring a small amount of information; for example a vector co-processor would require only the addresses of its vector operands to be transferred via the co-processor interface.

It turns out that a flash multiplier also operates much more quickly than is necessary. Only a pipelined vector processor can deliver operands at a rate consistent with the use of such devices. In fact, any useful floating point calculation involves more operand accesses than operations. As an example consider the assignment y[i] := y[i] + (t \* x[i]) which constitutes the core of the LINPACK floating point benchmark. To perform this it is necessary to load three operands, perform two operations and to store a result. If we assume that it takes twice as long to perform a floating point operation as to load or store a floating point number then the execution time of this example would be evenly split between operand access time and operation time. This means that there would be at most a factor of two available in performance improvement from the use of an infinitely fast floating point unit!

Unlike a flash multiplier, a fast normalising shifter is important for fast floating point operation. When implementing IEEE arithmetic it may be necessary to perform a long shift on every floating point operation and unless a fast shifter is incorporated into the floating point unit the maximum operation time can become very long. Fortunately, unlike a flash multiplier, it is possible to design a fast shifter in a reasonable area of silicon. The shifter used is designed to perform a shift in a single cycle and to normalise in two cycles.

Consequently, the floating point unit contains a fast normalising shifter but not a flash multiplier. However there is a certain amount of logic devoted to multiplication and division. Multiplication is performed three-bits per cycle, and division is performed two-bits per cycle. Figure 3.15 illustrates the physical layout of the floating point unit.

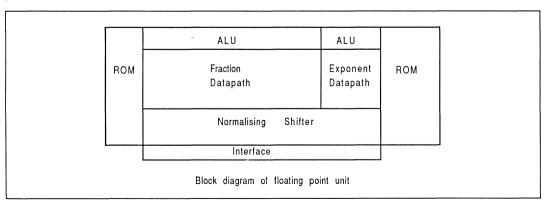


Figure 3.15 Floating point unit block diagram

The datapaths contain registers and shift paths. The fraction datapath is 59 bits wide, and the exponent data path is 13 bits wide. The normalising shifter interfaces to both the fraction data path and the exponent

datapath. This is because the data to be shifted will come from the fraction datapath whilst the magnitude of the shift is associated with the exponent datapath. One further interesting aspect of the design is the microcode ROM. Although the diagram shows two ROMs, they are both part of the same logical ROM. This has been split in two so that control signals do not need to be bussed through the datapaths.

# 3.9 Graphics capability

The fast block move instructions of the transputers make them suitable for use in graphics applications using byte-per-pixel colour displays. The block move on the transputer is designed to saturate the memory bandwidth, moving any number of bytes from any byte boundary in memory to any other byte boundary using the smallest possible number of word read and write operations.

Some transputers extend this capability by incorporation of a two-dimensional version of the block move (Move2d) which can move windows around a screen at full memory bandwidth, and conditional versions of the same block move which can be used to place templates and text into windows. One of these operations (Draw2d) copies bytes from source to destination, writing only non-zero bytes to the destination. A new object of any shape can therefore be drawn on top of the current image. A further operation (Clip2d) copies only zero bytes in the source. All of these instructions achieve the speed of the simple move instruction, enabling a 1 million pixel screen to be drawn many times per second. Unlike the conventional 'bit-bit' instruction, it is never necessary to read the destination data.

# 3.9.1 Example - drawing coloured text

Drawing proportional spaced text provides a simple example of the use of the two-dimensional move instructions. The font is stored in a two dimensional array **Font**; the height of **Font** is the fixed character height, and the start of each character is defined by an array **start**. The textures of the character and its background are selected from an array of textures; the textures providing a range of colours or even stripes and tartans!

An OCCAM procedure to perform such drawing is given below and the effect of each stage in the drawing process is illustrated by the diagrams on the final page of this document. First, (1) the texture for the character is selected and copied to a temporary area and (2) the character in the font is used to clip this texture to the appropriate shape. Then (3) the background texture is selected and copied to the screen, and (4) the new character drawn on top of it.

```
-- Draw character ch in texture F on background texture B
PROC DrawChar (VAL INT Ch, F, B)
  SEQ
      (x + width[ch]) > screenwidth
        SEQ
          x := 0
          y := y + height
      (x + width[ch]) <= screenwidth
        SKIP
    [height] [maxwidth] BYTE Temp :
    SEQ
      Move2d(Texture[F],0,0, Temp,0,0, width[ch],height)
      Clip2d(Font[ch], start[ch], 0, Temp, 0, 0, width[ch], height)
      Move2d(Texture[B],0,0, Screen,x,y, width[ch],height))
      Draw2d(Temp, 0, 0, Screen, x, y, width[ch], height)
      x := x + width[ch]
```

This procedure will fill a 1 million pixel screen with proportionally spaced characters in about 1/6 second. Obviously, a simpler and faster version could be used if the character colour or background colour was restricted. The operation of this procedure is illustrated in figure 3.16.

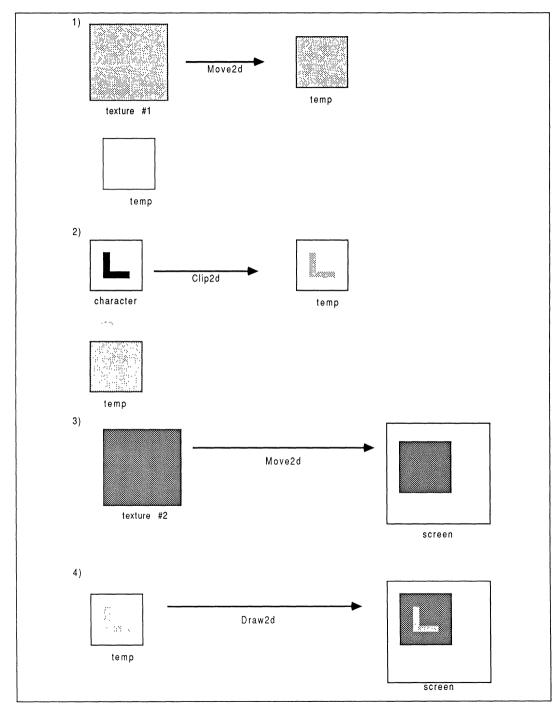
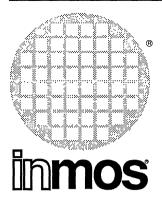


Figure 3.16 Use of enhanced graphics instructions

# 4 Conclusion

The INMOS transputer family is a range of system components which can be used to construct high performance concurrent systems. As all members of the family incorporate INMOS communications links, a system may be constructed from different members of the family. All transputers provide hardware support for concurrency and offer exceptional performance on process scheduling, inter-process communication and inter-transputer communication.

The design of the transputers takes careful note of silicon economics. The central processor used in the transputer offers a performance comparable with that of other VLSI processors several times larger. The small size of the processor allows a memory and communications system to be integrated on to the same VLSI device. This level of integration allows very fast access to memory and very fast inter-transputer communication. Similarly, the transputer floating point unit is integrated into the same device as the central processor, eliminating the delays inherent in communicating data between devices.



# IMS T805 transputer

# Preliminary Data

# **FEATURES**

32 bit architecture

33 ns internal cycle time

30 MIPS (peak) instruction rate

4.3 Mflops (peak) instruction rate

Pin compatible with IMS T800, IMS T425 and IMS T414 Debugging support

64 bit on-chip floating point unit which conforms to IEEE 754

4 Kbytes on-chip static RAM

120 Mbytes/sec sustained data rate to internal memory

4 Gbytes directly addressable external memory

40 Mbytes/sec sustained data rate to external memory 630 ns response to interrupts

Four INMOS serial links 5/10/20 Mbits/sec

Bi-directional data rate of 2.4 Mbytes/sec per link High performance graphics support with block move

instructions

Boot from ROM or communication links

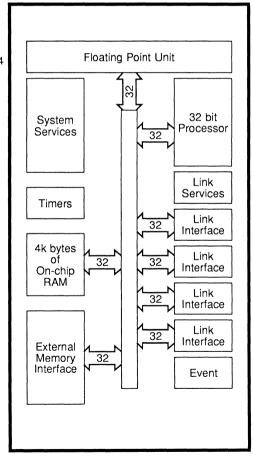
Single 5 MHz clock input

Single +5V ±5% power supply

MIL-STD-883C processing will be available

# **APPLICATIONS**

Scientific and mathematical applications
High speed multi processor systems
High performance graphics processing
Supercomputers
Workstations and workstation clusters
Digital signal processing
Accelerator processors
Distributed databases
System simulation
Telecommunications
Robotics
Fault tolerant systems
Image processing
Pattern recognition
Artificial intelligence



42 1440 00 May 1989

# 1 Introduction

The IMS T805 transputer is a 32 bit CMOS microcomputer with a 64 bit floating point unit and graphics support. It has 4 Kbytes on-chip RAM for high speed processing, a configurable memory interface and four standard INMOS communication links. The instruction set achieves efficient implementation of high level languages and provides direct support for the OCCAM model of concurrency when using either a single transputer or a network. Procedure calls, process switching and typical interrupt latency are sub-microsecond.

For convenience of description, the IMS T805 operation is split into the basic blocks shown in figure 1.1.

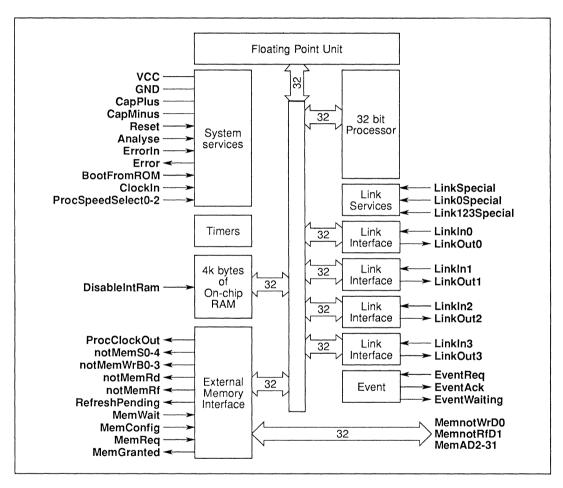


Figure 1.1 IMS T805 block diagram

The processor speed of a device can be pin-selected in stages from 17.5 MHz up to the maximum allowed for the part. A device running at 30 MHz achieves an instruction throughput of 30 MIPS peak and 15 MIPS sustained. The extended temperature version of the device complies with MIL-STD-883C.

The IMS T805 provides high performance arithmetic and floating point operations. The 64 bit floating point unit provides single and double length operation to the ANSI-IEEE 754-1985 standard for floating point arithmetic. It is able to perform floating point operations concurrently with the processor, sustaining a rate of 2.2 Mflops at a processor speed of 20 MHz and 3.3 Mflops at 30 MHz.

Introduction 49

High performance graphics support is provided by microcoded block move instructions which operate at the speed of memory. The two-dimensional block move instructions provide for contiguous block moves as well as block copying of either non-zero bytes of data only or zero bytes only. Block move instructions can be used to provide graphics operations such as text manipulation, windowing, panning, scrolling and screen updating.

Cyclic redundancy checking (CRC) instructions are available for use on arbitrary length serial data streams, to provide error detection where data integrity is critical. Another feature of the IMS T805, useful for pattern recognition, is the facility to count bits set in a word.

The IMS T805 can directly access a linear address space of 4 Gbytes. The 32 bit wide memory interface uses multiplexed data and address lines and provides a data rate of up to 4 bytes every 100 nanoseconds (40 Mbytes/sec) for a 30 MHz device. A configurable memory controller provides all timing, control and DRAM refresh signals for a wide variety of mixed memory systems.

System Services include processor reset and bootstrap control, together with facilities for error analysis. Error signals may be daisy-chained in multi-transputer systems.

The standard INMOS communication links allow networks of transputer family products to be constructed by direct point to point connections with no external logic. The IMS T805 links support the standard operating speed of 10 Mbits/sec, but also operate at 5 or 20 Mbits/sec. Each link can transfer data bi-directionally at up to 2.35 Mbytes/sec.

The IMS T805 is pin compatible with the IMS T800, as the extra inputs used are all held to ground on the IMS T800. The IMS T805-20 can thus be plugged directly into a circuit designed for a 20 MHz version of the IMS T800.

The transputer is designed to implement the OCCAM language, detailed in the OCCAM Reference Manual, but also efficiently supports other languages such as C, Pascal and Fortran. Access to the transputer at machine level is seldom required, but if necessary refer to the *Transputer Instruction Set - A Compiler Writers' Guide*. The instruction set of the IMS T805 is the same as that of the IMS T800.

This data sheet supplies hardware implementation and characterisation details for the IMS T805. It is intended to be read in conjunction with the Transputer Architecture chapter, which details the architecture of the transputer and gives an overview of OCCAM.

The IMS T805 instruction set contains a number of instructions to facilitate the implementation of breakpoints. For further information concerning breakpointing, refer to *Support for debugging/breakpointing in transputers* (technical note 61).

Figure 1.2 shows the internal datapaths for the IMS T805.

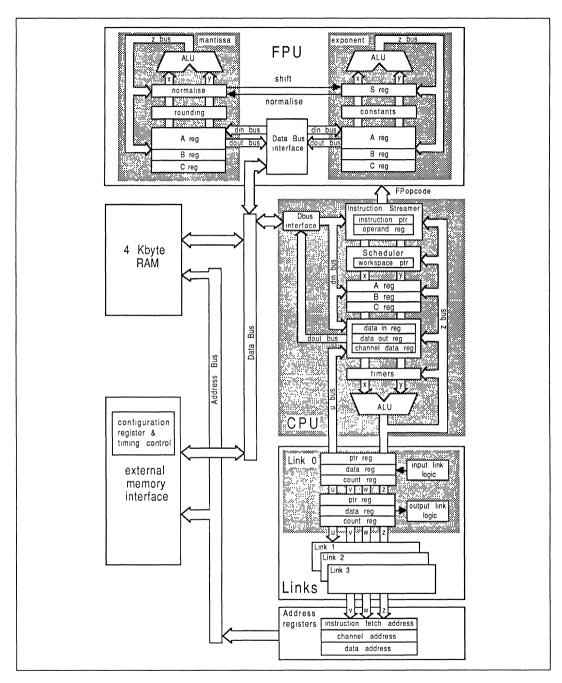


Figure 1.2 IMS T805 internal datapaths

# 2 Pin designations

Table 2.1 IMS T805 system services

Pin	In/Out	Function	
VCC, GND		Power supply and return	
CapPlus, CapMinus		External capacitor for internal clock power supply	
Clockin	in	Input clock	
ProcSpeedSelect0-2	in	Processor speed selectors	
Reset	in	System reset	
Error	out	Error indicator	
Errorin	in	Error daisychain input	
Analyse	in	Error analysis	
BootFromRom	in	Boot from external ROM or from link	
DisableIntRAM	in	Disable internal RAM	

Table 2.2 IMS T805 external memory interface

Pin	In/Out	Function	
ProcClockOut	out	Processor clock	
MemnotWrD0	in/out	Multiplexed data bit 0 and write cycle warning	
MemnotRfD1	in/out	Multiplexed data bit 1 and refresh warning	
MemAD2-31	in/out	Multiplexed data and address bus	
notMemRd	out	Read strobe	
notMemWrB0-3	out	Four byte-addressing write strobes	
notMemS0-4	out	Five general purpose strobes	
notMemRf	out	Dynamic memory refresh indicator	
RefreshPending	out	Dynamic refresh is pending	
MemWait	in	Memory cycle extender	
MemReq	in	Direct memory access request	
MemGranted	out	Direct memory access granted	
MemConfig	in	Memory configuration data input	

Table 2.3 IMS T805 event

Pin	In/Out	Function
EventReq	in	Event request
EventAck out		Event request acknowledge
EventWaiting	out	Event input requested by software

Table 2.4 IMS T805 link

Pin In/Out		Function	
Linkln0-3 in		Four serial data input channels	
LinkOut0-3 out Four serial data out		Four serial data output channels	
LinkSpecial in Select non-standard speed as 5 or 20 Mbits/se		Select non-standard speed as 5 or 20 Mbits/sec	
Link0Special in Select special speed for Link 0		Select special speed for Link 0	
Link123Special	in	Select special speed for Links 1,2,3	

Signal names are prefixed by **not** if they are active low, otherwise they are active high. Pinout details for various packages are given on page 120.

# 3 Processor

The 32 bit processor contains instruction processing logic, instruction and work pointers, and an operand register. It directly accesses the high speed 4 Kbyte on-chip memory, which can store data or program. Where larger amounts of memory or programs in ROM are required, the processor has access to 4 Gbytes of memory via the External Memory Interface (EMI).

# 3.1 Registers

The design of the transputer processor exploits the availability of fast on-chip memory by having only a small number of registers; six registers are used in the execution of a sequential process. The small number of registers, together with the simplicity of the instruction set, enables the processor to have relatively simple (and fast) data-paths and control logic. The six registers are:

The workspace pointer which points to an area of store where local variables are kept.

The instruction pointer which points to the next instruction to be executed.

The operand register which is used in the formation of instruction operands.

The A, B and C registers which form an evaluation stack.

A, B and C are sources and destinations for most arithmetic and logical operations. Loading a value into the stack pushes B into C, and A into B, before loading A. Storing a value from A, pops B into A and C into B.

Expressions are evaluated on the evaluation stack, and instructions refer to the stack implicitly. For example, the *add* instruction adds the top two values in the stack and places the result on the top of the stack. The use of a stack removes the need for instructions to respecify the location of their operands. Statistics gathered from a large number of programs show that three registers provide an effective balance between code compactness and implementation complexity.

No hardware mechanism is provided to detect that more than three values have been loaded onto the stack. It is easy for the compiler to ensure that this never happens.

Any location in memory can be accessed relative to the workpointer register, enabling the workspace to be of any size.

Further register details are given in Transputer Instruction Set - A Compiler Writers' Guide.

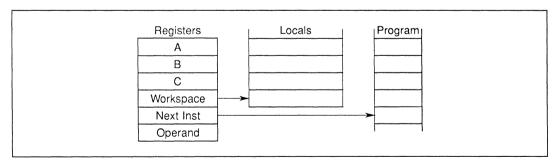


Figure 3.1 Registers

3 Processor 53

### 3.2 Instructions

The instruction set has been designed for simple and efficient compilation of high-level languages. All instructions have the same format, designed to give a compact representation of the operations occurring most frequently in programs.

Each instruction consists of a single byte divided into two 4-bit parts. The four most significant bits of the byte are a function code and the four least significant bits are a data value.

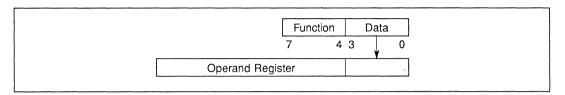


Figure 3.2 Instruction format

### 3.2.1 Direct functions

The representation provides for sixteen functions, each with a data value ranging from 0 to 15. Ten of these, shown in table 3.1, are used to encode the most important functions.

load constant	add constant	
load local	store local	load local pointer
load non-local	store non-local	
iump	conditional iump	call

Table 3.1 Direct functions

The most common operations in a program are the loading of small literal values and the loading and storing of one of a small number of variables. The *load constant* instruction enables values between 0 and 15 to be loaded with a single byte instruction. The *load local* and *store local* instructions access locations in memory relative to the workspace pointer. The first 16 locations can be accessed using a single byte instruction.

The *load non-local* and *store non-local* instructions behave similarly, except that they access locations in memory relative to the *A* register. Compact sequences of these instructions allow efficient access to data structures, and provide for simple implementations of the static links or displays used in the implementation of high level programming languages such as OCCAM, C, Fortran, Pascal or ADA.

#### 3.2.2 Prefix functions

Two more function codes allow the operand of any instruction to be extended in length; *prefix* and *negative prefix*.

All instructions are executed by loading the four data bits into the least significant four bits of the operand register, which is then used as the instruction's operand. All instructions except the prefix instructions end by clearing the operand register, ready for the next instruction.

The *prefix* instruction loads its four data bits into the operand register and then shifts the operand register up four places. The *negative prefix* instruction is similar, except that it complements the operand register before shifting it up. Consequently operands can be extended to any length up to the length of the operand register by a sequence of prefix instructions. In particular, operands in the range -256 to 255 can be represented using one prefix instruction.

The use of prefix instructions has certain beneficial consequences. Firstly, they are decoded and executed in the same way as every other instruction, which simplifies and speeds instruction decoding. Secondly, they simplify language compilation by providing a completely uniform way of allowing any instruction to take an operand of any size. Thirdly, they allow operands to be represented in a form independent of the processor wordlength.

#### 3.2.3 Indirect functions

The remaining function code, *operate*, causes its operand to be interpreted as an operation on the values held in the evaluation stack. This allows up to 16 such operations to be encoded in a single byte instruction. However, the prefix instructions can be used to extend the operand of an *operate* instruction just like any other. The instruction representation therefore provides for an indefinite number of operations.

Encoding of the indirect functions is chosen so that the most frequently occurring operations are represented without the use of a prefix instruction. These include arithmetic, logical and comparison operations such as add, exclusive or and greater than. Less frequently occurring operations have encodings which require a single prefix operation.

## 3.2.4 Expression evaluation

Evaluation of expressions sometimes requires use of temporary variables in the workspace, but the number of these can be minimised by careful choice of the evaluation order.

Program	Mnem	onic
x := 0	ldc	0
	stl	X
x := #24	pfix Idc	2
	ldc	4
	stl	X
x := y + z	Idl	У
	Idl	y z
	add	
	stl	X

Table 3.2 Expression evaluation

### 3.2.5 Efficiency of encoding

Measurements show that about 70% of executed instructions are encoded in a single byte; that is, without the use of prefix instructions. Many of these instructions, such as *load constant* and *add* require just one processor cycle.

The instruction representation gives a more compact representation of high level language programs than more conventional instruction sets. Since a program requires less store to represent it, less of the memory bandwidth is taken up with fetching instructions. Furthermore, as memory is word accessed the processor will receive four instructions for every fetch.

Short instructions also improve the effectiveness of instruction pre-fetch, which in turn improves processor performance. There is an extra word of pre-fetch buffer, so the processor rarely has to wait for an instruction fetch before proceeding. Since the buffer is short, there is little time penalty when a jump instruction causes the buffer contents to be discarded.

### 3.3 Processes and concurrency

A process starts, performs a number of actions, and then either stops without completing or terminates complete. Typically, a process is a sequence of instructions. A transputer can run several processes in parallel (concurrently). Processes may be assigned either high or low priority, and there may be any number of each (page 56).

The processor has a microcoded scheduler which enables any number of concurrent processes to be executed together, sharing the processor time. This removes the need for a software kernel.

At any time, a concurrent process may be

Active - Being executed.

- On a list waiting to be executed.

Inactive - Ready to input.

Ready to output.

- Waiting until a specified time.

The scheduler operates in such a way that inactive processes do not consume any processor time. It allocates a portion of the processor's time to each process in turn. Active processes waiting to be executed are held in two linked lists of process workspaces, one of high priority processes and one of low priority processes (page 56). Each list is implemented using two registers, one of which points to the first process in the list, the other to the last. In the Linked Process List figure 3.3, process S is executing and P, Q and R are active, awaiting execution. Only the low priority process queue registers are shown; the high priority process ones perform in a similar manner.

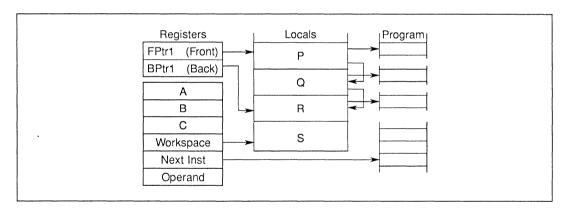


Figure 3.3 Linked process list

Table 3.3 Priority queue control registers

Function	High Priority	Low Priority
Pointer to front of active process list	Fptr0	Fptr1
Pointer to back of active process list	Bptr0	Bptr1

Each process runs until it has completed its action, but is descheduled whilst waiting for communication from another process or transputer, or for a time delay to complete. In order for several processes to operate in parallel, a low priority process is only permitted to run for a maximum of two time slices before it is forcibly descheduled at the next descheduling point (page 60). The time slice period is 5120 cycles of the external 5 MHz clock, giving ticks approximately 1 ms apart.

A process can only be descheduled on certain instructions, known as descheduling points (page 60). As a result, an expression evaluation can be guaranteed to execute without the process being timesliced part way through.

Whenever a process is unable to proceed, its instruction pointer is saved in the process workspace and the next process taken from the list. Process scheduling pointers are updated by instructions which cause scheduling operations, and should not be altered directly. Actual process switch times are less than 1  $\mu$ s, as little state needs to be saved and it is not necessary to save the evaluation stack on rescheduling.

The processor provides a number of special operations to support the process model, including *start process* and *end process*. When a main process executes a parallel construct, *start process* instructions are used to create the necessary additional concurrent processes. A *start process* instruction creates a new process by adding a new workspace to the end of the scheduling list, enabling the new concurrent process to be executed together with the ones already being executed. When a process is made active it is always added to the end of the list, and thus cannot pre-empt processes already on the same list.

The correct termination of a parallel construct is assured by use of the *end process* instruction. This uses a workspace location as a counter of the parallel construct components which have still to terminate. The counter is initialised to the number of components before the processes are *started*. Each component ends with an *end process* instruction which decrements and tests the counter. For all but the last component, the counter is non zero and the component is descheduled. For the last component, the counter is zero and the main process continues.

## 3.4 Priority

The IMS T805 supports two levels of priority. Priority 1 (low priority) processes are executed whenever there are no active priority 0 (high priority) processes.

High priority processes are expected to execute for a short time. If one or more high priority processes are able to proceed, then one is selected and runs until it has to wait for a communication, a timer input, or until it completes processing.

If no process at high priority is able to proceed, but one or more processes at low priority are able to proceed, then one is selected.

Low priority processes are periodically timesliced to provide an even distribution of processor time between computationally intensive tasks.

If there are  $\mathbf{n}$  low priority processes, then the maximum latency from the time at which a low priority process becomes active to the time when it starts processing is  $2\mathbf{n}$ -2 timeslice periods. It is then able to execute for between one and two timeslice periods, less any time taken by high priority processes. This assumes that no process monopolises the transputer's time; i.e. it has a distribution of descheduling points (page 60).

Each timeslice period lasts for 5120 cycles of the external 5 MHz input clock (approximately 1 ms at the standard frequency of 5 MHz).

If a high priority process is waiting for an external channel to become ready, and if no other high priority process is active, then the interrupt latency (from when the channel becomes ready to when the process starts executing) is typically 19 processor cycles, a maximum of 78 cycles (assuming use of on-chip RAM). If the floating point unit is not being used at the time then the maximum interrupt latency is only 58 cycles. To ensure this latency, certain instructions are interruptable.

## 3.5 Communications

Communication between processes is achieved by means of channels. Process communication is point-to-point, synchronised and unbuffered. As a result, a channel needs no process queue, no message queue and no message buffer.

A channel between two processes executing on the same transputer is implemented by a single word in memory; a channel between processes executing on different transputers is implemented by point-to-point links. The processor provides a number of operations to support message passing, the most important being input message and output message.

The *input message* and *output message* instructions use the address of the channel to determine whether the channel is internal or external. Thus the same instruction sequence can be used for both, allowing a process to be written and compiled without knowledge of where its channels are connected.

The process which first becomes ready must wait until the second one is also ready. A process performs an input or output by loading the evaluation stack with a pointer to a message, the address of a channel, and a count of the number of bytes to be transferred, and then executing an *input message* or *output message* instruction. Data is transferred if the other process is ready. If the channel is not ready or is an external one the process will deschedule.

#### 3.6 Block move

The block move on the transputer moves any number of bytes from any byte boundary in memory, to any other byte boundary, using the smallest possible number of word read, and word or part-word writes.

A block move instruction can be interrupted by a high priority process. On interrupt, block move is completed to a word boundary, independent of start position. When restarting after interrupt, the last word written is written again. This appears as an unnecessary read and write in the simplest case of word aligned block moves, and may cause problems with FIFOs. This problem can be overcome by incrementing the saved destination (BregIntSaveLoc) and source pointer (CregIntSaveLoc) values by BytesPerWord during the high priority process.

#### 3.7 Timers

The transputer has two 32 bit timer clocks which 'tick' periodically. The timers provide accurate process timing, allowing processes to deschedule themselves until a specific time.

One timer is accessible only to high priority processes and is incremented every microsecond, cycling completely in approximately 4295 seconds. The other is accessible only to low priority processes and is incremented every 64 microseconds, giving exactly 15625 ticks in one second. It has a full period of approximately 76 hours.

Table 3.4 Timer registers

Clock0	Current value of high priority (level 0) process clock
Clock1	Current value of low priority (level 1) process clock
TNextReg0	Indicates time of earliest event on high priority (level 0) timer queue
TNextReg1	Indicates time of earliest event on low priority (level 1) timer queue

The current value of the processor clock can be read by executing a *load timer* instruction. A process can arrange to perform a *timer input*, in which case it will become ready to execute after a specified time has been reached. The *timer input* instruction requires a time to be specified. If this time is in the 'past' then the instruction has no effect. If the time is in the 'future' then the process is descheduled. When the specified time is reached the process is scheduled again.

Figure 3.4 shows two processes waiting on the timer queue, one waiting for time 21, the other for time 31.

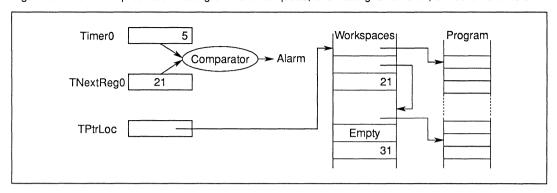


Figure 3.4 Timer registers

# 4 Instruction set summary

The Function Codes table 4.8 gives the basic function code set (page 53). Where the operand is less than 16, a single byte encodes the complete instruction. If the operand is greater than 15, one prefix instruction (*pfix*) is required for each additional four bits of the operand. If the operand is negative the first prefix instruction will be *nfix*.

Mnem	nonic		Function code	Memory code
ldc	#3		#4	#43
ldc	#35			
is coded as				
pfix	#3		#2	#23
ldc	#5		#4	#45
ldc	#987			
is coded as				
pfix	#9		#2	#29
pfix	#8		#2	#28
İdc	#7		#4	#47
ldc	-31	(ldc	#FFFFFFE1)	
is coded as		•	,	
nfix	#1		#6	#61
ldc	#1		#4	#41

Table 4.1 prefix cooing

Tables 4.9 to 4.28 give details of the operation codes. Where an operation code is less than 16 (e.g. add: operation code 05), the operation can be stored as a single byte comprising the operate function code F and the operand (5 in the example). Where an operation code is greater than 15 (e.g. ladd: operation code 16), the prefix function code 2 is used to extend the instruction.

Mnem	nonic	Function code	Memory code
add is coded as	(op. code #5)		#F5
opr	add	#F	#F5
ladd is coded as	(op. code #16)		#21F6
pfix opr	#1 #6	#2 #F	#21 #F6

Table 4.2 operate coding

The load device identity (*Iddevid*) instruction (table 4.20) pushes the device type identity into the A register. Each product is allocated a unique group of numbers for use with the *Iddevid* instruction. The product identity numbers for the IMS T805 are 10 to 19 inclusive.

In the Floating Point Operation Codes tables 4.22 to 4.28, a selector sequence code (page 69) is indicated in the Memory Code column by **s**. The code given in the Operation Code column is the indirection code, the operand for the *ldc* instruction.

The FPU and processor operate concurrently, so the actual throughput of floating point instructions is better than that implied by simply adding up the instruction times. For full details see *Transputer Instruction Set - A Compiler Writers' Guide*.

The Processor Cycles column refers to the number of periods **TPCLPCL** taken by an instruction executing in internal memory. The number of cycles is given for the basic operation only; where the memory code for an instruction is two bytes, the time for the *prefix* function (one cycle) should be added. For a 20 MHz transputer one cycle is 50 ns. Some instruction times vary. Where a letter is included in the cycles column it is interpreted from table 4.3.

Table 4.3 Instruction set interpretation

Ident	Interpretation
b	Bit number of the highest bit set in register A. Bit 0 is the least significant bit.
m	Bit number of the highest bit set in the absolute value of register A. Bit 0 is the least significant bit.
n	Number of places shifted.
w	Number of words in the message. Part words are counted as full words. If the message is not word aligned the number of words is increased to include the part words at either end of the message.
р	Number of words per row.
r	Number of rows.

The **DE** column of the tables indicates the descheduling/error features of an instruction as described in table 4.4.

Table 4.4 Instruction features

ldent	Feature	See page:
D	The instruction is a descheduling point	60
E	The instruction will affect the Error flag	61, 76
F	The instruction will affect the FP_Error flag	69, 61

# 4.1 Descheduling points

The instructions in table 4.5 are the only ones at which a process may be descheduled (page 55). They are also the ones at which the processor will halt if the **Analyse** pin is asserted (page 75).

Table 4.5 Descheduling point instructions

input message	output message	output byte	output word
timer alt wait	timer input	stop on error	alt wait
jump	loop end	end process	stop process

### 4.2 Error instructions

The instructions in table 4.6 are the only ones which can affect the *Error* flag (page 76) directly. Note, however, that the floating point unit error flag *FP\_Error* is set by certain floating point instructions (page 61), and that *Error* can be set from this flag by *fpcheckerror*.

Table 4.6 Error setting instructions

add	add constant	subtract	remainder
multiply	fractional multiply	divide	
long add	long subtract	long divide	
set error	testerr	fpcheckerror	check count from 1
check word	check subscript from 0	check single	

## 4.3 Debugging support

Table 4.21 contains a number of instructions to facilitate the implementation of breakpoints. These instructions overload the operation of *j0*. Normally *j0* is a no-op which might cause descheduling. *Setj0break* enables the breakpointing facilities and causes *j0* to act as a breakpointing instruction. When breakpointing is enabled, *j0* swaps the current **lptr** and **Wptr** with an **lptr** and **Wptr** stored above MemStart. The breakpoint instruction does not cause descheduling, and preserves the state of the registers. It is possible to single step the processor at machine level using these instructions. Refer to *Support for debugging/breakpointing in transputers* (technical note 61) for more detailed information regarding debugger support.

## 4.4 Floating point errors

The instructions in table 4.7 are the only ones which can affect the floating point error flag *FP\_Error* (page 69). *Error* is set from this flag by *fpcheckerror* if *FP\_Error* is set.

Table 4.7 Floating point error setting instructions

fpadd fpldnladdsn fpremfirst fpuseterror	fpsub fpldnladddb fpusqrtfirst fpuclearerror	fpmul fpldnlmulsn fpgt fptesterror	fpdiv fpldnlmuldb fpeq	
fpuexpincby32 fpur32tor64 fprtoi32	fpuexpdecby32 fpur64tor32 fpuabs	fpumulby2 fpucki32 fpint	fpudivby2 fpucki64	

Function Code	Memory Code	Mnemonic	Processor Cycles	Name	D E
0	0X	j	3	jump	D
1	1X	ldlp	1	load local pointer	
2	2X	pfix	1	prefix	
3	3X	Idni	2	load non-local	
4	4X	ldc	1	load constant	
5	5X	Idnip	1	load non-local pointer	
6	6X	nfix	1	negative prefix	
7	7X	ldl	2	load local	
8	8X	adc	1	add constant	E
9	9X	call	7	call	
Α	AX	cj	2	conditional jump (not taken)	
			4	conditional jump (taken)	
В	BX	ajw	1	adjust workspace	
С	CX	eqc	2	equals constant	
D	DX	sti	1	store local	
E	EX	stni	2	store non-local	
F	FX	opr	-	operate	

Table 4.8 IMS T805 function codes

Table 4.9 IMS T805 arithmetic/logical operation codes

Operation Code	Memory Code	Mnemonic	Processor Cycles	Name	D E
46	24F6	and	1	and	
4B	24FB	or	1	or	
33	23F3	xor	1	exclusive or	
32	23F2	not	1	bitwise not	
41	24F1	shl	n+2	shift left	
40	24F0	shr	n+2	shift right	
05	F5	add	1	add	Е
0C	FC	sub	1	subtract	E
53	25F3	mul	38	multiply	E
72	27F2	fmul	35	fractional multiply (no rounding)	E
			40	fractional multiply (rounding)	E
2C	22FC	div	39	divide	E
1F	21FF	rem	37	remainder	E
09	F9	gt	2	greater than	
04	F4	diff	1	difference	
52	25F2	sum	1	sum	
08	F8	prod	b+4	product for positive register A	
			<b>m</b> +5	product for negative register A	

Operation Code	Memory Code	Mnemonic	Processor Cycles	Name	D E
16	21F6	ladd	2	long add	E
38	23F8	Isub	2	long subtract	E
37	23F7	Isum	3	നg sum	
4F	24FF	Idiff	3	long diff	
31	23F1	Imul	33	long multiply	
1 A	21FA	ldiv	35	long divide	E
36	23F6	Ishl	n+3	long shift left (n<32)	
			n-28	long shift left(n≥32)	
35	23F5	Ishr	n+3	long shift right (n<32)	- 1
	•		n-28	long shift right (n≥32)	
19	21F9	norm	n+5	normalise (n<32)	
			n-26	normalise ( <b>n</b> ≥32)	
			3	normalise ( <b>n</b> =64)	

Table 4.10 IMS T805 long arithmetic operation codes

Table 4.11 IMS T805 general operation codes

Operation Code	Memory Code	Mnemonic	Processor Cycles	Name	D E
00	F0	rev	1	reverse	
3A	23FA	xword	4	extend to word	
56	25F6	cword	5	check word	E
1D	21FD	xdble	2	extend to double	
4C	24FC	csngl	3	check single	E
42	24F2	mint	1	minimum integer	
5A	25FA	dup	1	duplicate top of stack	1
79	27F9	pop	1	pop processor stack	

Table 4.12 IMS T805 2D block move operation codes

Operation Code	Memory Code	Mnemonic	Processor Cycles	Name	D E
5B	25FB	move2dinit	8	initialise data for 2D block move	
5C	25FC	move2dall	(2p+23)*r	2D block copy	
5D	25FD	move2dnonzero	(2 <b>p</b> +23)∗ <b>r</b>	2D block copy non-zero bytes	
5E	25FE	move2dzero	(2 <b>p</b> +23)* <b>r</b>	2D block copy zero bytes	

Table 4.13 IMS T805 CRC and bit operation codes

Operation Code	Memory Code	Mnemonic	Processor Cycles	Name	D E
74	27F4	crcword	35	calculate crc on word	
75	27F5	crcbyte	11	calculate crc on byte	
76	27F6	bitcnt	<b>b</b> +2	count bits set in word	
77	27F7	bitrevword	36	reverse bits in word	
78	27F8	bitrevnbits	n+4	reverse bottom n bits in word	

Operation Code	Memory Code	Mnemonic	Processor Cycles	Name	D E
02	F2	bsub	1	byte subscript	
0A	FA	wsub	2	word subscript	
81	28F1	wsubdb	3	form double word subscript	
34	23F4	bcnt	2	byte count	
3F	23FF	went	5	word count	1
01	F1	lb	5	load byte	İ
3B	23FB	sb	4	store byte	
4 <b>A</b>	24FA	move	2 <b>w</b> +8	move message	

Table 4.14 IMS T805 indexing/array operation codes

Table 4.15 IMS T805 timer handling operation codes

Operation Code	Memory Code	Mnemonic	Processor Cycles	Name	D E
22	22F2	ldtimer	2	load timer	
2B	22FB	tin	30	timer input (time future)	D
			4	timer input (time past)	D
4E	24FE	talt	4	timer alt start	
51	25F1	taltwt	15	timer alt wait (time past)	D
			48	timer alt wait (time future)	D
47	24F7	enbt	8	enable timer	
2E	22FE	dist	23	disable timer	

Table 4.16 IMS T805 input/output operation codes

Operation Code	Memory Code	Mnemonic	Processor Cycles	Name	D E
07	F7	in	2 <b>w</b> +19	input message	D
0B	FB	out	2 <b>w</b> +19	output message	D
0F	FF	outword	23	output word	D
0E	FE	outbyte	23	output byte	D
43	24F3	alt	2	alt start	
44	24F4	altwt	5	alt wait (channel ready)	D
			17	alt wait (channel not ready)	D
45	24F5	altend	4	alt end	
49	24F9	enbs	3	enable skip	
30	23F0	diss	4	disable skip	
12	21F2	resetch	3	reset channel	
48	24F8	enbc	7	enable channel (ready)	
			5	enable channel (not ready)	
2F	22FF	disc	8	disable channel	

Operation Code	Memory Code	Mnemonic	Processor Cycles	Name	D E
20	22F0	ret	5	return	
1B	21FB	ldpi	2	load pointer to instruction	
3C	23FC	gajw	2	general adjust workspace	1
06	F6	gcall	4	general call	1
21	22F1	lend	10	loop end (loop)	D
			5	loop end (exit)	D

Table 4.17 IMS T805 control operation codes

Table 4.18 IMS T805 scheduling operation codes

Operation Code	Memory Code	Mnemonic	Processor Cycles	Name	D E
0D	FD	startp	12	start process	D
03	F3	endp	13	end process	D
39	23F9	runp	10	run process	
15	21F5	stopp	11	stop process	
1E	21FE	Idpri	1	load current priority	

Table 4.19 IMS T805 error handling operation codes

Operation Code	Memory Code	Mnemonic	Processor Cycles	Name	D E
13	21F3	csub0	2	check subscript from 0	E
4D	24FD	ccnt1	3	check count from 1	E
29	22F9	testerr	2	test error false and clear (no error)	
			3	test error false and clear (error)	1
10	21F0	seterr	1	set error	E
55	25F5	stoperr	2	stop on error (no error)	D
57	25F7	cirhaiterr	1	clear halt-on-error	] [
58	25F8	sethalterr	1	set halt-on-error	] ]
59	25F9	testhalterr	2	test halt-on-error	

Table 4.20 IMS T805 processor initialisation operation codes

Operation Code	Memory Code	Mnemonic	Processor Cycles	Name	D E
2A	22FA	testpranal	2	test processor analysing	
3E	23FE	saveh	4	save high priority queue registers	
3D	23FD	savel	4	save low priority queue registers	
18	21F8	sthf	1	store high priority front pointer	
50	25F0	sthb	1	store high priority back pointer	
1C	21FC	stlf	1	store low priority front pointer	1 1
17	21F7	stlb	1	store low priority back pointer	1 (
54	25F4	sttimer	1	store timer	
17C	2127FC	Iddevid	1	load device identity	
7E	27FE	Idmemstartval	1	load value of memstart address	

Operation Code	Memory Code	Mnemonic	Processor Cycles	Name	D E
0	00	jump 0	3	jump 0 (break not enabled)	D
			11	jump 0 (break enabled, high priority)	
			13	jump 0 (break enabled, low priority)	
B1	2BF1	break	9	break (high priority)	į
			11	break (low priority)	
B2	2BF2	clrj0break	1	clear jump 0 break enable flag	
B3	2BF3	setj0break	1	set jump 0 break enable flag	l
B4	2BF4	testj0break	2	test jump 0 break enable flag set	Ì
7A	27FA	timerdisableh	1	disable high priority timer interrupt	
7B	27FB	timerdisablel	1	disable low priority timer interrupt	
7C	27FC	timerenableh	6	enable high priority timer interrupt	
7D	27FD	timerenablel	6	enable low priority timer interrupt	

Table 4.21 IMS T805 debugger support codes

Table 4.22 IMS T805 floating point load/store operation codes

Operation Code	Memory Code	Mnemonic	Processor Cycles	Name	D E
8E	28FE	fpidnisn	2	fp load non-local single	
8A	28FA	fpldnldb	3	fp load non-local double	1 1
86	28F6	fpldnlsni	4	fp load non-local indexed single	
82	28F2	fpldnldbi	6	fp load non-local indexed double	
9F	29FF	fpldzerosn	2	load zero single	
A0	2AF0	fpldzerodb	2	load zero double	
AA	2AFA	fpldnladdsn	8/11	fp load non local & add single	F
<b>A</b> 6	2AF6	fpldnladddb	9/12	fp load non local & add double	F
AC	2AFC	fpldnlmulsn	13/20	fp load non local & multiply single	F
A8	2AF8	fpldnlmuldb	21/30	fp load non local & multiply double	F
88	28F8	fpstnisn	2	fp store non-local single	
84	28F4	fpstnldb	3	fp store non-local double	
9E	29FE	fpstnli32	4	store non-local int32	

Processor cycles are shown as Typical/Maximum cycles.

Table 4.23 IMS T805 floating point general operation codes

Operation Code	Memory Code	Mnemonic	Processor Cycles	Name	D E
AB	2AFB	fpentry	1	floating point unit entry	
A4	2AF4	fprev	1	fp reverse	
А3	2AF3	fpdup	1	fp duplicate	

Operation Code	Memory Code	Mnemonic	Processor Cycles	Name	D E
22	S	fpurn	1	set rounding mode to round nearest	
06	S	fpurz	1	set rounding mode to round zero	
04	S	fpurp	1	set rounding mode to round positive	
05	s	fpurm	1	set rounding mode to round minus	

Table 4.24 IMS T805 floating point rounding operation codes

Table 4.25 IMS T805 floating point error operation codes

Operation Code	Memory Code	Mnemonic	Processor Cycles	Name	D E
83	28F3	fpchkerror	1	check fp error	E
9C	29FC	fptesterror	2	test fp error false and clear	F
23	s	fpuseterror	1	set fp error	F
9C	s	fpuclearerror	1	clear fp error	F

Table 4.26 IMS T805 floating point comparison operation codes

Operation Code	Memory Code	Mnemonic	Processor Cycles	Name	D E
94	29F4	fpgt	4/6	fp greater than	F
95	29F5	fpeq	3/5	fp equality	F
92	29F2	fpordered	3/4	fp orderability	
91	29F1	fpnan	2/3	fp NaN	
93	29F3	fpnotfinite	2/2	fp not finite	
0E	s	fpuchki32	3/4	check in range of type int32	F
0F	s	fpuchki64	3/4	check in range of type int64	F

Processor cycles are shown as Typical/Maximum cycles.

Table 4.27 IMS T805 floating point conversion operation codes

Operation Code	Memory Code	Mnemonic	Processor Cycles	Name	D E
07	S	fpur32tor64	3/4	real32 to real64	F
08	s	fpur64tor32	6/9	real64 to real32	F
9D	29FD	fprtoi32	7/9	real to int32	F
96	29F6	fpi32tor32	8/10	int32 to real32	1
98	29F8	fpi32tor64	8/10	int32 to real64	
9A	29FA	fpb32tor64	8/8	bit32 to real64	
0D	s	fpunoround	2/2	real64 to real32, no round	
A1	2AF1	fpint	5/6	round to floating integer	F

Processor cycles are shown as Typical/Maximum cycles.

Memory		Processo	or cycles		D
Code	Mnemonic	Single	Double	Name	E
28F7	fpadd	6/9	6/9	fp add	F
28F9	fpsub	6/9	6/9	fp subtract	F
28FB	fpmul	11/18	18/27	fp multiply	F
28FC	fpdiv	16/28	31/43	fp divide	F
s	fpuabs	2/2	2/2	fp absolute	F
28FF	fpremfirst	36/46	36/46	fp remainder first step	F
29F0	fpremstep	32/36	32/36	fp remainder iteration	1
s	fpusqrtfirst	27/29	27/29	fp square root first step	F
s	fpusqrtstep	42/42	42/42	fp square root step	
s	fpusqrtlast	8/9	8/9	fp square root end	
s	fpuexpinc32	6/9	6/9	multiply by 2 <sup>32</sup>	F
s	fpuexpdec32	6/9	6/9	divide by 2 <sup>32</sup>	F
s	fpumulby2	6/9	6/9	multiply by 2.0	F
s	fpudivby2	6/9	6/9	divide by 2.0	F
	28F7 28F9 28FB 28FC \$ 28FF 29F0 \$ \$ \$ \$ \$ \$ \$ \$ \$ \$ \$ \$ \$ \$ \$ \$ \$ \$ \$	Code Mnemonic  28F7 fpadd 28F9 fpsub 28FB fpmul 28FC fpdiv s fpuabs 28FF fpremfirst 29F0 fpremstep s fpusqrtfirst s fpusqrtstep s fpusqrtlast s fpuexpinc32 s fpuemulby2	Code         Mnemonic         Single           28F7         fpadd         6/9           28F9         fpsub         6/9           28FB         fpmul         11/18           28FC         fpdiv         16/28           s         fpuabs         2/2           28FF         fpremfirst         36/46           29F0         fpremstep         32/36           s         fpusqrtfirst         27/29           s         fpusqrtstep         42/42           s         fpusqrtlast         8/9           s         fpuexpinc32         6/9           s         fpuexpdec32         6/9           s         fpumulby2         6/9	Code         Mnemonic         Single         Double           28F7         fpadd         6/9         6/9           28F9         fpsub         6/9         6/9           28FB         fpmul         11/18         18/27           28FC         fpdiv         16/28         31/43           s         fpuabs         2/2         2/2           28FF         fpremfirst         36/46         36/46           29F0         fpremstep         32/36         32/36           s         fpusqrtfirst         27/29         27/29           s         fpusqrtstep         42/42         42/42           s         fpusqrtlast         8/9         8/9           s         fpuexpinc32         6/9         6/9           s         fpumulby2         6/9         6/9	Code         Mnemonic         Single         Double         Name           28F7         fpadd         6/9         6/9         fp add           28F9         fpsub         6/9         6/9         fp subtract           28FB         fpmul         11/18         18/27         fp multiply           28FC         fpdiv         16/28         31/43         fp divide           s         fpuabs         2/2         2/2         fp absolute           28FF         fpremfirst         36/46         36/46         fp remainder first step           29F0         fpremstep         32/36         32/36         fp remainder iteration           s         fpusqrtfirst         27/29         27/29         fp square root first step           s         fpusqrtstep         42/42         42/42         fp square root step           s         fpusqrtlast         8/9         8/9         fp square root end           s         fpuexplec32         6/9         6/9         divide by 2 <sup>32</sup> s         fpumulby2         6/9         6/9         multiply by 2.0

Table 4.28 IMS T805 floating point arithmetic operation codes

Processor cycles are shown as Typical/Maximum cycles.

# 5 Floating point unit

The 64 bit FPU provides single and double length arithmetic to floating point standard ANSI-IEEE 754-1985. It is able to perform floating point arithmetic concurrently with the central processor unit (CPU), sustaining 3.3 Mflops on a 30 MHz device. All data communication between memory and the FPU occurs under control of the CPU.

The FPU consists of a microcoded computing engine with a three deep floating point evaluation stack for manipulation of floating point numbers. These stack registers are *FA*, *FB* and *FC*, each of which can hold either 32 bit or 64 bit data; an associated flag, set when a floating point value is loaded, indicates which. The stack behaves in a similar manner to the CPU stack (page 52).

As with the CPU stack, the FPU stack is not saved when rescheduling (page 55) occurs. The FPU can be used in both low and high priority processes. When a high priority process interrupts a low priority one the FPU state is saved inside the FPU. The CPU will service the interrupt immediately on completing its current operation. The high priority process will not start, however, before the FPU has completed its current operation.

Points in an instruction stream where data need to be transferred to or from the FPU are called *synchronisation points*. At a synchronisation point the first processing unit to become ready will wait until the other is ready. The data transfer will then occur and both processors will proceed concurrently again. In order to make full use of concurrency, floating point data source and destination addresses can be calculated by the CPU whilst the FPU is performing operations on a previous set of data. Device performance is thus optimised by minimising the CPU and FPU idle times.

The FPU has been designed to operate on both single length (32 bit) and double length (64 bit) floating point numbers, and returns results which fully conform to the ANSI-IEEE 754-1985 floating point arithmetic standard. Denormalised numbers are fully supported in the hardware. All rounding modes defined by the standard are implemented, with the default being round to nearest.

The basic addition, subtraction, multiplication and division operations are performed by single instructions. However, certain less frequently used floating point instructions are selected by a value in register *A* (when allocating registers, this should be taken into account). A *load constant* instruction *ldc* is used to load register *A*; the *floating point entry* instruction *fpentry* then uses this value to select the floating point operation. This pair of instructions is termed a *selector sequence*.

Names of operations which use *fpentry* begin with *fpu*. A typical usage, returning the absolute value of a floating point number, would be

Idc fpuabs; fpentry;

Since the indirection code for fpuabs is **0B**, it would be encoded as

Table 5.1 fpentry coding

Mnemonic		Function code	Memory code
ldc	fpuabs	#4	#4B
fpentry is coded as	(op. code #AB)		#2AFB
pfix opr	#A #B	#2 #F	#2A #FB

The *remainder* and *square root* instructions take considerably longer than other instructions to complete. In order to minimise the interrupt latency period of the transputer they are split up to form instruction sequences. As an example, the instruction sequence for a single length square root is

fpusqrtfirst; fpusqrtstep; fpusqrtlast;

The FPU has its own error flag *FP\_Error*. This reflects the state of evaluation within the FPU and is set in circumstances where invalid operations, division by zero or overflow exceptions to the ANSI-IEEE 754-1985 standard would be flagged (page 61). *FP\_Error* is also set if an input to a floating point operation is infinite or is not a number (NaN). The *FP\_Error* flag can be set, tested and cleared without affecting the main *Error* flag, but can also set *Error* when required (page 61). Depending on how a program is compiled, it is possible for both unchecked and fully checked floating point arithmetic to be performed.

Further details on the operation of the FPU can be found in *Transputer Instruction Set - A Compiler Writers' Guide.* 

	T805-20		T805-30		
Operation	Single length	Double length	Single length	Double length	
add	350 ns	350 ns	233 ns	233 ns	
subtract	350 ns	350 ns	233 ns	233 ns	
multiply	550 ns	1000 ns	367 ns	667 ns	
divide	850 ns	1600 ns	567 ns	1067 ns	

Table 5.2 Typical floating point operation times for IMS T805

Timing is for operations where both operands are normalised fp numbers.

# 6 System services

System services include all the necessary logic to initialise and sustain operation of the device. They also include error handling and analysis facilities.

#### 6.1 Power

Power is supplied to the device via the VCC and GND pins. Several of each are provided to minimise inductance within the package. All supply pins must be connected. The supply must be decoupled close to the chip by at least one 100 nF low inductance (e.g. ceramic) capacitor between VCC and GND. Four layer boards are recommended; if two layer boards are used, extra care should be taken in decoupling.

Input voltages must not exceed specification with respect to VCC and GND, even during power-up and power-down ramping, otherwise *latchup* can occur. CMOS devices can be permanently damaged by excessive periods of latchup.

# 6.2 CapPlus, CapMinus

The internally derived power supply for internal clocks requires an external low leakage, low inductance  $1\mu F$  capacitor to be connected between **CapPlus** and **CapMinus**. A ceramic capacitor is preferred, with an impedance less than 3 Ohms between 100 KHz and 10 MHz. If a polarised capacitor is used the negative terminal should be connected to **CapMinus**. Total PCB track length should be less than 50 mm. The connections must not touch power supplies or other noise sources.

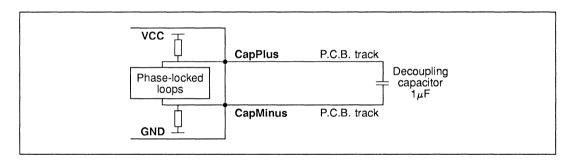


Figure 6.1 Recommended PLL decoupling

#### 6.3 Clockin

Transputer family components use a standard clock frequency, supplied by the user on the **ClockIn** input. The nominal frequency of this clock for all transputer family components is 5 MHz, regardless of device type, transputer word length or processor cycle time. High frequency internal clocks are derived from **ClockIn**, simplifying system design and avoiding problems of distributing high speed clocks externally.

A number of transputer devices may be connected to a common clock, or may have individual clocks providing each one meets the specified stability criteria. In a multi-clock system the relative phasing of **ClockIn** clocks is not important, due to the asynchronous nature of the links. Mark/space ratio is unimportant provided the specified limits of **ClockIn** pulse widths are met.

Oscillator stability is important. ClockIn must be derived from a crystal oscillator; RC oscillators are not sufficiently stable. ClockIn must not be distributed through a long chain of buffers. Clock edges must be monotonic and remain within the specified voltage and time limits.

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTE
TDCLDCH	ClockIn pulse width low	40			ns	1
TDCHDCL	ClockIn pulse width high	40			ns	1
TDCLDCL	ClockIn period		200		ns	1,2,4
TDCerror	ClockIn timing error	1	ļ	±0.5	ns	1,3
TDC1DC2	Difference in ClockIn for 2 linked devices			400	ppm	1,4
TDCr	ClockIn rise time			10	ns	1,5
TDCf	ClockIn fall time			8	ns	1,5

Table 6.1 Input clock

#### Notes

- 1 These parameters are not tested.
- 2 Measured between corresponding points on consecutive falling edges.
- 3 Variation of individual falling edges from their nominal times.
- 4 This value allows the use of 200ppm crystal oscillators for two devices connected together by a link.
- 5 Clock transitions must be monotonic within the range VIH to VIL (table 11.3).

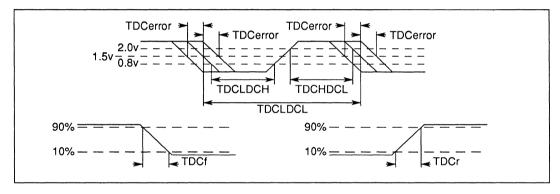


Figure 6.2 ClockIn timing

# 6.4 ProcSpeedSelect0-2

Processor speed of the IMS T805 is variable in discrete steps. The desired speed can be selected, up to the maximum rated for a particular component, by the three speed select lines **ProcSpeedSelect0-2**. The pins are tied high or low, according to the table below, for the various speeds. The pins are arranged so that the IMS T805 can be plugged directly into a board designed for a IMS T425.

Only six of the possible speed select combinations are currently used; the other two are not valid speed selectors. The frequency of **ClockIn** for the speeds given in the table is 5 MHz.

Proc Speed Select2	Proc Speed Select1	Proc Speed Select0	Processor Clock Speed MHz	Processor Cycle Time ns	Notes
0	0	0	20.0	50.0	
0	0	1	22.5	44.4	
0	1	0	25.0	40.0	1
0	1	1	30.0	33.3	
1	0	0	35.0	28.6	
1	0	1			Invalid
1	1	0	17.5	57.1	
1	1	1			Invalid

Table 6.2 Processor speed selection

Note: Inclusion of a speed selection in this table does not imply immediate availability.

## 6.5 Reset

Reset can go high with VCC, but must at no time exceed the maximum specified voltage for VIH. After VCC is valid ClockIn should be running for a minimum period TDCVRL before the end of Reset. The falling edge of Reset initialises the transputer, triggers the memory configuration sequence and starts the bootstrap routine. Link outputs are forced low during reset; link inputs and EventReq should be held low. Memory request (DMA) must not occur whilst Reset is high but can occur before bootstrap (page 94).

After the end of Reset there will be a delay of 144 periods of ClockIn (figure 6.3). Following this, the MemWrD0, MemRfD1 and MemAD2-31 pins will be scanned to check for the existence of a pre-programmed memory interface configuration (page 96). This lasts for a further 144 periods of ClockIn. Regardless of whether a configuration was found, 36 configuration read cycles will then be performed on external memory using the default memory configuration (page 98), in an attempt to access the external configuration ROM. A delay will then occur, its period depending on the actual configuration. Finally eight complete and consecutive refresh cycles will initialise any dynamic RAM, using the new memory configuration. If the memory configuration does not enable refresh of dynamic RAM the refresh cycles will be replaced by an equivalent delay with no external memory activity.

If **BootFromRom** is high bootstrapping will then take place immediately, using data from external memory; otherwise the transputer will await an input from any link. The processor will be in the low priority state.

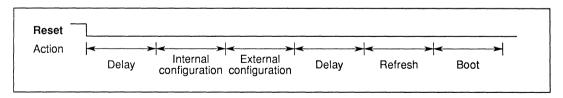


Figure 6.3 IMS T805 post-reset sequence

### 6.6 Bootstrap

The transputer can be bootstrapped either from a link or from external ROM. To facilitate debugging, **Boot-FromRom** may be dynamically changed but must obey the specified timing restrictions. It is sampled once only by the transputer, before the first instruction is executed after **Reset** is taken low.

If **BootFromRom** is connected high (e.g. to **VCC**) the transputer starts to execute code from the top two bytes in external memory, at address #7FFFFFFE. This location should contain a backward jump to a program in

ROM. Following this access, **BootFromRom** may be taken low if required. The processor is in the low priority state, and the *W* register points to *MemStart* (page 77).

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTE
TPVRH	Power valid before Reset	10			ms	
TRHRL	Reset pulse width high	8			Clockin	1
TDCVRL	Clockin running before Reset end	10			ms	2
TAHRH	Analyse setup before Reset	3	l		ms	
TRLAL	Analyse hold after Reset end	1			ClockIn	1
TBRVRL	BootFromRom setup	0			ms	
TRLBRX	BootFromRom hold after Reset	0			ms	3
TALBRX	BootFromRom hold after Analyse				1	3

Table 6.3 Reset and Analyse

#### Notes

- 1 Full periods of Clockin TDCLDCL required.
- 2 At power-on reset.
- 3 Must be stable until after end of bootstrap period. See Bootstrap section.

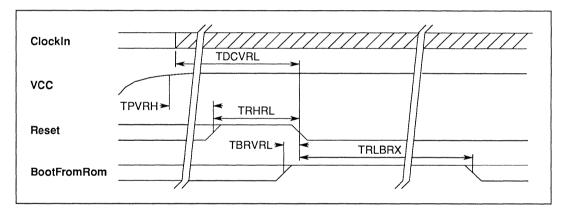


Figure 6.4 Transputer reset timing with Analyse low

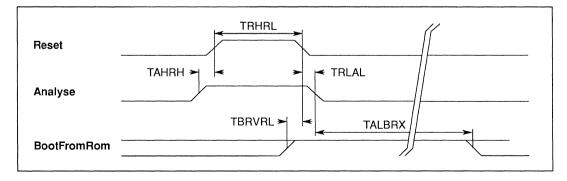


Figure 6.5 Transputer reset and analyse timing

If **BootFromRom** is connected low (e.g. to **GND**) the transputer will wait for the first bootstrap message to arrive on any one of its links. The transputer is ready to receive the first byte on a link within two processor cycles **TPCLPCL** after **Reset** goes low.

If the first byte received (the control byte) is greater than 1 it is taken as the quantity of bytes to be input. The following bytes, to that quantity, are then placed in internal memory starting at location *MemStart*. Following reception of the last byte the transputer will start executing code at *MemStart* as a low priority process. **BootFromRom** may be taken high after reception of the last byte, if required. The memory space immediately above the loaded code is used as work space. Messages arriving on other links after the control byte has been received and on the bootstrapping link after the last bootstrap byte will be retained until a process inputs from them.

# 6.7 Peek and poke

Any location in internal or external memory can be interrogated and altered when the transputer is waiting for a bootstrap from link. If the control byte is 0 then eight more bytes are expected on the same link. The first four byte word is taken as an internal or external memory address at which to poke (write) the second four byte word. If the control byte is 1 the next four bytes are used as the address from which to peek (read) a word of data; the word is sent down the output channel of the same link.

Following such a peek or poke, the transputer returns to its previously held state. Any number of accesses may be made in this way until the control byte is greater than 1, when the transputer will commence reading its bootstrap program. Any link can be used, but addresses and data must be transmitted via the same link as the control byte.

# 6.8 Analyse

If **Analyse** is taken high when the transputer is running, the transputer will halt at the next descheduling point (page 60). From **Analyse** being asserted, the processor will halt within three time slice periods plus the time taken for any high priority process to complete. As much of the transputer status is maintained as is necessary to permit analysis of the halted machine. Processor flags **Error**, **HaltOnError** and **EnableJ0Break** are normally cleared at reset on the IMS T805; however, if **Analyse** is asserted the flags are not altered. Memory refresh continues.

Input links will continue with outstanding transfers. Output links will not make another access to memory for data but will transmit only those bytes already in the link buffer. Providing there is no delay in link acknowledgement, the links should be inactive within a few microseconds of the transputer halting.

Reset should not be asserted before the transputer has halted and link transfers have ceased. When Reset is taken low whilst Analyse is high, neither the memory configuration sequence nor the block of eight refresh cycles will occur; the previous memory configuration will be used for any external memory accesses. If BootFromRom is high the transputer will bootstrap as soon as Analyse is taken low, otherwise it will await a control byte on any link. If Analyse is taken low without Reset going high the transputer state and operation are undefined. After the end of a valid Analyse sequence the registers have the values given in table 6.4.

Table 6.4 Register values after Analyse

- I MemStart if bootstrapping from a link, or the external memory bootstrap address if bootstrapping from ROM.
- W MemStart if bootstrapping from ROM, or the address of the first free word after the bootstrap program if bootstrapping from link.
- A The value of I when the processor halted.
- B The value of W when the processor halted, together with the priority of the process when the transputer was halted (i.e. the W descriptor).
- C The ID of the bootstrapping link if bootstrapping from link.

## 6.9 Error, ErrorIn

The **Error** pin carries the OR'ed output of the internal *Error* flag and the **ErrorIn** input. If **Error** is high it indicates either that **ErrorIn** is high or that an error was detected in one of the processes. An internal error can be caused, for example, by arithmetic overflow, divide by zero, array bounds violation or software setting the flag directly (page 61). It can also be set from the floating point unit under certain circumstances (page 61, 69). Once set, the *Error* flag is only cleared by executing the instruction *testerr*. The error is not cleared by processor reset, in order that analysis can identify any errant transputer (page 75).

A process can be programmed to stop if the *Error* flag is set; it cannot then transmit erroneous data to other processes, but processes which do not require that data can still be scheduled. Eventually all processes which rely, directly or indirectly, on data from the process in error will stop through lack of data. **ErrorIn** does not directly affect the status of a processor in any way.

By setting the *HaltOnError* flag the transputer itself can be programmed to halt if *Error* becomes set. If *Error* becomes set after *HaltOnError* has been set, all processes on that transputer will cease but will not necessarily cause other transputers in a network to halt. Setting *HaltOnError* after *Error* will not cause the transputer to halt; this allows the processor reset and analyse facilities to function with the flags in indeterminate states.

An alternative method of error handling is to have the errant process or transputer cause all transputers to halt. This can be done by 'daisy-chaining' the **ErrorIn** and **Error** pins of a number of processors and applying the final **Error** output signal to the **EventReq** pin of a suitably programmed master transputer. Since the process state is preserved when stopped by an error, the master transputer can then use the analyse function to debug the fault. When using such a circuit, note that the *Error* flag is in an indeterminate state on power up; the circuit and software should be designed with this in mind.

Error checks can be removed completely to optimise the performance of a proven program; any unexpected error then occurring will have an arbitrary undefined effect.

If a high priority process pre-empts a low priority one, status of the *Error* and *HaltOnError* flags is saved for the duration of the high priority process and restored at the conclusion of it. Status of both flags is transmitted to the high priority process. Either flag can be altered in the process without upsetting the error status of any complex operation being carried out by the pre-empted low priority process.

In the event of a transputer halting because of *HaltOnError*, the links will finish outstanding transfers before shutting down. If **Analyse** is asserted then all inputs continue but outputs will not make another access to memory for data. Memory refresh will continue to take place.

After halting due to the *Error* flag changing from 0 to 1 whilst *HaltOnError* is set, register *I* points two bytes past the instruction which set *Error*. After halting due to the **Analyse** pin being taken high, register *I* points one byte past the instruction being executed. In both cases *I* will be copied to register *A*.

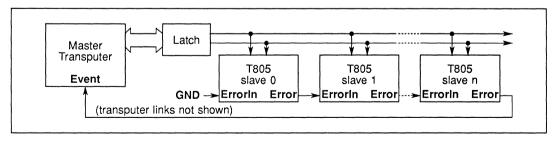


Figure 6.6 Error handling in a multi-transputer system

# 7 Memory

The IMS T805 has 4 Kbytes of fast internal static memory for high rates of data throughput. Each internal memory access takes one processor cycle **ProcClockOut** (page 82). The transputer can also access 4 Gbytes of external memory space. Internal and external memory are part of the same linear address space. Internal RAM can be disabled by holding **DisableIntRAM** high. All internal addresses are then mapped to external RAM. This pin should not be altered after **Reset** has been taken low.

IMS T805 memory is byte addressed, with words aligned on four-byte boundaries. The least significant byte of a word is the lowest addressed byte.

The bits in a byte are numbered 0 to 7, with bit 0 the least significant. The bytes are numbered from 0, with byte 0 the least significant. In general, wherever a value is treated as a number of component values, the components are numbered in order of increasing numerical significance, with the least significant component numbered 0. Where values are stored in memory, the least significant component value is stored at the lowest (most negative) address.

Internal memory starts at the most negative address #80000000 and extends to #80000FFF. User memory begins at #80000070; this location is given the name *MemStart*. An instruction *Idmemstartval* is provided to obtain the value of **MemStart**.

The context of a process in the transputer model involves a workspace descriptor (WPtr) and an instruction pointer (IPtr). WPtr is a word address pointer to a workspace in memory. IPtr points to the next instruction to be executed for the process which is the currently executing process. The context switch performed by the breakpoint instruction swaps the WPtr and IPtr of the currently executing process with the WPtr and IPtr and IPtr of the currently executing process with the WPtr and IPtr had above MemStart, one for high priority and one for low priority; this allows processes at both levels to have breakpoints. Note that on bootstrapping from a link, these contexts are overwritten by the loaded code. If this is not acceptable, the values should be peeked from memory before bootstrapping from a link.

The reserved area of internal memory below *MemStart* is used to implement link and event channels.

Two words of memory are reserved for timer use, *TPtrLoc0* for high priority processes and *TPtrLoc1* for low priority processes. They either indicate the relevant priority timer is not in use or point to the first process on the timer queue at that priority level.

Values of certain processor registers for the current low priority process are saved in the reserved *IntSaveLoc* locations when a high priority process pre-empts a low priority one. Other locations are reserved for extended features such as block moves and floating point operations.

External memory space starts at #80001000 and extends up through #00000000 to #7FFFFFFF. Memory configuration data and ROM bootstrapping code must be in the most positive address space, starting at #7FFFFF6C and #7FFFFFFE respectively. Address space immediately below this is conventionally used for ROM based code.

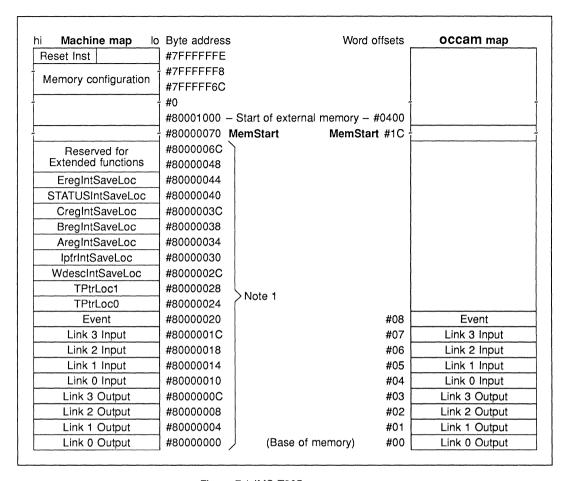


Figure 7.1 IMS T805 memory map

# 8 External memory interface

The External Memory Interface (EMI) allows access to a 32 bit address space, supporting dynamic and static RAM as well as ROM and EPROM. EMI timing can be configured at **Reset** to cater for most memory types and speeds, and a program is supplied with the Transputer Development System to aid in this configuration.

There are 17 internal configurations which can be selected by a single pin connection (page 96). If none are suitable the user can configure the interface to specific requirements, as shown in page 98.

The timing parameters in the following tables are based on 17 MHz and 20 MHz parts. Data for higher speeds is based on tests on a limited number of samples and may change when full characterisation is completed.

The external memory cycle is divided into six Tstates with the following functions:

- T1 Address setup time before address valid strobe.
- T2 Address hold time after address valid strobe.
- **T3** Read cycle tristate or write cycle data setup.
- T4 Extendable data setup time.
- T5 Read or write data.
- T6 Data hold.

Under normal conditions each **Tstate** may be from one to four periods **Tm** long, the duration being set during memory configuration. The default condition on **Reset** is that all **Tstates** are the maximum four periods **Tm** long to allow external initialisation cycles to read slow ROM.

Period T4 can be extended indefinitely by adding externally generated wait states.

An external memory cycle is always an even number of periods **Tm** in length and the start of **T1** always coincides with a rising edge of **ProcClockOut**. If the total configured quantity of periods **Tm** is an odd number, one extra period **Tm** will be added at the end of **T6** to force the start of the next **T1** to coincide with a rising edge of **ProcClockOut**. This period is designated **E** in configuration diagrams (figure 8.19).

During an internal memory access cycle the external memory interface bus **MemAD2-31** reflects the word address used to access internal RAM, **MemnotWrD0** reflects the read/write operation and **MemnotRfD1** is high; all control strobes are inactive. This is true unless and until a memory refresh cycle or DMA (memory request) activity takes place, when the bus will carry the appropriate external address or data.

The bus activity is not adequate to trace the internal operation of the transputer in full, but may be used for hardware debugging in conjuction with peek and poke (page 75).

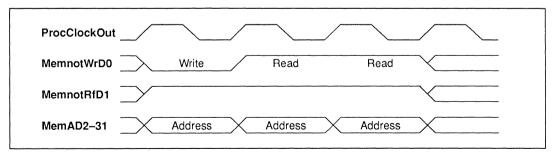


Figure 8.1 IMS T805 bus activity for internal memory cycle

#### 8.1 Pin functions

### 8.1.1 MemAD2-31

External memory addresses and data are multiplexed on one bus. Only the top 30 bits of address are output on the external memory interface, using pins MemAD2-31. They are normally output only during Tstates T1 and T2, and should be latched during this time. The data bus is 32 bits wide. It uses MemAD2-31 for the top 30 bits and MemnotRfD1 and MemnotWrD0 for the lower two bits.

#### 8.1.2 notMemRd

For a read cycle the read strobe **notMemRd** is low during **T4** and **T5**. Data is read by the transputer on the rising edge of this strobe, and may be removed immediately afterward. If the strobe duration is insufficient it may be extended by adding extra periods **Tm** to either or both of the **Tstates T4** and **T5**. Further extension may be obtained by inserting wait states at the end of **T4**.

#### 8.1.3 MemnotWrD0

During T1 and T2 this pin will be low if the cycle is a write cycle, otherwise it will be high. During Tstates T3 to T6 it becomes bit 0 of the data bus. In both cases it follows the general timing of MemAD2-31.

#### 8.1.4 notMemWrB0-3

Because the transputer uses word addressing, four write strobes are provided; one to write each byte of the word. notMemWrB0 addresses the least significant byte.

#### 8.1.5 notMemS0-4

To facilitate control of different types of memory and devices, the EMI is provided with five strobe outputs, four of which can be configured by the user. The strobes are conventionally assigned the functions shown in the read and write cycle diagrams, although there is no compulsion to retain these designations.

#### 8.1.6 MemWait

Wait states can be selected by taking MemWait high. Externally generated wait states can be added to extend the duration of T4 indefinitely.

#### 8.1.7 MemnotRfD1

During T1 and T2, this pin is low if the address on MemAD2-31 is a refresh address, otherwise it is high. During Tstates T3 to T6 it becomes bit 1 of the data bus. In both cases it follows the general timing of MemAD2-31.

## 8.1.8 notMemRf

The IMS T805 can be operated with memory refresh enabled or disabled. The selection is made during memory configuration, when the refresh interval is also determined.

# 8.1.9 RefreshPending

When high, this pin signals that a refresh cycle is pending.

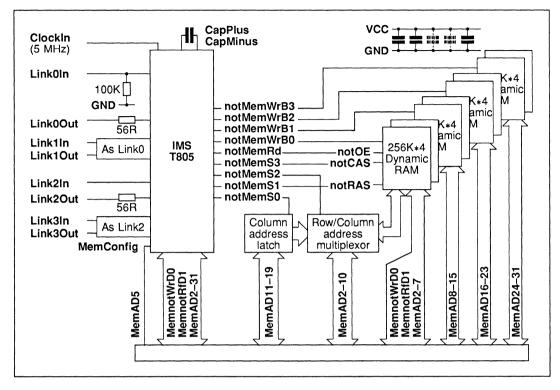


Figure 8.2 IMS T805 dynamic RAM application

# 8.1.10 MemReq, MemGranted

Direct memory access (DMA) can be requested at any time by driving the asynchronous MemReq input high.

**MemGranted** follows the timing of the bus being tristated and can be used to signal to the device requesting the DMA that it has control of the bus. Note that **MemGranted** changes on the falling edge of **ProcClockOut** and can therefore be sampled to establish control of the bus on the rising edge of **ProcClockOut**.

# 8.1.11 MemConfig

MemConfig is an input pin used to read configuration data when setting external memory interface (EMI) characteristics.

### 8.1.12 ProcClockOut

This clock is derived from the internal processor clock, which is in turn derived from **ClockIn**. Its period is equal to one internal microcode cycle time, and can be derived from the formula

### TPCLPCL = TDCLDCL / PLLx

where TPCLPCL is the ProcClockOut Period, TDCLDCL is the ClockIn Period and PLLx is the phase lock loop factor for the relevant speed part, obtained from the ordering details (Ordering section).

The time value **Tm** is used to define the duration of **Tstates** and, hence, the length of external memory cycles; its value is exactly half the period of one **ProcClockOut** cycle (0.5\***TPCLPCL**), regardless of mark/space ratio of **ProcClockOut**.

Edges of the various external memory strobes coincide with rising or falling edges of **ProcClockOut**. It should be noted, however, that there is a skew associated with each coincidence. The value of skew depends on whether coincidence occurs when the **ProcClockOut** edge and strobe edge are both rising, when both are falling or if either is rising when the other is falling. Timing values given in the strobe tables show the best and worst cases. If a more accurate timing relationship is required, the exact **Tstate** timing and strobe edge to **ProcClockOut** relationships should be calculated and the correct skew factors applied from the edge skew timing table 8.4.

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTE
TPCLPCL	ProcClockOut period	<b>a</b> -2	а	<b>a</b> +2	ns	1,5
TPCHPCL	ProcClockOut pulse width high	<b>b</b> -11.5	b	<b>b</b> +3.5	ns	2,5
TPCLPCH	ProcClockOut pulse width low		С	I	ns	3,5
Tm	ProcClockOut half cycle	<b>b</b> -1	b	b+1	ns	2,5
TPCstab	ProcClockOut stability			8	%	4,5

Table 8.1 ProcClockOut

#### Notes

- 1 a is TDCLDCL/PLLx.
- 2 b is 0.5\*TPCLPCL (half the processor clock period).
- 3 c is TPCLPCL-TPCHPCL.
- 4 Stability is the variation of cycle periods between two consecutive cycles, measured at corresponding points on the cycles.
- 5 This parameter is sampled and not 100% tested.

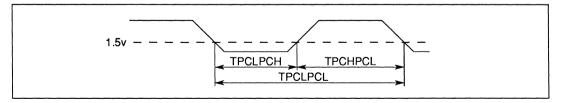


Figure 8.3 IMS T805 ProcClockOut timing

## 8.2 Read cycle

Byte addressing is carried out internally by the transputer for read cycles. For a read cycle the read strobe **notMemRd** is low during **T4** and **T5**. Read cycle data may be set up on the data bus at any time after the start of **T3**, but must be valid when the transputer reads it at the end of **T5**. Data may be removed any time during **T6**, but must be off the bus no later than the end of that period.

notMemS0 is a fixed format strobe. Its leading edge is always coincident with the start of T2 and its trailing edge always coincident with the end of T5.

The leading edge of **notMemS1** is always coincident with the start of **T2**, but its duration may be configured to be from zero to 31 periods **Tm**. Regardless of the configured duration, the strobe will terminate no later than the end of **T6**. The strobe is sometimes programmed to extend beyond the normal end of **Tmx**. When wait states are inserted into an EMI cycle the end of **Tmx** is delayed, but the potential active duration of the strobe is not altered. Thus the strobe can be configured to terminate relatively early under certain conditions (page 89). If **notMemS1** is configured to be zero it will never go low.

notMemS2, notMemS3 and notMemS4 are identical in operation. They all terminate at the end of T5, but the start of each can be delayed from one to 31 periods Tm beyond the start of T2. If the duration of one of these strobes would take it past the end of T5 it will stay high. This can be used to cause a strobe to become active only when wait states are inserted. If one of these strobes is configured to zero it will never go low. Figure 8.6 shows the effect of Wait on strobes in more detail; each division on the scale is one period Tm.

In the read cycle timing diagrams **ProcClockOut** is included as a guide only; it is shown with each **Tstate** configured to one period **Tm**.

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTE
TaZdV	Address tristate to data valid	0			ns	
TdVRdH	Data setup before read	25			ns	
TRdHdX	Data hold after read	0		1	ns	
TS0LRdL	notMemS0 before start of read	a-4	а	a+4	ns	1
TS0HRdH	End of read from end of notMemS0	-4	}	4	ns	
TRdLRdH	Read period	<b>b</b> -3		<b>b</b> +5	ns	2

Table 8.2 Read

#### Notes

- 1 a is total of T2+T3 where T2, T3 can be from one to four periods Tm each in length.
- 2 b is total of T4+Twait+T5 where T4, T5 can be from one to four periods Tm each in length and Twait may be any number of periods Tm in length.

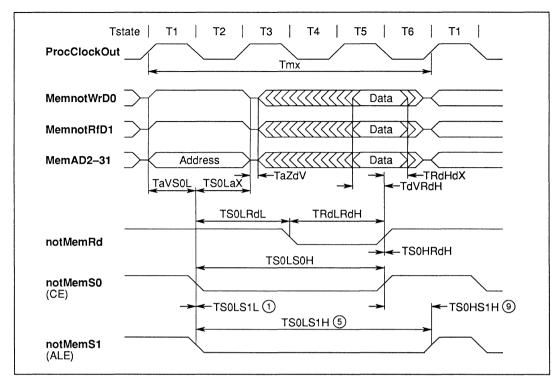


Figure 8.4 IMS T805 external read cycle: static memory

SYMBOL	n	PARAMETER	MIN	NOM	MAX	UNITS	NOTE
TaVS0L		Address setup before notMemS0	<b>a</b> -8			ns	1
TS0LaX		Address hold after notMemS0	<b>b</b> -8	b	<b>b</b> +8	ns	2
TS0LS0H	l	notMemS0 pulse width low	<b>c</b> -5		<b>c</b> +6	ns	3
TS0LS1L	1	notMemS1 from notMemS0	-4		4	ns	
TS0LS1H	5	notMemS1 end from notMemS0	d-1		<b>d</b> +9	ns	4,6
TS0HS1H	9	notMemS1 end from notMemS0 end	<b>e</b> -8		e+4	ns	5,6
TS0LS2L	2	notMemS2 delayed after notMemS0	f-6		f+5	ns	7
TS0LS2H	6	notMemS2 end from notMemS0	<b>c</b> -5		c+7	ns	3
TS0HS2H	10	notMemS2 end from notMemS0 end	-4	]	7	ns	
TS0LS3L	3	notMemS3 delayed after notMemS0	f-6		f+5	ns	7
TS0LS3H	7	notMemS3 end from notMemS0	<b>c</b> -5	1	c+7	ns	3
TS0HS3H	11	notMemS3 end from notMemS0 end	-4		7	ns	
TS0LS4L	4	notMemS4 delayed after notMemS0	f-6		f+5	ns	7
TS0LS4H	8	notMemS4 end from notMemS0	<b>c</b> -5		<b>c</b> +7	ns	3
TS0HS4H	12	notMemS4 end from notMemS0 end	-4	ĺ	7	ns	
Tmx		Complete external memory cycle		g			8

Table 8.3 IMS T805 strobe timing

- 1 a is T1 where T1 can be from one to four periods Tm in length.
- 2 b is T2 where T2 can be from one to four periods Tm in length.
- 3 c is total of T2+T3+T4+Twait+T5 where T2, T3, T4, T5 can be from one to four periods Tm each in length and Twait may be any number of periods Tm in length.
- 4 d can be from zero to 31 periods Tm in length.
- 5 e can be from -27 to +4 periods Tm in length.
- 6 If the configuration would cause the strobe to remain active past the end of **T6** it will go high at the end of **T6**. If the strobe is configured to zero periods **Tm** it will remain high throughout the complete cycle **Tmx**.
- 7 f can be from zero to 31 periods Tm in length. If this length would cause the strobe to remain active past the end of T5 it will go high at the end of T5. If the strobe value is zero periods Tm it will remain low throughout the complete cycle Tmx.
- 8 g is one complete external memory cycle comprising the total of T1+T2+T3+T4+Twait+T5+T6 where T1, T2, T3, T4, T5 can be from one to four periods Tm each in length, T6 can be from one to five periods Tm in length and Twait may be zero or any number of periods Tm in length.

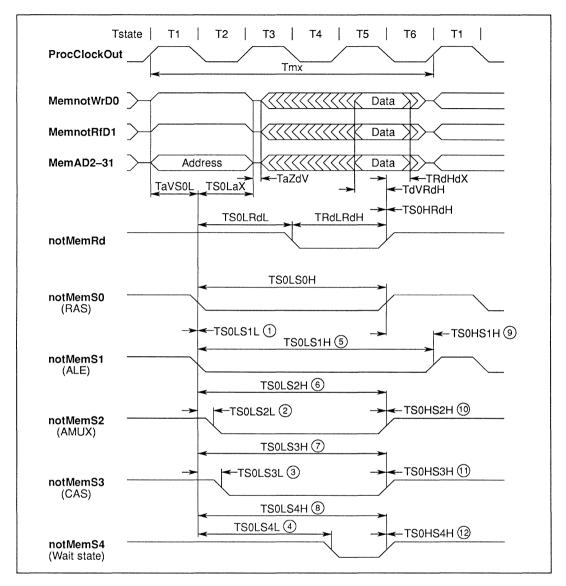


Figure 8.5 IMS T805 external read cycle: dynamic memory

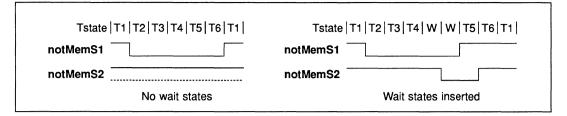


Figure 8.6 IMS T805 effect of wait states on strobes

Table 8.4 Strobe S0 to ProcClockOut skew

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTE
TPCHS0H	notMemS0 rising from ProcClockOut rising	-6		4	ns	
TPCLS0H	notMemS0 rising from ProcClockOut falling	-5		10	ns	
TPCHS0L	notMemS0 falling from ProcClockOut rising	-8		3	ns	
TPCLS0L	notMemS0 falling from ProcClockOut falling	-5		7	ns	

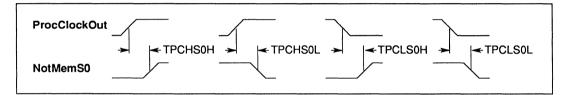


Figure 8.7 IMS T805 skew of notMemS0 to ProcClockOut

### 8.3 Write cycle

For write cycles the relevant bytes in memory are addressed by the write strobes **notMemWrB0-3**. If a particular byte is not to be written, then the corresponding data outputs are tristated.

For a write cycle pin **MemnotWrD0** will be low during **T1** and **T2**. Write data is placed on the bus at the start of **T3** and removed at the end of **T6**. If **T6** is extended to force the next cycle **Tmx** (page 80) to start on a rising edge of **ProcClockOut**, data will be valid during this time also.

The transputer has both early and late write cycle modes. For a late write cycle the relevant write strobes notMemWrB0-3 are low during T4 and T5; for an early write they are also low during T3. Data should be latched into memory on the rising edge of the strobes in both cases, although it is valid until the end of T6. If the strobe duration is insufficient, it may be extended at configuration time by adding extra periods Tm to either or both of Tstates T4 and T5 for both early and late modes. For an early cycle they may also be added to T3. Further extension may be obtained by inserting wait states at the end of T4. If the data hold time is insufficient, extra periods Tm may be added to T6 to extend it.

In the write cycle timing diagram **ProcClockOut** is included as a guide only; it is shown with each **Tstate** configured to one period **Tm**. The strobe is inactive during internal memory cycles.

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTE
TdVWrH	Data setup before write	<b>d</b> -7		<b>d</b> +10	ns	1,5
TWrHdX	Data hold after write	<b>a</b> -10		<b>a</b> +5	ns	1,2
TS0LWrL	notMemS0 before start of early write	<b>b</b> -5		<b>b</b> +5	ns	1,3
	notMemS0 before start of late write	<b>c</b> -5		<b>c</b> +5	ns	1,4
TS0HWrH	End of write from end of notMemS0	-5		4	ns	1
TWrLWrH	Early write pulse width	<b>d</b> -4		<b>d</b> +7	ns	1,5
	Late write pulse width	<b>e</b> -4		<b>e</b> +7	ns	1,6

Table 8.5 Write

- 1 Timing is for all write strobes notMemWrB0-3.
- 2 a is T6 where T6 can be from one to five periods Tm in length.
- 3 b is T2 where T2 can be from one to four periods Tm in length.
- 4 c is total of T2+T3 where T2. T3 can be from one to four periods Tm each in length.
- 5 d is total of T3+T4+Twait+T5 where T3, T4, T5 can be from one to four periods Tm each in length and Twait may be zero or any number of periods Tm in length.
- 6 e is total of T4+Twait+T5 where T4, T5 can be from one to four periods Tm each in length and Twait may be zero or any number of periods Tm in length.

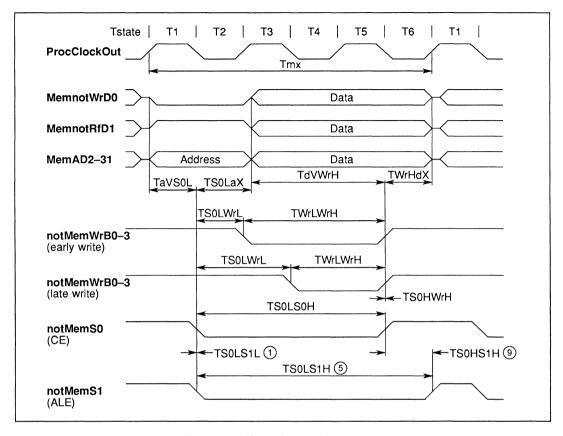


Figure 8.8 IMS T805 external write cycle

#### 8.4 Wait

Taking **MemWait** high with the timing shown (figure 8.9) will extend the duration of **T4. MemWait** is sampled relative to the falling edge of **ProcClockOut** during a **T3** period, and should not change state in this region. By convention, **notMemS4** is used to synchronize wait state insertion. If this or another strobe is used, its delay should be such as to take the strobe low an even number of periods **Tm** after the start of **T1**, to coincide with a rising edge of **ProcClockOut**.

**MemWait** may be kept high indefinitely, although if dynamic memory refresh is used it should not be kept high long enough to interfere with refresh timing. **MemWait** operates normally during all cycles, including refresh and configuration cycles. It does not affect internal memory access in any way.

If the start of **T5** would coincide with a falling edge of **ProcClockOut** an extra wait period **Tm** (**EW**) is generated by the EMI to force coincidence with a rising edge. Rising edge coincidence is only forced if wait states are added, otherwise coincidence with a falling edge is permitted.

Table 8.6 Memory	v wait	
------------------	--------	--

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTE
TPCLWtH	Wait setup	-(0.5 <b>Tm</b> +9)			ns	1,2
TPCLWtL	Wait hold	0.5 <b>Tm</b> +10			ns	1,2
TWtLWtH	Delay before re-assertion of Wait	2Tm				

- 1 ProcClockOut load should not exceed 50pf.
- 2 If wait period exceeds refresh interval, refresh cycles will be lost.

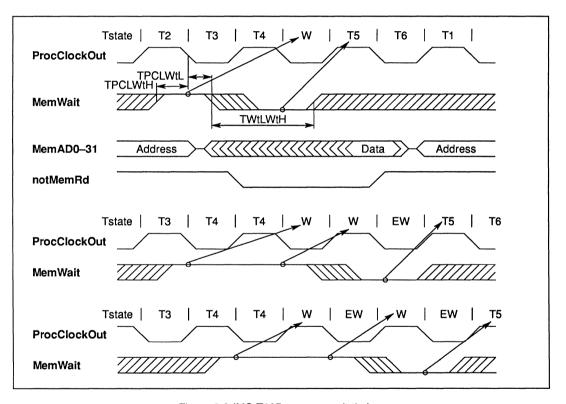


Figure 8.9 IMS T805 memory wait timing

### 8.5 Memory refresh

The RefreshPending pin is asserted high when the external memory interface is about to perform a refresh cycle. It remains high until the refresh cycle is started by the transputer. The mimimum time for the RefreshPending pin to be high is for one cycle of ProcClockOut (two periods Tm), when the EMI was not about to perform a memory read or write. If the EMI was held in the tristate condition with MemGranted asserted, then RefreshPending will be asserted when the refresh controller in the EMI is ready to perform a refresh. MemReq may be re-asserted any time after the commencement of the refresh cycle. Refresh-Pending changes state near the rising edge of ProcClockOut and can therefore be sampled by the falling edge of ProcClockOut.

If no DMA is active then refresh will be performed following the end of the current internal or external memory cycle. If DMA is active the transputer will wait for DMA to terminate before commencing the refresh cycle. Unlike **MemnotRfD1**, **RefreshPending** is never tristated and can thus be interrogated by the DMA device; the DMA cycle can then be suspended, at the discretion of the DMA device, to allow refresh to take place.

The simple circuit of Figure 8.10 will suspend DMA requests from the external logic when **RefreshPending** is asserted, so that a memory refresh cycle can be performed. DMA is restored on completion of the refresh cycle. The transputer will not perform an external memory cycle other than a refresh cycle, using this method, until the requesting device removes its DMA request.

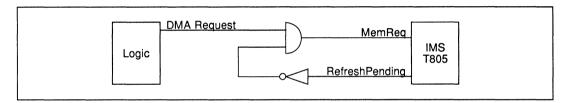


Figure 8.10 IMS T805 refresh with DMA

When refresh is disabled no refresh cycles occur. During the post-Reset period eight dummy refresh cycles will occur with the appropriate timing but with no bus or strobe activity.

A refresh cycle uses the same basic external memory timing as a normal external memory cycle, except that it starts two periods **Tm** before the start of **T1**. If a refresh cycle is due during an external memory access, it will be delayed until the end of that external cycle. Two extra periods **Tm** (periods **R** in the diagram) will then be inserted between the end of **T6** of the external memory cycle and the start of **T1** of the refresh cycle itself. The refresh address and various external strobes become active approximately one period **Tm** before **T1**. Bus signals are active until the end of **T2**, whilst **notMemRf** remains active until the end of **T6**.

For a refresh cycle, MemnotRfD1 goes low before notMemRf goes low and MemnotWrD0 goes high with the same timing as MemnotRfD1. All the address lines share the same timing, but only MemAD2-11 give the refresh address. MemAD12-30 stay high during the address period, whilst MemAD31 remains low. Refresh cycles generate strobes notMemS0-4 with timing as for a normal external cycle, but notMemRd and notMemWrB0-3 remain high. MemWait operates normally during refresh cycles.

Refresh cycles do not interrupt internal memory accesses, although the internal addresses cannot be reflected on the external bus during refresh.

Table 8.	7 Memory	refresh
----------	----------	---------

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTE
TRfLRfH	Refresh pulse width low	<b>a</b> -2		<b>a</b> +9	ns	1
TRaVS0L	Refresh address setup before notMemS0	<b>b</b> -12			ns	
TRfLS0L	Refresh indicator setup before notMemS0	<b>b</b> -4	b	<b>b</b> +6	ns	2

- 1 a is total Tmx+Tm.
- 2 b is total T1+Tm where T1 can be from one to four periods Tm in length.

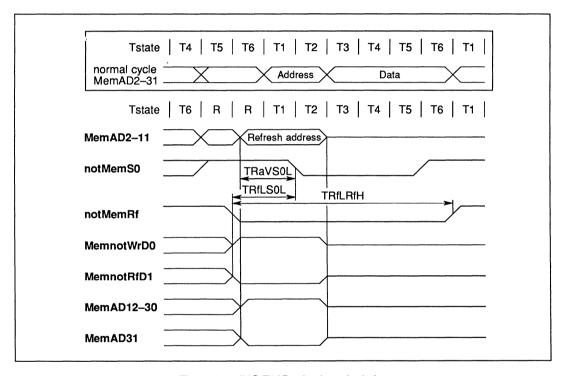


Figure 8.11 IMS T805 refresh cycle timing

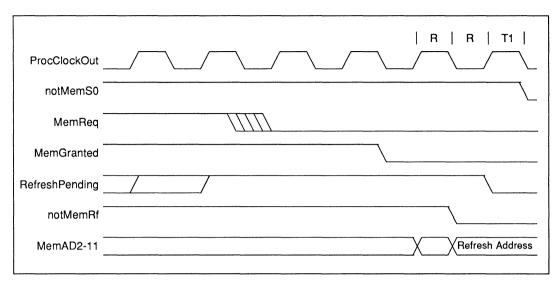


Figure 8.12 IMS T805 RefreshPending timing

### 8.6 Direct memory access

Direct memory access (DMA) can be requested at any time by taking the asynchronous **MemReq** input high. The transputer samples **MemReq** during the final period **Tm** of **T6** of both refresh and external memory cycles. To guarantee taking over the bus immediately following either, **MemReq** must be set up at least two periods **Tm** before the end of **T6**. In the absence of an external memory cycle, **MemReq** is sampled during every low period of **ProcClockOut**. The address bus is tristated two periods **Tm** after the **ProcClockOut** rising edge which follows the sample. **MemGranted** is asserted one period **Tm** after that.

Removal of **MemReq** is sampled during each low period of **ProcClockOut** and **MemGranted** is removed synchronously with the next falling edge of **ProcClockOut**. If accurate timing of DMA is required, **MemReq** should be set low coincident with a falling edge of **ProcClockOut**. Further external bus activity, either refresh, external cycles or reflection of internal cycles, will commence at the next rising edge of **ProcClockOut**.

Strobes are left in their inactive states during DMA. DMA cannot interrupt a refresh or external memory cycle, and outstanding refresh cycles will occur before the bus is released to DMA. DMA does not interfere with internal memory cycles in any way, although a program running in internal memory would have to wait for the end of DMA before accessing external memory. DMA cannot access internal memory. If DMA extends longer than one refresh interval (Memory Refresh Configuration Coding, table 8.11), the DMA user becomes responsible for refresh. DMA may also inhibit an internally running program from accessing external memory.

DMA allows a bootstrap program to be loaded into external RAM ready for execution after reset. If **MemReq** is held high throughout reset, **MemGranted** will be asserted before the bootstrap sequence begins. **MemReq** must be high at least one period **TDCLDCL** of **ClockIn** before **Reset**. The circuit should be designed to ensure correct operation if **Reset** could interrupt a normal DMA cycle.

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTE
TMRHMGH	Memory request response time	4Tm-2ns		7 <b>Tm</b> +7ns		1
TMRLMGL	Memory request end response time	2Tm-2ns		5 <b>Tm</b> +22ns		
TADZMGH	Bus tristate before memory granted	Tm-2ns		Tm+22ns		
TMGLADV	Bus active after end of memory granted	-10ns		Tm+2ns		

Table 8.8 Memory request

#### Notes

1 These values assume no external memory cycle is in progress. If an external cycle is active, maximum time could be (1 EMI cycle Tmx)+(1 refresh cycle TRfLRfH)+(6 periods Tm).

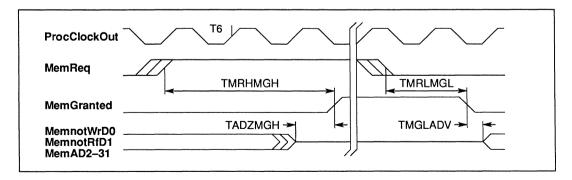


Figure 8.13 IMS T805 memory request timing

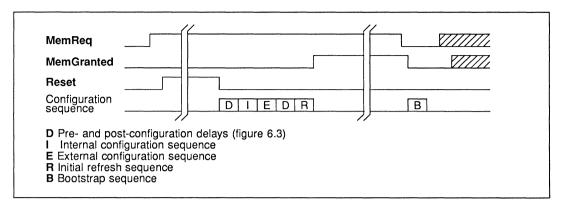


Figure 8.14 IMS T805 DMA sequence at reset

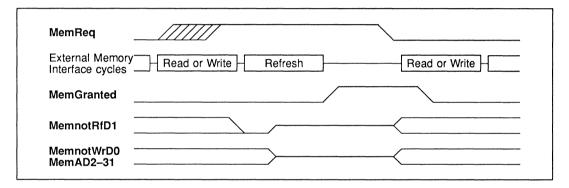


Figure 8.15 IMS T805 operation of MemReq, MemGranted with external, refresh memory cycles

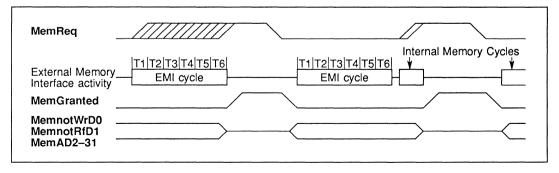


Figure 8.16 IMS T805 operation of MemReq, MemGranted with external, internal memory cycles

### 8.7 Memory configuration

**MemConfig** is an input pin used to read configuration data when setting external memory interface (EMI) characteristics. It is read by the processor on two occasions after **Reset** goes low; first to check if one of the preset internal configurations is required, then to determine a possible external configuration.

### 8.7.1 Internal configuration

The internal configuration scan comprises 64 periods TDCLDCL of ClockIn during the internal scan period of 144 ClockIn periods. MemnotWrD0, MemnotRfD1 and MemAD2-32 are all high at the beginning of the scan. Starting with MemnotWrD0, each of these lines goes low successively at intervals of two ClockIn periods and stays low until the end of the scan. If one of these lines is connected to MemConfig the preset internal configuration mode associated with that line will be used as the EMI configuration. The default configuration is that defined in the table for MemAD31; connecting MemConfig to VCC will also produce this default configuration. Note that only 17 of the possible configurations are valid, all others remain at the default configuration.

	Di	uratio p		each ds Tn		ite	(	Strobe coefficie			Write cycle	Refresh interval	Cycle time
Pin	T1	T2	ТЗ	T4	T5	Т6	s1	s2	s3	s4	type	Clockin cycles	Proc cycles
MemnotWrD0	1	1	1	1	1	1	30	1	3	5	late	72	3
MemnotRfD1	1	2	1	1	1	2	30	1	2	7	late	72	4
MemAD2	1	2	1	1	2	3	30	1	2	7	late	72	5
MemAD3	2	3	1	1	2	3	30	1	3	8	late	72	6
MemAD4	1	1	1	1	1	1	3	1	2	3	early	72	3
MemAD5	1	1	2	1	2	1	5	1	2	3	early	72	4
MemAD6	2	1	2	1	3	1	6	1	2	3	early	72	5
MemAD7	2	2	2	1	3	2	7	1	3	4	early	72	6
MemAD8	1	1	1	1	1	1	30	1	2	3	early	†	3
MemAD9	1	1	2	1	2	1	30	2	5	9	early	l †	4
MemAD10	2	2	2	2	4	2	30	2	3	8	late	72	7
MemAD11	3	3	3	3	3	3	30	2	4	13	late	72	9
MemAD12	1	1	2	1	2	1	4	1	2	3	early	72	4
MemAD13	2	1	2	1	2	2	5	1	2	3	early	72	5
MemAD14	2	2	2	1	3	2	6	1	3	4	early	72	6
MemAD15	2	1	2	3	3	3	8	1	2	3	early	72	7
MemAD31	4	4	4	4	4	4	31	30	30	18	late	72	12

Table 8.9 IMS T805 internal configuration coding

<sup>†</sup> Provided for static RAM only.

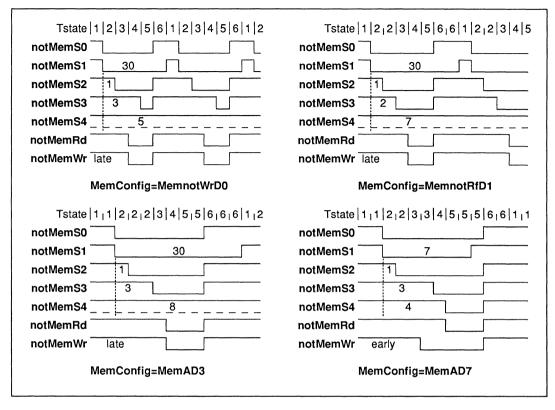


Figure 8.17 IMS T805 internal configuration

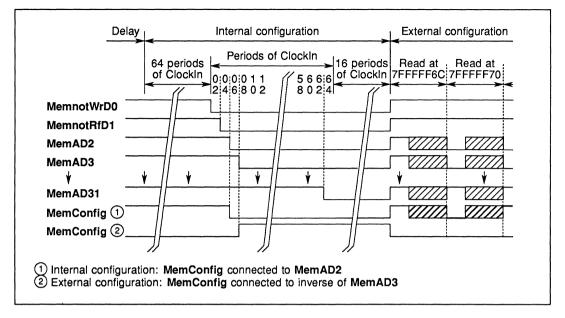


Figure 8.18 IMS T805 internal configuration scan

### 8.7.2 External configuration

If **MemConfig** is held low until **MemnotWrD0** goes low the internal configuration is ignored and an external configuration will be loaded instead. An external configuration scan always follows an internal one, but if an internal configuration occurs any external configuration is ignored.

The external configuration scan comprises 36 successive external read cycles, using the default EMI configuration preset by **MemAD31**. However, instead of data being read on the data bus as for a normal read cycle, only a single bit of data is read on **MemConfig** at each cycle. Addresses put out on the bus for each read cycle are shown in table 8.10, and are designed to address ROM at the top of the memory map. The table shows the data to be held in ROM; data required at the **MemConfig** pin is the inverse of this.

**MemConfig** is typically connected via an inverter to **MemnotWrD0**. Data bit zero of the least significant byte of each ROM word then provides the configuration data stream. By switching **MemConfig** between various data bus lines up to 32 configurations can be stored in ROM, one per bit of the data bus. **MemConfig** can be permanently connected to a data line or to **GND**. Connecting **MemConfig** to **GND** gives all **Tstates** configured to four periods; **notMemS1** pulse of maximum duration; **notMemS2-4** delayed by maximum; refresh interval 72 periods of **ClockIn**; refresh enabled; late write.

The external memory configuration table 8.10 shows the contribution of each memory address to the 13 configuration fields. The lowest 12 words (#7FFFFF6C to #7FFFF98, fields 1 to 6) define the number of extra periods **Tm** to be added to each **Tstate**. If field 2 is 3 then three extra periods will be added to **T2** to extend it to the maximum of four periods.

The next five addresses (field 7) define the duration of **notMemS1** and the following fifteen (fields 8 to 10) define the delays before strobes **notMemS2-4** become active. The five bits allocated to each strobe allow durations of from 0 to 31 periods **Tm**, as described in strobes page 80.

Addresses #7FFFFEC to #7FFFFF4 (fields 11 and 12) define the refresh interval and whether refresh is to be used, whilst the final address (field 13) supplies a high bit to **MemConfig** if a late write cycle is required.

The columns to the right of the coding table show the values of each configuration bit for the four sample

external configuration diagrams. Note the inclusion of period E at the end of T6 in some diagrams. This is inserted to bring the start of the next Tstate T1 to coincide with a rising edge of ProcClockOut (page 82).

Wait states **W** have been added to show the effect of them on strobe timing; they are not part of a configuration. In each case which includes wait states, two wait periods are defined. This shows that if a wait state would cause the start of **T5** to coincide with a falling edge of **ProcClockOut**, another period **Tm** is generated by the EMI to force it to coincide with a rising edge of **ProcClockOut**. This coincidence is only necessary if wait states are added, otherwise coincidence with a falling edge is permitted. Any configuration memory access is only permitted to be extended using wait, up to a total of 14 **ClockIn** periods.

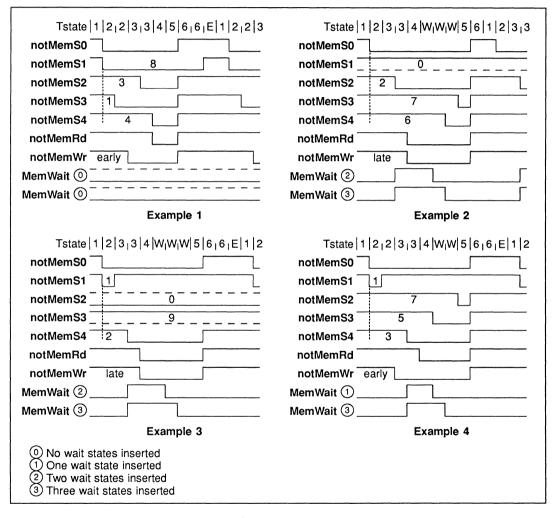


Figure 8.19 IMS T805 external configuration

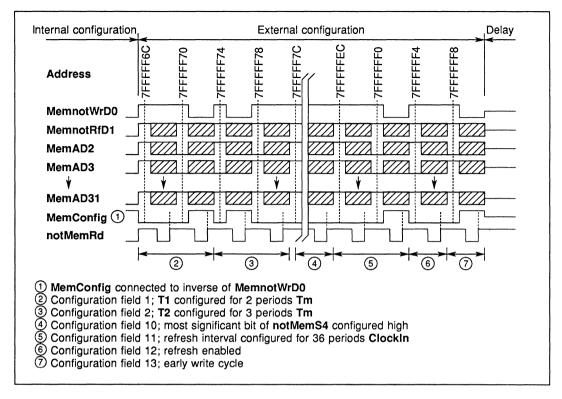


Figure 8.20 IMS T805 external configuration scan

Table 8.10 IMS T805 external configuration coding

Scan	MemAD			Exa	mple	diagi	ram
cycle	address	Field	Function	1	2	3	4
1	7FFFFF6C	1	T1 least significant bit	0	0	0	0
2	7FFFFF70	1	T1 most significant bit	0	0	0	0
3	7FFFFF74	2	T2 least significant bit	1	0	0	1
4	7FFFFF78	2	T2 most significant bit	0	0	0	0
5	7FFFFF7C	3	T3 least significant bit	1	1	1	1
6	7FFFFF80	3	T3 most significant bit	0	0	0	0
7	7FFFFF84	4	T4 least significant bit	0	0	0	0
8	7FFFFF88	4	T4 most significant bit	0	0	0	0
9	7FFFFF8C	5	T5 least significant bit	0	0	0	0
10	7FFFFF90	5	T5 most significant bit	0	0	0	0
11	7FFFFF94	6	T6 least significant bit	1	0	1	1
12	7FFFFF98	6	T6 most significant bit	0	0	0	0
13	7FFFFF9C	7	notMemS1 least significant bit	0	0	1	1
14	7FFFFFA0	7		0	0	0	0
15	7FFFFFA4	7		0	0	0	0
16	7FFFFFA8	7		1	0	0	0
17	7FFFFAC	7	notMemS1 most significant bit	0	0	0	0
18	7FFFFB0	8	notMemS2 least significant bit	1	0	0	1
19	7FFFFB4	8		1	1	0	1
20	7FFFFB8	8		0	0	, 0	1
21	7FFFFBC	8		0	0	0	0
22	7FFFFC0	8	notMemS2 most significant bit	0	0	0	0
23	7FFFFC4	9	notMemS3 least significant bit	1	1	1	1
24	7FFFFFC8	9		0	1	0	0
25	7FFFFFCC	9	↓ ↓	0	1	0	1
26	7FFFFFD0	9		0	0	1	0
27	7FFFFFD4	9	notMemS3 most significant bit	0	0	0	0
28	7FFFFFD8	10	notMemS4 least significant bit	0	0	0	1
29	7FFFFDC	10	·	0	1	1	1
30	7FFFFE0	10	]	1	1	0	0
31	7FFFFFE4	10		0	0	0	0
32	7FFFFE8	10	notMemS4 most significant bit	0	0	0	0
33	7FFFFEC	11	Refresh Interval least significant bit	-	-	-	
34	7FFFFFF0	11	Refresh Interval most significant bit	-	-	-	-
35	7FFFFFF4	12	Refresh Enable	-	-	-	-
36	7FFFFFF8	13	Late Write	0	1	1	0

Refresh interval	Interval in μs	Field 11 encoding	Complete cycle (ms)
18	3.6	00	0.922
36	7.2	01	1.843
54	10.8	10	2.765
72	14.4	11	3.686

Table 8.11 IMS T805 memory refresh configuration coding

Refresh intervals are in periods of Clockin and Clockin frequency is 5 MHz:

Interval = 18 \* 200 = 3600 ns

Refresh interval is between successive incremental refresh addresses. Complete cycles are shown for 256 row DRAMS.

Table 8.12 Memory configuration

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTE
TMCVRdH	Memory configuration data setup	25			ns	
TRdHMCX	Memory configuration data hold	0			ns	
TS0LRdH	notMemS0 to configuration data read	<b>a</b> -12		<b>a</b> +12	ns	1

### Notes

1 a is 16 periods Tm.

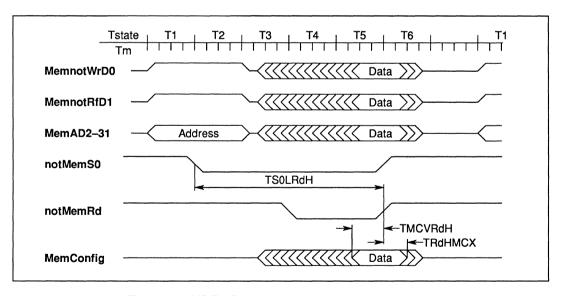


Figure 8.21 IMS T805 external configuration read cycle timing

### 9 Events

**EventReq** and **EventAck** provide an asynchronous handshake interface between an external event and an internal process. When an external event takes **EventReq** high the external event channel (additional to the external link channels) is made ready to communicate with a process. When both the event channel and the process are ready the processor takes **EventAck** high and the process, if waiting, is scheduled. **EventAck** is removed after **EventReq** goes low.

**EventWaiting** is asserted high by the transputer when a process executes an input on the event channel; typically with the occam **EVENT** ? **ANY** instruction. It remains high whilst the transputer is waiting for or servicing **EventReq** and is returned low when **EventAck** goes high. The **EventWaiting** pin changes near the falling edge of **ProcClockOut** and can therefore be sampled by the rising edge of **ProcClockOut**.

The **EventWaiting** pin can only be asserted by executing an *in* instruction on the event channel. The **EventWaiting** pin is not asserted high when an enable channel (*enbc*) instruction is executed on the Event channel (during an ALT construct in occam, for example). The **EventWaiting** pin can be asserted by executing the occam input on the event channel (such as **Event? ANY**), provided that this does not occur as a guard in an alternative process. The **EventWaiting** pin can not be used to signify that an alternative process (ALT) is waiting on an input from the event channel.

**EventWaiting** allows a process to control external logic; for example, to clock a number of inputs into a memory mapped data latch so that the event request type can be determined.

Only one process may use the event channel at any given time. If no process requires an event to occur **EventAck** will never be taken high. Although **EventReq** triggers the channel on a transition from low to high, it must not be removed before **EventAck** is high. **EventReq** should be low during **Reset**; if not it will be ignored until it has gone low and returned high. **EventAck** is taken low when **Reset** occurs.

If the process is a high priority one and no other high priority process is running, the latency is as described on page 56. Setting a high priority task to wait for an event input allows the user to interrupt a transputer program running at low priority. The time taken from asserting **EventReq** to the execution of the microcode interrupt handler in the CPU is four cycles. The following functions take place during the four cycles:

- Cycle 1 Sample EventReg at pad on the rising edge of ProcClockOut and synchronise.
- Cycle 2 Edge detect the synchronised EventReg and form the interrupt request.
- Cycle 3 Sample interrupt vector for microcode ROM in the CPU.
- **Cycle 4** Execute the interrupt routine for Event rather than the next instruction.

Table 9.1 Event

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTE
TVHKH	Event request response	0			ns	
TKHVL	Event request hold	0			ns	
TVLKL	Delay before removal of event acknowledge	0		6 <b>Tm</b> +7ns		
TKLVH	Delay before re-assertion of event request	0			ns	
TKHEWL	Event acknowledge to end of event waiting	0			ns	
TKLEWH	End of event acknowledge to event waiting	0			ns	

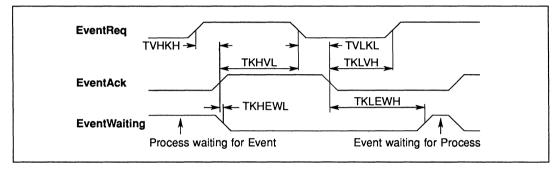


Figure 9.1 IMS T805 event timing

### 10 Links

Four identical INMOS bi-directional serial links provide synchronized communication between processors and with the outside world. Each link comprises an input channel and output channel. A link between two transputers is implemented by connecting a link interface on one transputer to a link interface on the other transputer. Every byte of data sent on a link is acknowledged on the input of the same link, thus each signal line carries both data and control information.

The quiescent state of a link output is low. Each data byte is transmitted as a high start bit followed by a one bit followed by eight data bits followed by a low stop bit. The least significant bit of data is transmitted first. After transmitting a data byte the sender waits for the acknowledge, which consists of a high start bit followed by a zero bit. The acknowledge signifies both that a process was able to receive the acknowledged data byte and that the receiving link is able to receive another byte. The sending link reschedules the sending process only after the acknowledge for the final byte of the message has been received.

The IMS T805 links support the standard INMOS communication speed of 10 Mbits/sec. In addition they can be used at 5 or 20 Mbits/sec for 17 MHz, 20 MHz, and 25 MHz devices, and 20 Mbits/sec for faster devices. Links are not synchronised with **ClockIn** or **ProcClockOut** and are insensitive to their phases. Thus links from independently clocked systems may communicate, providing only that the clocks are nominally identical and within specification.

Links are TTL compatible and intended to be used in electrically quiet environments, between devices on a single printed circuit board or between two boards via a backplane. Direct connection may be made between devices separated by a distance of less than 300 millimetres. For longer distances a matched 100 ohm transmission line should be used with series matching resistors **RM**. When this is done the line delay should be less than 0.4 bit time to ensure that the reflection returns before the next data bit is sent.

Buffers may be used for very long transmissions. If so, their overall propagation delay should be stable within the skew tolerance of the link, although the absolute value of the delay is immaterial.

Link speeds can be set by LinkSpecial, Link0Special and Link123Special. The link 0 speed can be set independently. Table 10.1 shows uni-directional and bi-directional data rates in Kbytes/sec for each link speed; LinknSpecial is to be read as Link0Special when selecting link 0 speed and as Link123Special for the others. Data rates are quoted for a transputer using internal memory, and will be affected by a factor depending on the number of external memory accesses and the length of the external memory cycle.

Link	Linkn		Kbytes/sec		
Special	Special	Mbits/sec	Uni	Bi	Notes
0	0	10	910	1250	
0	1	5	450	670	1
1	0	10	910	1250	
1	1	20	1740	2350	

Table 10.1 Speed Settings for Transputer Links

### Notes

1 This setting is reserved for IMS T805-30 and faster devices.

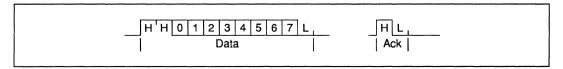


Figure 10.1 IMS T805 link data and acknowledge packets

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTE
TJQr	LinkOut rise time			20	ns	1
TJQf	LinkOut fall time		1	10	ns	1
TJDr	LinkIn rise time	1	1	20	ns	1
TJDf	LinkIn fall time	ì	1	20	ns	1
TJQJD	Buffered edge delay	0		ĺ	ns	
TJBskew	Variation in TJQJD 20 Mbits/s			3	ns	2
	10 Mbits/s	İ	ļ	10	ns	2
İ	5 Mbits/s		1	30	ns	2
CLIZ	LinkIn capacitance @ f=1MHz		1	7	pF	1
CLL	LinkOut load capacitance			50	pF	
RM	Series resistor for 100Ω transmission line		56		ohms	

Table 10.2 Link

- 1 These parameters are sampled, but not 100% tested.
- 2 This is the variation in the total delay through buffers, transmission lines, differential receivers etc., caused by such things as short term variation in supply voltages and differences in delays for rising and falling edges.

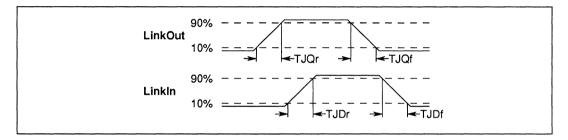


Figure 10.2 IMS T805 link timing

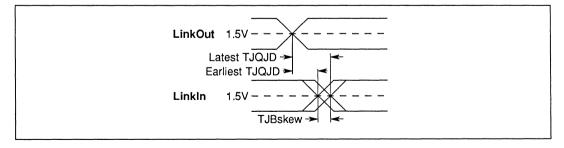


Figure 10.3 IMS T805 buffered link timing

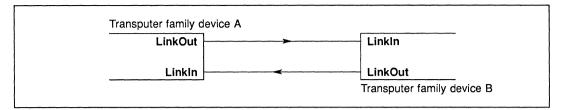


Figure 10.4 IMS T805 Links directly connected

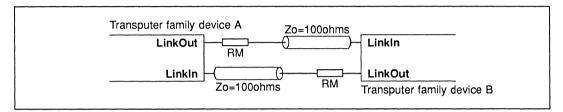


Figure 10.5 IMS T805 Links connected by transmission line

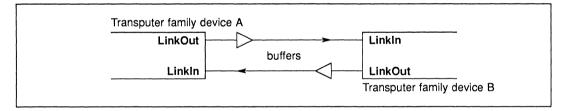


Figure 10.6 IMS T805 Links connected by buffers

## 11 Electrical specifications

#### 11.1 DC electrical characteristics

Table 11.1 Absolute maximum ratings

SYMBOL	PARAMETER	MIN	MAX	UNITS	NOTE
VCC	DC supply voltage	0	7.0	V	1,2,3
VI, VO	Voltage on input and output pins	-0.5	VCC+0.5	V	1,2,3
	Input current		±25	mA	4
OSCT	Output short circuit time (one pin)		1	S	2
TS	Storage temperature	-65	150	°C	2
TA	Ambient temperature under bias	-55	125	°C	2
PDmax	Maximum allowable dissipation		2	W	

#### Notes

- 1 All voltages are with respect to GND.
- 2 This is a stress rating only and functional operation of the device at these or any other conditions beyond those indicated in the operating sections of this specification is not implied. Stresses greater than those listed may cause permanent damage to the device. Exposure to absolute maximum rating conditions for extended periods may affect reliability.
- 3 This device contains circuitry to protect the inputs against damage caused by high static voltages or electrical fields. However, it is advised that normal precautions be taken to avoid application of any voltage higher than the absolute maximum rated voltages to this high impedance circuit. Unused inputs should be tied to an appropriate logic level such as VCC or GND.
- 4 The input current applies to any input or output pin and applies when the voltage on the pin is between GND and VCC.

Table 11.2 Operating conditions

SYMBOL	PARAMETER	MIN	MAX	UNITS	NOTE
VCC	DC supply voltage	4.75	5.25	V	1
VI, VO	Input or output voltage	0	VCC	V	1,2
CL	Load capacitance on any pin		60	pF	
TA	Operating temperature range IMS T805-S	0	70	°C	3
TA	Operating temperature range IMS T805-M	-55	125	°C	3

- 1 All voltages are with respect to GND.
- 2 Excursions beyond the supplies are permitted but not recommended; see DC characteristics.
- 3 Air flow rate 400 linear ft/min transverse air flow.

SYMBOL	PARAME	TER	MIN	MAX	UNITS	NOTE
VIH	High level input voltage		2.0	VCC+0.5	٧	1,2
VIL	Low level input voltage		-0.5	0.8	V	1,2
11	Input current	@ GND <vi<vcc< td=""><td></td><td>±10</td><td><math>\mu</math>A</td><td>1,2</td></vi<vcc<>		±10	$\mu$ A	1,2
VOH	Output high voltage	@ IOH=2mA	VCC-1		٧	1,2
VOL	Output low voltage	@ IOL=4mA		0.4	V	1,2
IOS	Output short circuit curren	t @ GND <vo<vcc< td=""><td>36</td><td>65</td><td>mA</td><td>1,2,3,6</td></vo<vcc<>	36	65	mA	1,2,3,6
			65	100	mA	1,2,4,6
IOZ	Tristate output current	@ GND <vo<vcc< td=""><td></td><td>±10</td><td><math>\mu A</math></td><td>1,2</td></vo<vcc<>		±10	$\mu A$	1,2
PD	Power dissipation			1.2	W	2,5
CIN	Input capacitance	@ f=1MHz		7	pF	6
COZ	Output capacitance	@ f=1MHz		10	pF	6

Table 11.3 DC characteristics

- 1 All voltages are with respect to GND.
- 2 Parameters for IMS T805-S measured at 4.75V<VCC<5.25V and 0°C<TA<70°C. Input clock frequency = 5 MHz.</p>
- 3 Current sourced from non-link outputs.
- 4 Current sourced from link outputs.
- 5 Power dissipation varies with output loading and program execution. Power dissipation for processor operating at 20 MHz.
- 6 This parameter is sampled and not 100% tested.

# 11.2 Equivalent circuits

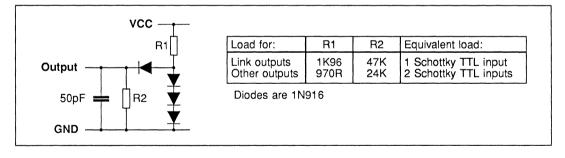


Figure 11.1 Load circuit for AC measurements

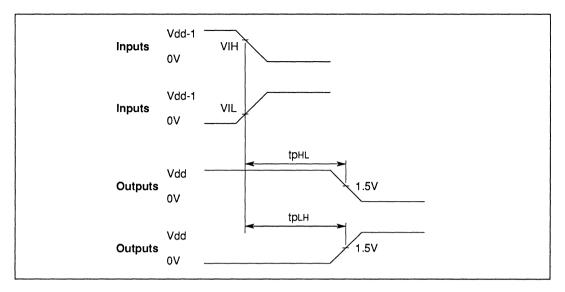


Figure 11.2 AC measurements timing waveforms

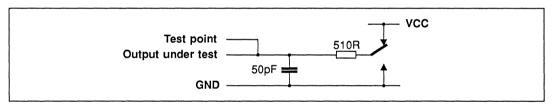


Figure 11.3 Tristate load circuit for AC measurements

### 11.3 AC timing characteristics

Table 11.4 Input, output edges

SYMBOL	PARAMETER	MIN	MAX	UNITS	NOTE
TDr	Input rising edges	2	20	ns	1,2,4
TDf	Input falling edges	2	20	ns	1,2,4
TQr	Output rising edges		25	ns	1,5
TQf	Output falling edges		15	ns	1,5
TS0LaHZ	Address high to tristate	а	<b>a</b> +6	ns	3
TS0LaLZ	Address low to tristate	а	<b>a</b> +6	ns	3

- 1 Non-link pins; see section on links.
- 2 All inputs except Clockin; see section on Clockin.
- 3 a is T2 where T2 can be from one to four periods Tm in length. Address lines include MemnotWrD0, MemnotRfD1, MemAD2-31.
- 4 These parameters are not tested.
- 5 These parameters are sampled, but not 100% tested.

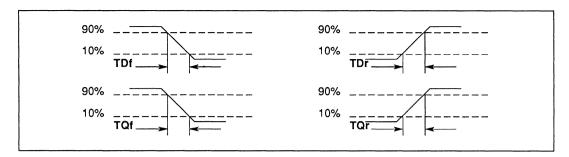


Figure 11.4 IMS T805 input and output edge timing

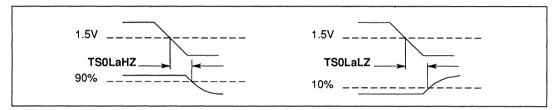


Figure 11.5 IMS T805 tristate timing relative to notMemS0

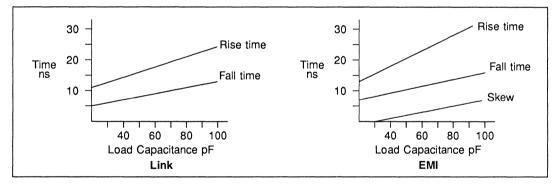


Figure 11.6 Typical rise/fall times

1 Skew is measured between **notMemS0** with a standard load (2 Schottky TTL inputs and 30 pF) and **notMemS0** with a load of 2 Schottky TTL inputs and varying capacitance.

### 11.4 Power rating

Internal power dissipation  $P_{INT}$  of transputer and peripheral chips depends on **VCC**, as shown in figure 11.7.  $P_{INT}$  is substantially independent of temperature.

Total power dissipation  $P_D$  of the chip is

$$P_D = P_{INT} + P_{IO}$$

where  $P_{IO}$  is the power dissipation in the input and output pins; this is application dependent.

Internal working temperature  $T_J$  of the chip is

$$T_J = T_A + \theta J_A * P_D$$

where  $T_A$  is the external ambient temperature in °C and  $\theta J_A$  is the junction-to-ambient thermal resistance in °C/W.  $\theta J_A$  for each package is given in the Packaging Specifications section.

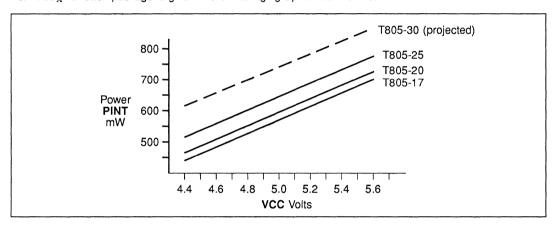


Figure 11.7 IMS T805 internal power dissipation vs VCC

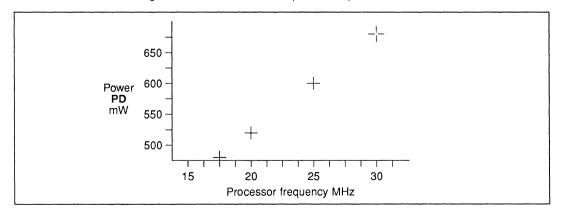


Figure 11.8 IMS T805 typical power dissipation with processor speed

### 12 Performance

The performance of the transputer is measured in terms of the number of bytes required for the program, and the number of (internal) processor cycles required to execute the program. The figures here relate to OCCam programs. For the same function, other languages should achieve approximately the same performance as occam.

With transputers incorporating an FPU, this type of performance calculation is straight forward when considering only integer data types. However, when floating point calculations using the **REAL32** and **REAL64** data types are present in the program, complications arise due to the concurrency inherent in the transputer's design whereby integer calculations can be overlapped with floating point calculations. A more comprehensive guide to the impact of this concurrency on transputer performance can be found in the *Transputer Instruction Set - A Compiler Writers' Guide*.

### 12.1 Performance overview

These figures are averages obtained from detailed simulation, and should be used only as an initial guide; they assume operands are of type **INT**. The abbreviations in table 12.1 are used to represent the quantities indicated. In the replicator section of the table, figures in braces {} are not necessary if the number of replications is a compile time constant. To estimate performance, add together the time for the variable references and the time for the operation.

Table 12.1 Key to performance table

np	number of component processes
ne	number of processes earlier in queue
r	1 if INT parameter or array parameter, 0 if not
ts	number of table entries (table size)
w	width of constant in nibbles
р	number of places to shift
Eg	expression used in a guard
Et	timer expression used in a guard
Tb	most significant bit set of multiplier ((-1) if the multiplier is 0)
Tbp	most significant bit set in a positive multiplier when counting from zero ((-1) if the multiplier is 0)
Tbc	most significant bit set in the two's complement of a negative multiplier
nsp	Number of scalar parameters in a procedure
nap	Number of array parameters in a procedure

Table 12.2 Performance

	Size (bytes)	Time (cycles)
Names		
variables		
in expression	1.1+r	2.1+2( <b>r</b> )
assigned to or input to	1.1+r	1.1+(r)
in PROC or FUNCTION call,	1	``
corresponding to an INT parameter	1.1+r	1.1+(r)
channels	1.1	2.1
Array Variables (for single dimension arrays)		
constant subscript	0	l o
variable subscript	5.3	7.3
expression subscript	5.3	7.3
	0.0	7.0
Declarations	0.1	
CHAN OF protocol	3.1	3.1
[size]CHAN OF protocol	9.4	2.2 + 20.2*size
PROC	body+2	0
Primitives	ļ	
assignment	0	0
input	4	26.5
output	1	26
STOP	2	25
SKIP	0	0
Arithmetic operators		
+ - '	1	1
*	2	39
/	2	40
REM	2	38
<b>&gt;&gt; &lt;&lt;</b>	2	3+ <b>p</b>
Modulo Arithmetic operators		-
PLUS	2	2
MINUS	1 1	1
TIMES (fast multiply, positive operand)	1	4+Tbp
TIMES (fast multiply, negative operand)	1 1	5+Tbc
, , , , , , , , , , , , , , , , , , , ,	·	51126
Boolean operators OR	4	
AND NOT	1 1	8 2
		2
Comparison operators		
= constant	0	1
= variable	2	3
<> constant	1	3
<> variable	3	5
> <	1	2
>= <=	2	4
Bit operators		
/\ \/ <b>×</b> ~	2	2
Expressions		
constant in expression	w	w
check if error	4	6

12 Performance 115

Table 12.3 Performance

	Size (bytes)	Time (cycles)
Timers		
timer input	2	3
timer AFTER		
if past time	2	4
with empty timer queue	2 2 2	31
non-empty timer queue	2	38+ <b>ne</b> ∗9
ALT (timer)		
with empty timer queue	6	52
non-empty timer queue	6	59+ <b>ne</b> *9
timer alt guard	8+2Eg+2Et	34+2 <b>Eg</b> +2 <b>Et</b>
Constructs		
SEQ	0	0
IF	1.3	1.4
if guard	3	4.3
ALT (non timer)	6	26
alt channel guard	10.2+2Eg	20+2 <b>Eg</b>
skip alt guard	8+2 <b>E</b> g	10+2Eg
PAR	11.5+( <b>np</b> -1)*7.5	19.5+( <b>np</b> -1)*30.5
WHILE	4	12
Procedure or function call		
	3.5+(nsp-2)*1.1	16.5+(nsp-2)*1.1
	+nap*2.3	+nap∗2.3
Replicators		
replicated SEQ	7.3{+5.1}	(-3.8)+15.1*count{+7.1}
replicated IF	12.3{+5.1}	(-2.6)+19.4*count{+7.1}
replicated ALT	24.8{+10.2}	25.4+33.4*count{+14.2}
replicated timer ALT	24.8{+10.2}	62.4+33.4*count(+14.2)
replicated PAR	39.1{+5.1}	(-6.4)+70.9*count{+7.1}

### 12.2 Fast multiply, TIMES

The IMS T805 has a fast integer multiplication instruction *product*. For a positive multiplier its execution time is 4+**Tbp** cycles, and for a negative multiplier 5+**Tbc** cycles (table 12.1). The time taken for a multiplication by zero is 3 cycles.

Implementations of high level languages on the transputer may take advantage of this instruction. For example, the OCCAM modulo arithmetic operator **TIMES** is implemented by the instruction and the right-hand operand is treated as the multiplier. The fast multiplication instruction is also used in high level language implementations for the multiplication implicit in multi-dimensional array access.

### 12.3 Arithmetic

A set of functions are provided within the development system to support the efficient implementation of multiple length integer arithmetic. In the IMS T805, floating point arithmetic is taken care of by the FPU. In table 12.4 n gives the number of places shifted and all arguments and results are assumed to be local. Full details of these functions are provided in the OCCAM reference manual, supplied as part of the development system and available as a separate publication.

When calculating the execution time of the predefined maths functions, no time needs to be added for calling overhead. These functions are compiled directly into special purpose instructions which are designed to support the efficient implementation of multiple length integer arithmetic and floating point arithmetic.

Function		Cycles	+ cycles for parameter access †
LONGADD		2	7
LONGSUM		3	8
LONGSUB		2	7
LONGDIFF		3	8
LONGPROD		34	8
LONGDIV		36	8
SHIFTRIGHT	(n<32)	4+n	8
	( <b>n</b> >=32)	n-27	
SHIFTLEFT	(n<32)	4+n	8
	( <b>n</b> >=32)	n-27	
NORMALISE	( <b>n</b> <32)	n+6	7
	(n>=32)	n-25	
	( <b>n=</b> 64)	4	
ASHIFTRIGHT		SHIFTRIGHT+2	5
ASHIFTLEFT		SHIFTLEFT+4	5
ROTATERIGHT		SHIFTRIGHT	7
ROTATELEFT		SHIFTLEFT	7
FRACMUL		LONGPROD+4	5

Table 12.4 Arithmetic performance

<sup>†</sup> Assuming local variables.

12 Performance 117

### 12.4 Floating point operations

All references to **REAL32** or **REAL64** operands within programs compiled for the IMS T805 normally produce the following performance figures.

	Size (bytes)	REAL32 Time (cycles)	REAL64 Time (cycles)
Names			
variables			
in expression	3.1	3 3	5 5
assigned to or input to	3.1	3	5
in <b>PROC</b> or <b>FUNCTION</b> call,			
corresponding to a REAL			
parameter	1.1+ <b>r</b>	1.1+r	1.1+r
Arithmetic operators			
+ -	2	7	7
*	2 2 2 11	11	20
/	2	17	32
REM	11	19	34
Comparison operators			
= .	2	4	4
<>	2 3 2 3	6	6
> <	2	6 5 7	6 5 7
>= <=	3	7	7
Conversions			
REAL32 to -	2		3
REAL64 to -	2	6	
To INT32 from -	2 2 5 18 3	9	9
To INT64 from -	18	32	32
INT32 to -		7	7
INT64 to -	14	24	22

Table 12.5 Floating point performance

## 12.4.1 Floating point functions

These functions are provided by the development system. They are compiled directly into special purpose instructions designed to support the efficient implementation of some of the common mathematical functions of other languages. The functions provide ABS and SQRT for both REAL32 and REAL64 operand types.

		+ cycles for parameter access †					
Function	Cycles	REAL32	REAL 64				
ABS	2	8					
SQRT	118	8					
DABS	2		12				
DSQRT	244		12				

Table 12.6 IMS T805 floating point arithmetic performance

<sup>†</sup> Assuming local variables.

### 12.4.2 Special purpose functions and procedures

The functions and procedures given in tables 12.8 and 12.9 are provided by the development system to give access to the special instructions available on the IMS T805. Table 12.7 shows the key to the table.

Table 12.7 Key to special performance table

Tb	most significant bit set in the word counting from zero
n	number of words per row (consecutive memory locations)
r	number of rows in the two dimensional move
nr	number of bits to reverse

Table 12.8 Special purpose functions performance

Function	Cycles	+ cycles for parameter access †		
BITCOUNT	2+ <b>Tb</b>	2		
CRCBYTE	11	8		
CRCWORD	35	8		
BITREVNBIT	5+nr	4		
BITREVWORD	36	2		

† Assuming local variables.

Table 12.9 Special purpose procedures performance

Procedure	Cycles	+ cycles for parameter access †			
MOVE2D	8+(2n+23)*r	8			
DRAW2D	8+(2n+23)*r	8			
CLIP2D	8+(2n+23)*r	8			

† Assuming local variables.

### 12.5 Effect of external memory

Extra processor cycles may be needed when program and/or data are held in external memory, depending both on the operation being performed, and on the speed of the external memory. After a processor cycle which initiates a write to memory, the processor continues execution at full speed until at least the next memory access.

Whilst a reasonable estimate may be made of the effect of external memory, the actual performance will depend upon the exact nature of the given sequence of operations.

External memory is characterized by the number of extra processor cycles per external memory cycle, denoted as **e**. For the IMS T805, with the fastest external memory the value of **e** is 2; a typical value for a large external memory is 5.

If a program is stored in external memory, and **e** has the value 2 or 3, then no extra cycles need be estimated for linear code sequences. For larger values of **e**, the number of extra cycles required for linear code sequences may be estimated at (**e**-3)/4. A transfer of control may be estimated as requiring **e**+3 cycles.

These estimates may be refined for various constructs. In table 12.10 n denotes the number of components in a construct. In the case of IF, the n'th conditional is the first to evaluate to TRUE, and the costs include the

12 Performance 119

costs of the conditionals tested. The number of bytes in an array assignment or communication is denoted by  ${\bf b}$ .

	IMS T805			
	Program off chip	Data off chip		
Boolean expressions	<b>e</b> -2	0		
IF	3 <b>en</b> -8	en		
Replicated IF	(6 <b>e</b> -4) <b>n</b> +7	(5 <b>e</b> -2) <b>n</b> +8		
Replicated SEQ	(3 <b>e</b> -3) <b>n</b> +2	(4e-2)n		
PAR	(3 <b>e</b> -1) <b>n</b> +8	3 <b>en</b> +4		
Replicated PAR	(10 <b>e</b> -8) <b>n</b> +8	16 <b>en</b> -12		
ALT	(2 <b>e</b> -4) <b>n</b> +6 <b>e</b>	(2e-2)n+10e-8		
Array assignment and communication in one transputer	0	max (2e, e(b/2))		

Table 12.10 External memory performance

The following simulation results illustrate the effect of storing program and/or data in external memory. The results are normalized to 1 for both program and data on chip. The first program (Sieve of Erastosthenes) is an extreme case as it is dominated by small, data access intensive loops; it contains no concurrency, communication, or even multiplication or division. The second program is the pipeline algorithm for Newton Raphson square root computation.

	Program	e=2	e=3	e=4	e=5	On chip
Program off chip	1	1.3	1.5	1.7	1.9	1
	2	1.1	1.2	1.2	1.3	1
Data off chip	1	1.5	1.8	2.1	2.3	1
·	2	1.2	1.4	1.6	1.7	1
Program and data off chip	1	1.8	2.2	2.7	3.2	1
	2	1.3	1.6	1.8	2.0	1

Table 12.11 IMS T805 external memory performance

### 12.6 Interrupt latency

If the process is a high priority one and no other high priority process is running, the latency is as described in table 12.12. The timings given are in full processor cycles **TPCLPCL**; the number of **Tm** states is also given where relevant. Maximum latency assumes all memory accesses are internal ones.

	Тур	ical	Maximum		
	TPCLPCL	Tm	TPCLPCL	Tm	
IMS T805 with FPU in use	19	38	78	156	
IMS T805 with FPU not in use	19	38	58	116	

Table 12.12 Interrupt latency

# 13 Package specifications

# 13.1 84 pin grid array package

	1	2	3	4	5	6	7	8	9	10
A	Refresh Pending	Link Special	Proc Clock Out	Link 123 Special	Link In0	Link Out1	Link In2	Event Ack	GND	Mem Wait
В	Proc Speed Select0	Clockin	Event Waiting	Link0 Special	Link Out0	Link Out2	Link Out3	Event Req	Mem Req	not Mem WrB3
С	GND	vcc	Cap Minus	Cap Plus	vcc	Link In1	Link In3	Mem Config	Mem Granted	not Mem WrB1
D	Error	Proc Speed Select2	Errorin	Inde	Index				not Mem WrB2	not Mem WrB0
E	Disable Int RAM	Boot From ROM	Reset			T805		not Mem Rd	not Mem S0	vcc
F	Proc Speed Select1	Analyse	Mem AD31			rid array view		not Mem S3	not Mem S2	not Mem S4
G	Mem AD30	GND	Mem AD27					Mem not WrD0	GND	not Mem S1
Н	Mem AD29	Mem AD25	Mem AD23	vcc	Mem AD16	Mem AD12	Mem AD8	Mem AD4	Mem AD3	Mem not RfD1
J	Mem AD28	Mem AD24	Mem AD22	Mem AD19	Mem AD17	Mem AD13	GND	Mem AD6	Mem AD5	Mem AD2
ĸ	Mem AD26	Mem AD21	Mem AD20	Mem AD18	Mem AD15	Mem AD14	Mem AD11	Mem AD10	Mem AD9	Mem AD7

Figure 13.1 IMS T805 84 pin grid array package pinout

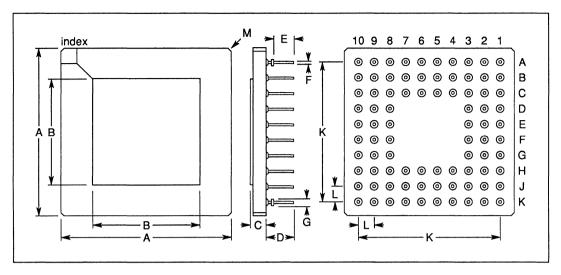


Figure 13.2 84 pin grid array package dimensions

Table 13.1 84 pin grid array package dimensions

	Millim	etres	Inches		
DIM	NOM	TOL	NOM	TOL	Notes
Α	26.924	±0.254	1.060	±0.010	
В	17.019	±0.127	0.670	±0.005	
С	2.456	±0.278	0.097	±0.011	
D	4.572	±0.127	0.180	±0.005	
Ε	3.302	±0.127	0.130	±0.005	
F	0.457	±0.025	0.018	±0.002	Pin diameter
G	1.143	±0.127	0.045	±0.005	Flange diameter
K	22.860	±0.127	0.900	±0.005	
L	2.540	±0.127	0.100	±0.005	
М	0.508		0.020		Chamfer

Package weight is approximately 7.2 grams

Table 13.2 84 pin grid array package junction to ambient thermal resistance

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTE
$\theta$ JA	At 400 linear ft/min transverse air flow			35	°C/W	

#### 13.2 84 pin PLCC J-bend package

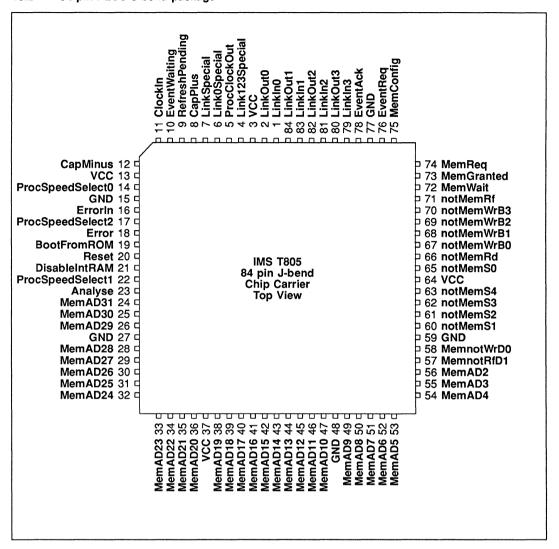


Figure 13.3 IMS T805 84 pin PLCC J-bend package pinout

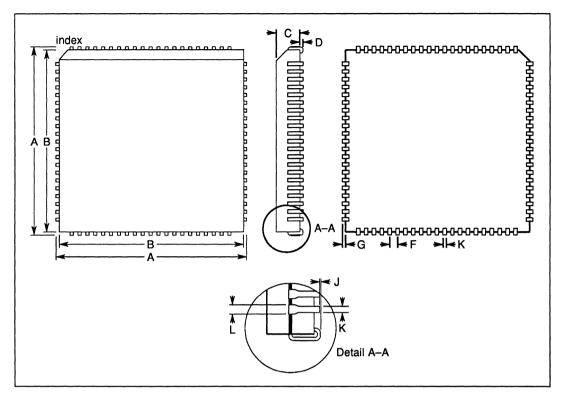


Figure 13.4 84 pin PLCC J-bend package dimensions

Table 13.3 84 pin PLCC J-bend package dimensions

	Millim	etres	Inches		
DIM	МОМ	TOL	МОМ	TOL	Notes
Α	30.226	±0.127	1.190	±0.005	
В	29.312	±0.127	1.154	±0.005	
C	3.810	±0.127	0.150	±0.005	
D	0.508	±0.127	0.020	±0.005	
F	1.270	±0.127	0.050	±0.005	
G	0.457	±0.127	0.018	±0.005	
J	0.000	±0.051	0.000	±0.002	
K	0.457	±0.127	0.018	±0.005	
L	0.762	±0.127	0.030	±0.005	

Package weight is approximately 7.0 grams

Table 13.4 84 pin PLCC J-bend package junction to ambient thermal resistance

SYMBOL	PARAMETER		NOM	MAX	UNITS	NOTE
$\theta$ JA	At 400 linear ft/min transverse air flow		35		°C/W	

### 13.3 84 lead quad cerpack package

The leads are unformed to allow the user to form them to specific requirements.

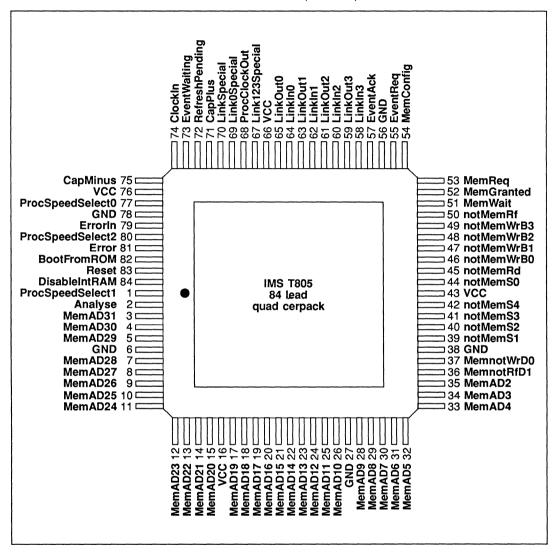


Figure 13.5 IMS T805 84 lead quad cerpack package pinout

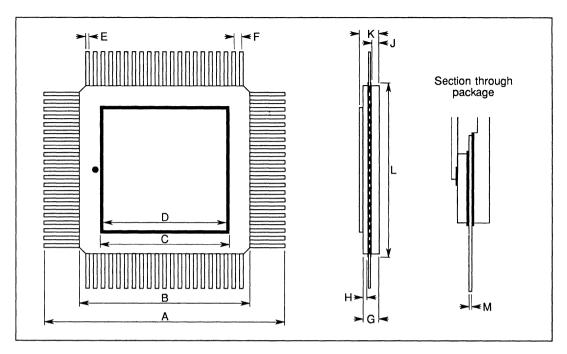


Figure 13.6 84 lead quad cerpack package dimensions

	Millimetres		Inc	ches	
DIM	NOM	TOL	МОМ	TOL	Notes
Α	38.100	±0.508	1.500	±0.020	
В	26.924	±0.305	1.060	±0.012	
С	20.574	±0.203	0.810	±0.008	
D	19.558	±0.254	0.770	±0.010	
E	0.508		0.020		
F	1.270	±0.051	0.050	±0.002	
G	2.489	±0.305	0.098	±0.012	
н	0.635	±0.076	0.025	±0.003	
J	1.143	±0.102	0.045	±0.004	
K	3.099		0.122		Max.
L	27.940		1.100		Мах.
М	0.178	±0.025	0.007	±0.001	

Table 13.5 84 lead quad cerpack package dimensions

## 14 Ordering

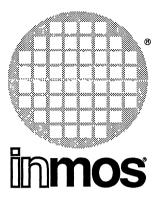
This section indicates the designation of speed and package selections for the various devices. Speed of **ClockIn** is 5 MHz for all parts. Transputer processor cycle time is nominal; it can be calculated more exactly using the phase lock loop factor **PLLx**, as detailed in the external memory section.

For availability contact local INMOS sales office or authorised distributor.

Table 14.1 IMS T805 ordering details

INMOS designation	Processor clock speed	Processor cycle time	PLLx	Package
IMS T805-G17S	17.5 MHz	57 ns	3.5	Ceramic Pin Grid
IMS T805-G20S	20.0 MHz	50 ns	4.0	Ceramic Pin Grid
IMS T805-G25S	25.0 MHz	40 ns	5.0	Ceramic Pin Grid
IMS T805-G30S	30.0 MHz	33 ns	6.0	Ceramic Pin Grid
IMS T805-J17S IMS T805-J20S	17.5 MHz 20.0 MHz	57 ns 50 ns	3.5 4.0	Plastic PLCC J-Bend Plastic PLCC J-Bend
IMS T805-G17M	17.5 MHz	57 ns	3.5	Ceramic Pin Grid MIL Spec
IMS T805-G20M	20.0 MHz	50 ns	4.0	Ceramic Pin Grid MIL Spec
IMS T805-Q17M IMS T805-Q20M	17.5 MHz 20.0 MHz	57 ns 50 ns	3.5 4.0	Quad Cerpack MIL Spec Quad Cerpack MIL Spec

The timing parameters in this datasheet are based on 17 MHz and 20 MHz parts. Data for higher speeds is based on tests on a limited number of samples and may change when full characterisation is completed.



# IMS T801 transputer

# Preliminary Data

### **FEATURES**

32 bit architecture

33 ns internal cycle time

30 MIPS (peak) instruction rate

4.3 Mflops (peak) instruction rate

Debugging support

64 bit on-chip floating point unit which conforms to IEEE 754

4 Kbytes on-chip static RAM

120 Mbytes/sec sustained data rate to internal memory

4 Gbytes directly addressable external memory

60 Mbytes/sec sustained data rate to external memory

630 ns response to interrupts

Four INMOS serial links 10/20 Mbits/sec

Bi-directional data rate of 2.4 Mbytes/sec per link High performance graphics support with block move

instructions

Boot from ROM or communication links

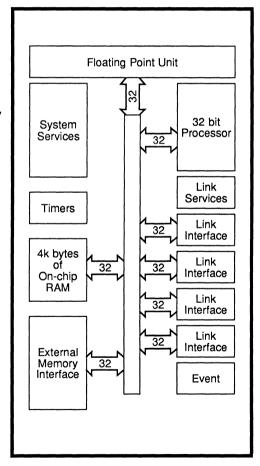
Single 5 MHz clock input

Single +5V ±5% power supply

MIL-STD-883C processing will be available

## **APPLICATIONS**

Scientific and mathematical applications
High speed multi processor systems
High performance graphics processing
Supercomputers
Workstations and workstation clusters
Digital signal processing
Accelerator processors
Distributed databases
System simulation
Telecommunications
Robotics
Fault tolerant systems
Image processing
Pattern recognition
Artificial intelligence



42 1441 00 May 1989

#### 1 Introduction

The IMS T801 transputer is a 32 bit CMOS microcomputer with a 64 bit floating point unit and graphics support. It has 4 Kbytes on-chip RAM for high speed processing, a 32 bit non-multiplexed external memory interface and four standard INMOS communication links. The instruction set achieves efficient implementation of high level languages and provides direct support for the OCCAM model of concurrency when using either a single transputer or a network. Procedure calls, process switching and typical interrupt latency are sub-microsecond.

For convenience of description, the IMS T801 operation is split into the basic blocks shown in figure 1.1.

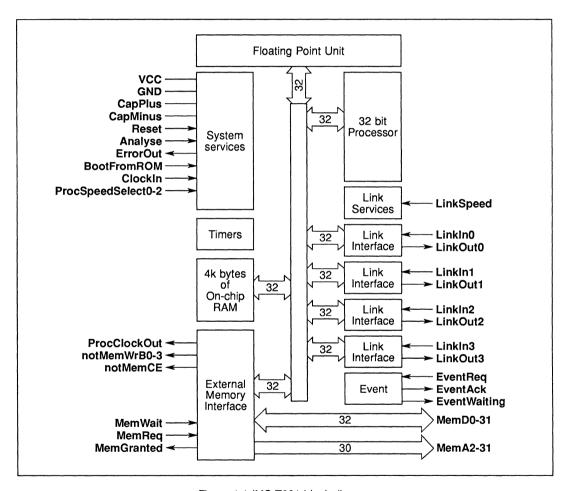


Figure 1.1 IMS T801 block diagram

The processor speed of a device can be pin-selected in stages from 17.5 MHz up to the maximum allowed for the part. A device running at 30 MHz achieves an instruction throughput of 30 MIPS peak and 15 MIPS sustained. The extended temperature version of the device complies with MIL-STD-883C.

The IMS T801 provides high performance arithmetic and floating point operations. The 64 bit floating point unit provides single and double length operation to the ANSI-IEEE 754-1985 standard for floating point arithmetic. It is able to perform floating point operations concurrently with the processor, sustaining a rate of 2.2 Mflops at a processor speed of 20 MHz and 3.3 Mflops at 30 MHz.

1 Introduction 129

High performance graphics support is provided by microcoded block move instructions which operate at the speed of memory. The two-dimensional block move instructions provide for contiguous block moves as well as block copying of either non-zero bytes of data only or zero bytes only. Block move instructions can be used to provide graphics operations such as text manipulation, windowing, panning, scrolling and screen updating.

Cyclic redundancy checking (CRC) instructions are available for use on arbitrary length serial data streams, to provide error detection where data integrity is critical. Another feature of the IMS T801, useful for pattern recognition, is the facility to count bits set in a word.

The IMS T801 can directly access a linear address space of 4 Gbytes. The 32 bit wide memory interface uses non-multiplexed data and address lines and provides a data rate of up to 4 bytes every 66 nanoseconds (60 Mbytes/sec) for a 30 MHz device.

System Services include processor reset and bootstrap control, together with facilities for error analysis.

The standard INMOS communication links allow networks of transputer family products to be constructed by direct point to point connections with no external logic. The IMS T801 links support the standard operating speed of 10 Mbits/sec, but also operate at 20 Mbits/sec. Each link can transfer data bi-directionally at up to 2.35 Mbytes/sec.

The transputer is designed to implement the OCCAM language, detailed in the OCCAM Reference Manual, but also efficiently supports other languages such as C, Pascal and Fortran. Access to the transputer at machine level is seldom required, but if necessary refer to the *Transputer Instruction Set - A Compiler Writers' Guide*, where the IMS T800 instruction set is applicable.

This data sheet supplies hardware implementation and characterisation details for the IMS T801. It is intended to be read in conjunction with the Transputer Architecture chapter, which details the architecture of the transputer and gives an overview of OCCAM.

The IMS T801 instruction set contains a number of instructions to facilitate the implementation of breakpoints. For further information concerning breakpointing, refer to *Support for debugging/breakpointing in transputers* (technical note 61).

Figure 1.2 shows the internal datapaths for the IMS T801.

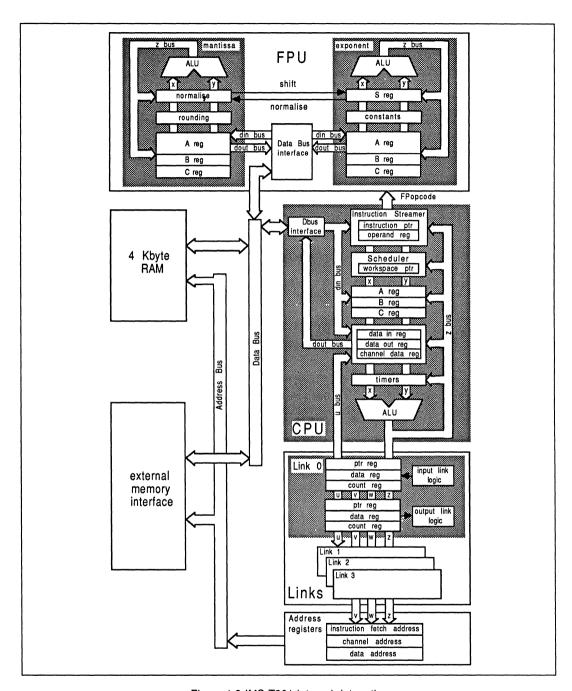


Figure 1.2 IMS T801 internal datapaths

## 2 Pin designations

Table 2.1 IMS T801 system services

Pin	In/Out	Function
VCC, GND		Power supply and return
CapPlus, CapMinus		External capacitor for internal clock power supply
Clockin	in	Input clock
ProcSpeedSelect0-2	in	Processor speed selectors
Reset	in	System reset
ErrorOut	out	Error indicator
Analyse	in	Error analysis
BootFromRom	in	Boot from external ROM or from link

Table 2.2 IMS T801 external memory interface

Pin	In/Out	Function
ProcClockOut	out	Processor clock
MemA2-31	out	Thirty address lines
Data0-31	in/out	Thirty-two non-multiplexed data lines
notMemWrB0-3	out	Four byte-addressing write strobes
notMemCE	out	Chip enable
MemWait	in	Memory cycle extender
MemReq	in	Direct memory access request
MemGranted	out	Direct memory access granted

Table 2.3 IMS T801 event

Pin	In/Out	Function
EventReq	in	Event request
EventAck	out	Event request acknowledge
EventWaiting	out	Event input requested by software

Table 2.4 IMS T801 link

Pin	In/Out	Function
Linkin0-3	in	Four serial data input channels
LinkOut0-3	out	Four serial data output channels
LinkSpeed	in	Select speed for Links 0-3 to 10 or 20 Mbits/sec

Signal names are prefixed by **not** if they are active low, otherwise they are active high. Pinout details for various packages are given on page 186.

#### 3 Processor

The 32 bit processor contains instruction processing logic, instruction and work pointers, and an operand register. It directly accesses the high speed 4 Kbyte on-chip memory, which can store data or programs. Where larger amounts of memory or programs in ROM are required, the processor has access to 4 Gbytes of memory via the External Memory Interface (EMI).

### 3.1 Registers

The design of the transputer processor exploits the availability of fast on-chip memory by having only a small number of registers; six registers are used in the execution of a sequential process. The small number of registers, together with the simplicity of the instruction set, enables the processor to have relatively simple (and fast) data-paths and control logic. The six registers are:

The workspace pointer which points to an area of store where local variables are kept.

The instruction pointer which points to the next instruction to be executed.

The operand register which is used in the formation of instruction operands.

The A, B and C registers which form an evaluation stack.

A, B and C are sources and destinations for most arithmetic and logical operations. Loading a value into the stack pushes B into C, and A into B, before loading A. Storing a value from A, pops B into A and C into B.

Expressions are evaluated on the evaluation stack, and instructions refer to the stack implicitly. For example, the *add* instruction adds the top two values in the stack and places the result on the top of the stack. The use of a stack removes the need for instructions to respecify the location of their operands. Statistics gathered from a large number of programs show that three registers provide an effective balance between code compactness and implementation complexity.

No hardware mechanism is provided to detect that more than three values have been loaded onto the stack. It is easy for the compiler to ensure that this never happens.

Any location in memory can be accessed relative to the workpointer register, enabling the workspace to be of any size.

Further register details are given in Transputer Instruction Set - A Compiler Writers' Guide.

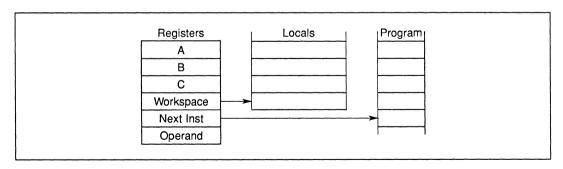


Figure 3.1 Registers

Processor 133

#### 3.2 Instructions

The instruction set has been designed for simple and efficient compilation of high-level languages. All instructions have the same format, designed to give a compact representation of the operations occurring most frequently in programs.

Each instruction consists of a single byte divided into two 4-bit parts. The four most significant bits of the byte are a function code and the four least significant bits are a data value.

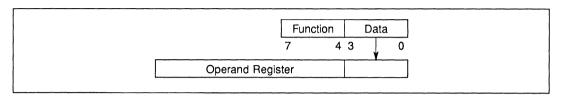


Figure 3.2 Instruction format

#### 3.2.1 Direct functions

The representation provides for sixteen functions, each with a data value ranging from 0 to 15. Ten of these, shown in table 3.1, are used to encode the most important functions.

load constant	add constant	
load local	store local	load local pointer
load non-local	store non-local	
jump	conditional jump	call

Table 3.1 Direct functions

The most common operations in a program are the loading of small literal values and the loading and storing of one of a small number of variables. The *load constant* instruction enables values between 0 and 15 to be loaded with a single byte instruction. The *load local* and *store local* instructions access locations in memory relative to the workspace pointer. The first 16 locations can be accessed using a single byte instruction.

The *load non-local* and *store non-local* instructions behave similarly, except that they access locations in memory relative to the *A* register. Compact sequences of these instructions allow efficient access to data structures, and provide for simple implementations of the static links or displays used in the implementation of high level programming languages such as OCCAM, C, Fortran, Pascal or ADA.

#### 3.2.2 Prefix functions

Two more function codes allow the operand of any instruction to be extended in length; *prefix* and *negative prefix*.

All instructions are executed by loading the four data bits into the least significant four bits of the operand register, which is then used as the instruction's operand. All instructions except the prefix instructions end by clearing the operand register, ready for the next instruction.

The *prefix* instruction loads its four data bits into the operand register and then shifts the operand register up four places. The *negative prefix* instruction is similar, except that it complements the operand register before shifting it up. Consequently operands can be extended to any length up to the length of the operand register by a sequence of prefix instructions. In particular, operands in the range -256 to 255 can be represented using one prefix instruction.

The use of prefix instructions has certain beneficial consequences. Firstly, they are decoded and executed in the same way as every other instruction, which simplifies and speeds instruction decoding. Secondly, they simplify language compilation by providing a completely uniform way of allowing any instruction to take an operand of any size. Thirdly, they allow operands to be represented in a form independent of the processor wordlength.

#### 3.2.3 Indirect functions

The remaining function code, *operate*, causes its operand to be interpreted as an operation on the values held in the evaluation stack. This allows up to 16 such operations to be encoded in a single byte instruction. However, the prefix instructions can be used to extend the operand of an *operate* instruction just like any other. The instruction representation therefore provides for an indefinite number of operations.

Encoding of the indirect functions is chosen so that the most frequently occurring operations are represented without the use of a prefix instruction. These include arithmetic, logical and comparison operations such as add, exclusive or and greater than. Less frequently occurring operations have encodings which require a single prefix operation.

### 3.2.4 Expression evaluation

Evaluation of expressions sometimes requires use of temporary variables in the workspace, but the number of these can be minimised by careful choice of the evaluation order.

Program	Mnem	onic
x := 0	ldc	0
	stl	X
x := #24	pfix	2
	Idc	2 4
	stl	X
x := y + z	ldl	у
	ldl	Z
	add	
	stl	X

Table 3.2 Expression evaluation

### 3.2.5 Efficiency of encoding

Measurements show that about 70% of executed instructions are encoded in a single byte; that is, without the use of prefix instructions. Many of these instructions, such as *load constant* and *add* require just one processor cycle.

The instruction representation gives a more compact representation of high level language programs than more conventional instruction sets. Since a program requires less store to represent it, less of the memory bandwidth is taken up with fetching instructions. Furthermore, as memory is word accessed the processor will receive four instructions for every fetch.

Short instructions also improve the effectiveness of instruction pre-fetch, which in turn improves processor performance. There is an extra word of pre-fetch buffer, so the processor rarely has to wait for an instruction fetch before proceeding. Since the buffer is short, there is little time penalty when a jump instruction causes the buffer contents to be discarded.

3 Processor 135

#### 3.3 Processes and concurrency

A process starts, performs a number of actions, and then either stops without completing or terminates complete. Typically, a process is a sequence of instructions. A transputer can run several processes in parallel (concurrently). Processes may be assigned either high or low priority, and there may be any number of each (page 136).

The processor has a microcoded scheduler which enables any number of concurrent processes to be executed together, sharing the processor time. This removes the need for a software kernel.

At any time, a concurrent process may be

Active - Being executed.

On a list waiting to be executed.

Inactive - Ready to input.

Ready to output.

- Waiting until a specified time.

The scheduler operates in such a way that inactive processes do not consume any processor time. It allocates a portion of the processor's time to each process in turn. Active processes waiting to be executed are held in two linked lists of process workspaces, one of high priority processes and one of low priority processes (page 136). Each list is implemented using two registers, one of which points to the first process in the list, the other to the last. In the Linked Process List figure 3.3, process S is executing and P, Q and R are active, awaiting execution. Only the low priority process queue registers are shown; the high priority process ones perform in a similar manner.

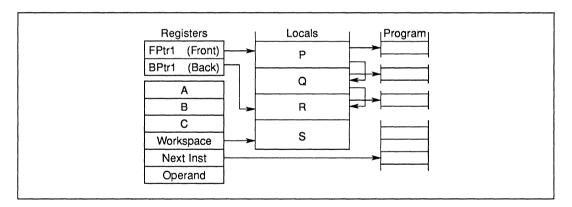


Figure 3.3 Linked process list

Table 3.3 Priority queue control registers

Function	High Priority	Low Priority
Pointer to front of active process list	Fptr0 Bptr0	Fptr1 Bptr1
Pointer to back of active process list	Bptr0	

Each process runs until it has completed its action, but is descheduled whilst waiting for communication from another process or transputer, or for a time delay to complete. In order for several processes to operate in parallel, a low priority process is only permitted to run for a maximum of two time slices before it is forcibly descheduled at the next descheduling point (page 140). The time slice period is 5120 cycles of the external 5 MHz clock, giving ticks approximately 1 ms apart.

A process can only be descheduled on certain instructions, known as descheduling points (page 140). As a result, an expression evaluation can be guaranteed to execute without the process being timesliced part way through.

Whenever a process is unable to proceed, its instruction pointer is saved in the process workspace and the next process taken from the list. Process scheduling pointers are updated by instructions which cause scheduling operations, and should not be altered directly. Actual process switch times are less than 1  $\mu$ s, as little state needs to be saved and it is not necessary to save the evaluation stack on rescheduling.

The processor provides a number of special operations to support the process model, including *start process* and *end process*. When a main process executes a parallel construct, *start process* instructions are used to create the necessary additional concurrent processes. A *start process* instruction creates a new process by adding a new workspace to the end of the scheduling list, enabling the new concurrent process to be executed together with the ones already being executed. When a process is made active it is always added to the end of the list, and thus cannot pre-empt processes already on the same list.

The correct termination of a parallel construct is assured by use of the *end process* instruction. This uses a workspace location as a counter of the parallel construct components which have still to terminate. The counter is initialised to the number of components before the processes are *started*. Each component ends with an *end process* instruction which decrements and tests the counter. For all but the last component, the counter is non zero and the component is descheduled. For the last component, the counter is zero and the main process continues.

#### 3.4 Priority

The IMS T801 supports two levels of priority. Priority 1 (low priority) processes are executed whenever there are no active priority 0 (high priority) processes.

High priority processes are expected to execute for a short time. If one or more high priority processes are able to proceed, then one is selected and runs until it has to wait for a communication, a timer input, or until it completes processing.

If no process at high priority is able to proceed, but one or more processes at low priority are able to proceed, then one is selected.

Low priority processes are periodically timesliced to provide an even distribution of processor time between computationally intensive tasks.

If there are **n** low priority processes, then the maximum latency from the time at which a low priority process becomes active to the time when it starts processing is 2**n**-2 timeslice periods. It is then able to execute for between one and two timeslice periods, less any time taken by high priority processes. This assumes that no process monopolises the transputer's time; i.e. it has a distribution of descheduling points (page 140).

Each timeslice period lasts for 5120 cycles of the external 5 MHz input clock (approximately 1 ms at the standard frequency of 5 MHz).

If a high priority process is waiting for an external channel to become ready, and if no other high priority process is active, then the interrupt latency (from when the channel becomes ready to when the process starts executing) is typically 19 processor cycles, a maximum of 78 cycles (assuming use of on-chip RAM). If the floating point unit is not being used at the time then the maximum interrupt latency is only 58 cycles. To ensure this latency, certain instructions are interruptable.

#### 3.5 Communications

Communication between processes is achieved by means of channels. Process communication is point-to-point, synchronised and unbuffered. As a result, a channel needs no process queue, no message queue and no message buffer.

A channel between two processes executing on the same transputer is implemented by a single word in memory; a channel between processes executing on different transputers is implemented by point-to-point links. The processor provides a number of operations to support message passing, the most important being input message and output message.

The *input message* and *output message* instructions use the address of the channel to determine whether the channel is internal or external. Thus the same instruction sequence can be used for both, allowing a process to be written and compiled without knowledge of where its channels are connected.

The process which first becomes ready must wait until the second one is also ready. A process performs an input or output by loading the evaluation stack with a pointer to a message, the address of a channel, and a count of the number of bytes to be transferred, and then executing an *input message* or *output message* instruction. Data is transferred if the other process is ready. If the channel is not ready or is an external one the process will deschedule.

#### 3.6 Block move

The block move on the transputer moves any number of bytes from any byte boundary in memory, to any other byte boundary, using the smallest possible number of word read, and word or part-word writes.

A block move instruction can be interrupted by a high priority process. On interrupt, block move is completed to a word boundary, independent of start position. When restarting after interrupt, the last word written is written again. This appears as an unnecessary read and write in the simplest case of word aligned block moves, and may cause problems with FIFOs. This problem can be overcome by incrementing the saved destination (BregIntSaveLoc) and source pointer (CregIntSaveLoc) values by BytesPerWord during the high priority process.

#### 3.7 Timers

The transputer has two 32 bit timer clocks which 'tick' periodically. The timers provide accurate process timing, allowing processes to deschedule themselves until a specific time.

One timer is accessible only to high priority processes and is incremented every microsecond, cycling completely in approximately 4295 seconds. The other is accessible only to low priority processes and is incremented every 64 microseconds, giving exactly 15625 ticks in one second. It has a full period of approximately 76 hours.

Table 3.4 Timer registers

Clock0	Current value of high priority (level 0) process clock
Clock1	Current value of low priority (level 1) process clock
TNextReg0	Indicates time of earliest event on high priority (level 0) timer queue
TNextReg1	Indicates time of earliest event on low priority (level 1) timer queue

The current value of the processor clock can be read by executing a *load timer* instruction. A process can arrange to perform a *timer input*, in which case it will become ready to execute after a specified time has been reached. The *timer input* instruction requires a time to be specified. If this time is in the 'past' then the instruction has no effect. If the time is in the 'future' then the process is descheduled. When the specified time is reached the process is scheduled again.

Figure 3.4 shows two processes waiting on the timer queue, one waiting for time 21, the other for time 31.

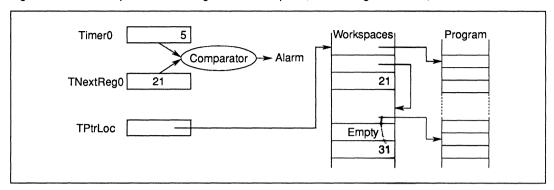


Figure 3.4 Timer registers

## 4 Instruction set summary

The Function Codes table 4.8 gives the basic function code set (page 133). Where the operand is less than 16, a single byte encodes the complete instruction. If the operand is greater than 15, one prefix instruction (*pfix*) is required for each additional four bits of the operand. If the operand is negative the first prefix instruction will be *nfix*.

Mnem	onic		Function code	Memory code
ldc	#3		#4	#43
<i>ldc</i> is coded as	#35			
pfix	#3		#2	#23
Idc	#5		#4	#45
ldc	#987			
is coded as				
pfix	#9		#2	#29
pfix	#8		#2	#28
ldc	#7		#4	#47
ldc	-31	(ldc	#FFFFFE1)	
is coded as		•	,	
nfix	#1		#6	#61
ldc	#1		#4	#41

Table 4.1 prefix coding

Tables 4.9 to 4.28 give details of the operation codes. Where an operation code is less than 16 (e.g. *add*: operation code **05**), the operation can be stored as a single byte comprising the *operate* function code **F** and the operand (**5** in the example). Where an operation code is greater than 15 (e.g. *ladd*: operation code **16**), the *prefix* function code **2** is used to extend the instruction.

Mnem	nonic	Function code	Memory code
add is coded as	(op. code #5)		#F5
opr	add	#F	#F5
ladd is coded as	(op. code #16)		#21F6
pfix	#1	#2	#21
opr	#6	#F	#F6

Table 4.2 operate coding

The load device identity (*Iddevid*) instruction (table 4.20) pushes the device type identity into the A register. Each product is allocated a unique group of numbers for use with the *Iddevid* instruction. The product identity numbers for the IMS T801 are 20 to 29 inclusive.

In the Floating Point Operation Codes tables 4.22 to 4.28, a selector sequence code (page 149) is indicated in the Memory Code column by **s**. The code given in the Operation Code column is the indirection code, the operand for the *Idc* instruction.

The FPU and processor operate concurrently, so the actual throughput of floating point instructions is better than that implied by simply adding up the instruction times. For full details see *Transputer Instruction Set - A Compiler Writers' Guide*.

The Processor Cycles column refers to the number of periods **TPCLPCL** taken by an instruction executing in internal memory. The number of cycles is given for the basic operation only; where the memory code for an instruction is two bytes, the time for the *prefix* function (one cycle) should be added. For a 20 MHz transputer one cycle is 50 ns. Some instruction times vary. Where a letter is included in the cycles column it is interpreted from table 4.3.

Table 4.3 Instruction set interpretation

ldent	Interpretation
b	Bit number of the highest bit set in register A. Bit 0 is the least significant bit.
m	Bit number of the highest bit set in the absolute value of register <i>A</i> . Bit 0 is the least significant bit.
n	Number of places shifted.
w	Number of words in the message. Part words are counted as full words. If the message is not word aligned the number of words is increased to include the part words at either end of the message.
р	Number of words per row.
r	Number of rows.

The **DE** column of the tables indicates the descheduling/error features of an instruction as described in table 4.4.

Table 4.4 Instruction features

Ident	Feature	See page:
D	The instruction is a descheduling point	140
E	The instruction will affect the Error flag	141, 156
F	The instruction will affect the FP_Error flag	149, 141

## 4.1 Descheduling points

The instructions in table 4.5 are the only ones at which a process may be descheduled (page 135). They are also the ones at which the processor will halt if the **Analyse** pin is asserted (page 155).

Table 4.5 Descheduling point instructions

jump loop end end process stop process	input message timer alt wait jump	output message timer input loop end	output byte stop on error end process	output word alt wait stop process	
--	---	---	---	---	--

#### 4.2 Error instructions

The instructions in table 4.6 are the only ones which can affect the *Error* flag (page 156) directly. Note, however, that the floating point unit error flag *FP\_Error* is set by certain floating point instructions (page 141), and that *Error* can be set from this flag by *fpcheckerror*.

Table 4.6 Error setting instructions

add	add constant	subtract	remainder
multiply	fractional multiply	divide	
long add	long subtract	long divide	
set error	testerr	fpcheckerror	check count from 1
check word	check subscript from 0	check single	

## 4.3 Debugging support

Table 4.21 contains a number of instructions to facilitate the implementation of breakpoints. These instructions overload the operation of *j0*. Normally *j0* is a no-op which might cause descheduling. *Setj0break* enables the breakpointing facilities and causes *j0* to act as a breakpointing instruction. When breakpointing is enabled, *j0* swaps the current **lptr** and **Wptr** with an **lptr** and **Wptr** stored above MemStart. The breakpoint instruction does not cause descheduling, and preserves the state of the registers. It is possible to single step the processor at machine level using these instructions. Refer to *Support for debugging/breakpointing in transputers* (technical note 61) for more detailed information regarding debugger support.

#### 4.4 Floating point errors

The instructions in table 4.7 are the only ones which can affect the floating point error flag *FP\_Error* (page 149). *Error* is set from this flag by *fpcheckerror* if *FP\_Error* is set.

Table 4.7 Floating point error setting instructions

fpadd fpldnladdsn	fpsub fpldnladddb	fpmul fpldnlmulsn	fpdiv fpldnlmuldb
fpremfirst	fpusqrtfirst	fpgt	fpeq
fpuseterror fpuexpincbv32	fpuclearerror fpuexpdecby32	fptesterror fpumulby2	fpudivby2
fpur32tor64	fpur64tor32	fpucki32	fpucki64
fprtoi32	fpuabs	fpint	

Table	48	IMS	T801	function	codes

Function Code	Memory Code	Mnemonic	Processor Cycles	Name	D E
0	0X	j	3	jump	D
1	1X	ldlp	1	load local pointer	
2	2X	pfix	1	prefix	
3	3X	ldnl	2	load non-local	
4	4X	ldc	1	load constant	
5	5X	ldnlp	1	load non-local pointer	
6	6X	nfix	1	negative prefix	
7	7X	ldl	2	load local	
8	8X	adc	1	add constant	E
9	9X	call	7	call	1
Α	AX	cj	2	conditional jump (not taken)	
			4	conditional jump (taken)	
В	вх	ajw	1	adjust workspace	1
С	CX	eqc	2	equals constant	
D	DX	stl	1	store local	
E	EX	stnl	2	store non-local	
F	FX	opr		operate	

Table 4.9 IMS T801 arithmetic/logical operation codes

Operation Code	Memory Code	Mnemonic	Processor Cycles	Name	D E
46	24F6	and	1	and	
4B	24FB	or	1	or	
33	23F3	xor	1	exclusive or	
32	23F2	not	1	bitwise not	
41	24F1	shl	n+2	shift left	
40	24F0	shr	n+2	shift right	
05	F5	add	1	add	E
0C	FC	sub	1	subtract	E
53	25F3	mul	38	multiply	E
72	27F2	fmul	35	fractional multiply (no rounding)	E
			40	fractional multiply (rounding)	E
2C	22FC	div	39	divide	E
1F	21FF	rem	37	remainder	E
09	F9	gt	2	greater than	
04	F4	diff	1	difference	
52	25F2	sum	1	sum	
08	F8	prod	b+4	product for positive register A	
			m+5	product for negative register A	

Operation Code	Memory Code	Mnemonic	Processor Cycles	Name	D E
16	21F6	ladd	2	long add	E
38	23F8	Isub	2	long subtract	E
37	23F7	Isum	3	long sum	
4F	24FF	Idiff	3	long diff	
31	23F1	Imul	33	long multiply	
1A	21FA	ldiv	35	long divide	E
36	23F6	Ishl	n+3	long shift left (n<32)	
		)	n-28	long shift left(n≥32)	
35	23F5	Ishr	n+3	long shift right (n<32)	
			<b>n</b> -28	long shift right (n≥32)	
19	21F9	norm	<b>n</b> +5	normalise (n<32)	
			<b>n</b> -26	normalise (n≥32)	
			3	normalise (n=64)	

Table 4.10 IMS T801 long arithmetic operation codes

Table 4.11 IMS T801 general operation codes

Operation Code	Memory Code	Mnemonic	Processor Cycles	Name	D E
00	F0	rev	1	reverse	
3A	23FA	xword	4	extend to word	
56	25F6	cword	5	check word	E
1D	21FD	xdble	2	extend to double	
4C	24FC	csngl	3	check single	E
42	24F2	mint	1	minimum integer	
5A	25FA	dup	1	duplicate top of stack	
79	27F9	pop	1	pop processor stack	

Table 4.12 IMS T801 2D block move operation codes

Operation Code	Memory Code	Mnemonic	Processor Cycles	Name	ΒD
5B	25FB	move2dinit	8	initialise data for 2D block move	
5C	25FC	move2dall	(2p+23)*r	2D block copy	
5D	25FD	move2dnonzero	(2p+23)*r	2D block copy non-zero bytes	
5E	25FE	move2dzero	(2p+23)*r	2D block copy zero bytes	

Table 4.13 IMS T801 CRC and bit operation codes

Operation Code	Memory Code	Mnemonic	Processor Cycles	Name	D E
74	27F4	crcword	35	calculate crc on word	
75	27F5	crcbyte	11	calculate crc on byte	
76	27F6	bitcnt	<b>b</b> +2	count bits set in word	
77	27F7	bitrevword	36	reverse bits in word	
78	27F8	bitrevnbits	n+4	reverse bottom n bits in word	

Operation Code	Memory Code	Mnemonic	Processor Cycles	Name	D
02	F2	bsub	1	byte subscript	
0A	FA	wsub	2	word subscript	ľ
81	28F1	wsubdb	3	form double word subscript	
34	23F4	bcnt	2	byte count	
3F	23FF	wcnt	5	word count	
01	F1	lb	5	load byte	
3B	23FB	sb	4	store byte	
4A	24FA	move	2 <b>w</b> +8	move message	

Table 4.14 IMS T801 indexing/array operation codes

Table 4.15 IMS T801 timer handling operation codes

Operation Code	Memory Code	Mnemonic	Processor Cycles	Name	D E
22	22F2	ldtimer	2	load timer	
2B	22FB	tin	30	timer input (time future)	D
			4	timer input (time past)	D
4E	24FE	talt	4	timer alt start	
51	25F1	taltwt	15	timer alt wait (time past)	D
			48	timer alt wait (time future)	D
47	24F7	enbt	8	enable timer	
2E	22FE	dist	23	disable timer	

Table 4.16 IMS T801 input/output operation codes

Operation	Memory		Processor		D
Code	Code	Mnemonic	Cycles	Name	E
07	F7	in	2 <b>w</b> +19	input message	D
0B	FB	out	2 <b>w</b> +19	output message	D
0F	FF	outword	23	output word	D
0E	FE	outbyte	23	output byte	D
43	24F3	alt	2	alt start	
44	24F4	altwt	5	alt wait (channel ready)	D
			17	alt wait (channel not ready)	D
45	24F5	altend	4	alt end	
49	24F9	enbs	3	enable skip	
30	23F0	diss	4	disable skip	
	2010	uiss	7	disable skip	
12	21F2	resetch	3	reset channel	
48	24F8	enbc	7	enable channel (ready)	
			5	enable channel (not ready)	
2F	22FF	disc	8	disable channel	

Operation Code	Memory Code	Mnemonic	Processor Cycles	Name	D E
20	22F0	ret	5	return	
1B	21FB	ldpi	2	load pointer to instruction	İ
3C	23FC	gajw	2	general adjust workspace	Ì
06	F6	gcall	4	general call	
21 22F1	lend	10	loop end (loop)	D	
			5	loop end (exit)	D

Table 4.17 IMS T801 control operation codes

Table 4.18 IMS T801 scheduling operation codes

Operation Code	Memory Code	Mnemonic	Processor Cycles	Name	D E
0D	FD	startp	12	start process	D
03	F3	endp	13	end process	D
39	23F9	runp	10	run process	
15	21F5	stopp	11	stop process	
1E	21FE	Idpri	1	load current priority	

Table 4.19 IMS T801 error handling operation codes

Operation Code	Memory Code	Mnemonic	Processor Cycles	Name	D E
13	21F3	csub0	2	check subscript from 0	E
4D	24FD	ccnt1	3	check count from 1	E
29	22F9	testerr	2	test error false and clear (no error)	
			3	test error false and clear (error)	
10	21F0	seterr	1	set error	E
55	25F5	stoperr	2	stop on error (no error)	D
57	25F7	cirhaiterr	1	clear halt-on-error	
58	25F8	sethalterr	1	set halt-on-error	
59	25F9	testhalterr	2	test halt-on-error	

Table 4.20 IMS T801 processor initialisation operation codes

Operation Code	Memory Code	Mnemonic	Processor Cycles	Name	D E
2A	22FA	testpranal	2	test processor analysing	
3E	23FE	saveh	4	save high priority queue registers	
3D	23FD	savel	4	save low priority queue registers	
18	21F8	sthf	1	store high priority front pointer	
50	25F0	sthb	1	store high priority back pointer	
1C	21FC	stlf	1	store low priority front pointer	
17	21F7	stlb	1	store low priority back pointer	
54	25F4	sttimer	1	store timer	
17C	2127FC	Iddevid	1	load device identity	
7E	27FE	Idmemstartval	1	load value of memstart address	

Operation Code	Memory Code	Mnemonic	Processor Cycles	Name	DE
0	00	jump 0	3	jump 0 (break not enabled)	D
			11	jump 0 (break enabled, high priority)	
			13	jump 0 (break enabled, low priority)	
B1	2BF1	break	9	break (high priority)	
			11	break (low priority)	
B2	2BF2	clrj0break	1	clear jump 0 break enable flag	
B3	2BF3	setj0break	1	set jump 0 break enable flag	
B4	2BF4	testj0break	2	test jump 0 break enable flag set	
7 <b>A</b>	27FA	timerdisableh	1	disable high priority timer interrupt	
7B	27FB	timerdisablel	1	disable low priority timer interrupt	
7C	27FC	timerenableh	6	enable high priority timer interrupt	
7D	27FD	timerenablel	6	enable low priority timer interrupt	

Table 4.21 IMS T801 debugger support codes

Table 4.22 IMS T801 floating point load/store operation codes

Operation Code	Memory Code	Mnemonic	Processor Cycles	Name	D E
8E	28FE	fpidnisn	2	fp load non-local single	
8A	28FA	fpldnldb	3	fp load non-local double	
86	28F6	fpldnlsni	4	fp load non-local indexed single	
82	28F2	fpldnldbi	6	fp load non-local indexed double	
9F	29FF	fpldzerosn	2	load zero single	
A0	2AF0	fpldzerodb	2	load zero double	
AA	2AFA	fpldnladdsn	8/11	fp load non local & add single	
A6	2AF6	fpldnladddb	9/12	fp load non local & add double	
AC	2AFC	fpldnlmulsn	13/20	fp load non local & multiply single	
A8	2AF8	fpldnlmuldb	21/30	fp load non local & multiply double	F
88	28F8	fpstnlsn	2	fp store non-local single	
84	28F4	fpstnldb	3	fp store non-local double	
9E	29FE	fpstnli32	4	store non-local int32	

Processor cycles are shown as Typical/Maximum cycles.

Table 4.23 IMS T801 floating point general operation codes

Operation Code	Memory Code	Mnemonic	Processor Cycles	Name	D E
AB	2AFB	fpentry	1	floating point unit entry	
A4	2AF4	fprev	1	fp reverse	
A3	2AF3	fpdup	1	fp duplicate	

Operation Processor Memory D **Mnemonic** Name Е Code Code Cycles 22 set rounding mode to round nearest s fpurn 1 set rounding mode to round zero 06 fpurz 1 s set rounding mode to round positive 04 s fpurp 1 05 1 set rounding mode to round minus s fpurm

Table 4.24 IMS T801 floating point rounding operation codes

Table 4.25 IMS T801 floating point error operation codes

Operation Code	Memory Code	Mnemonic	Processor Cycles	Name	DE
83	28F3	fpchkerror	1	check fp error	E
9C	29FC	fptesterror	2	test fp error false and clear	F
23	S	fpuseterror	1	set fp error	F
9C	s	fpuclearerror	1	clear fp error	F

Table 4.26 IMS T801 floating point comparison operation codes

Operation Code	Memory Code	Mnemonic	Processor Cycles	Name	D E
94	29F4	fpgt	4/6	fp greater than	F
95	29F5	fpeq	3/5	fp equality	F
92	29F2	fpordered	3/4	fp orderability	
91	29F1	fpnan	2/3	fp NaN	
93	29F3	fpnotfinite	2/2	fp not finite	
0E	29F5 fpeq 29F2 fpordered 29F1 fpnan 29F3 fpnotfinite s fpuchki32		3/4	check in range of type int32	F
0F	s	fpuchki64	3/4	check in range of type int64	F

Processor cycles are shown as Typical/Maximum cycles.

Table 4.27 IMS T801 floating point conversion operation codes

Operation Code	Memory Code	Mnemonic	Processor Cycles	Name	D E
07	S	fpur32tor64	3/4	real32 to real64	F
08	S	fpur64tor32	6/9	real64 to real32	F
9D	29FD	fprtoi32	7/9	real to int32	F
96	29F6	fpi32tor32	8/10	int32 to real32	1
98	29F8	fpi32tor64	8/10	int32 to real64	ł
9A	29FA	fpb32tor64	8/8	bit32 to real64	ł
0D	s	fpunoround	2/2	real64 to real32, no round	
A1	2AF1	fpint	5/6	round to floating integer	F

Processor cycles are shown as Typical/Maximum cycles.

F

11

Operation Memory Processor cycles D Ε Code Code **Mnemonic** Single Double Name 28F7 87 fpadd 6/9 6/9 fp add F F 89 fpsub 28F9 6/9 6/9 fp subtract F 8B 28FB fpmul 11/18 18/27 fp multiply 8C 28FC fpdiv 16/28 31/43 fp divide F F 0B s fpuabs 2/2 2/2 fp absolute F 8F 28FF foremfirst 36/46 36/46 fp remainder first step 29F0 32/36 90 fpremstep 32/36 fp remainder iteration 01 27/29 27/29 F fpusgrtfirst fp square root first step s 02 fpusgrtstep 42/42 42/42 fp square root step s 03 fpusqrtlast 8/9 8/9 fp square root end s 0A 6/9 6/9 multiply by 232 F fpuexpinc32 s 6/9 6/9 divide by 232 F 09 s fpuexpdec32 12 fpumulby2 6/9 6/9 multiply by 2.0 F s

6/9

6/9

divide by 2.0

Table 4.28 IMS T801 floating point arithmetic operation codes

Processor cycles are shown as Typical/Maximum cycles.

fpudivby2

s

## 5 Floating point unit

The 64 bit FPU provides single and double length arithmetic to floating point standard ANSI-IEEE 754-1985. It is able to perform floating point arithmetic concurrently with the central processor unit (CPU), sustaining 3.3 Mflops on a 30 MHz device. All data communication between memory and the FPU occurs under control of the CPU.

The FPU consists of a microcoded computing engine with a three deep floating point evaluation stack for manipulation of floating point numbers. These stack registers are *FA*, *FB* and *FC*, each of which can hold either 32 bit or 64 bit data; an associated flag, set when a floating point value is loaded, indicates which. The stack behaves in a similar manner to the CPU stack (page 132).

As with the CPU stack, the FPU stack is not saved when rescheduling (page 135) occurs. The FPU can be used in both low and high priority processes. When a high priority process interrupts a low priority one the FPU state is saved inside the FPU. The CPU will service the interrupt immediately on completing its current operation. The high priority process will not start, however, before the FPU has completed its current operation.

Points in an instruction stream where data need to be transferred to or from the FPU are called *synchronisation points*. At a synchronisation point the first processing unit to become ready will wait until the other is ready. The data transfer will then occur and both processors will proceed concurrently again. In order to make full use of concurrency, floating point data source and destination addresses can be calculated by the CPU whilst the FPU is performing operations on a previous set of data. Device performance is thus optimised by minimising the CPU and FPU idle times.

The FPU has been designed to operate on both single length (32 bit) and double length (64 bit) floating point numbers, and returns results which fully conform to the ANSI-IEEE 754-1985 floating point arithmetic standard. Denormalised numbers are fully supported in the hardware. All rounding modes defined by the standard are implemented, with the default being round to nearest.

The basic addition, subtraction, multiplication and division operations are performed by single instructions. However, certain less frequently used floating point instructions are selected by a value in register A (when allocating registers, this should be taken into account). A *load constant* instruction *ldc* is used to load register A; the *floating point entry* instruction *fpentry* then uses this value to select the floating point operation. This pair of instructions is termed a *selector sequence*.

Names of operations which use *fpentry* begin with *fpu*. A typical usage, returning the absolute value of a floating point number, would be

Idc fpuabs; fpentry;

Since the indirection code for fpuabs is **0B**, it would be encoded as

Table 5.1 *fpentry* coding

Mnem	Mnemonic		Memory code
ldc	fpuabs	#4	#4B
fpentry is coded as	(op. code #AB)		#2AFB
pfix opr	#A #B	#2 #F	#2A #FB

The *remainder* and *square root* instructions take considerably longer than other instructions to complete. In order to minimise the interrupt latency period of the transputer they are split up to form instruction sequences. As an example, the instruction sequence for a single length square root is

fpusqrtfirst; fpusqrtstep; fpusqrtstep; fpusqrtlast;

The FPU has its own error flag *FP\_Error*. This reflects the state of evaluation within the FPU and is set in circumstances where invalid operations, division by zero or overflow exceptions to the ANSI-IEEE 754-1985 standard would be flagged (page 141). *FP\_Error* is also set if an input to a floating point operation is infinite or is not a number (NaN). The *FP\_Error* flag can be set, tested and cleared without affecting the main *Error* flag, but can also set *Error* when required (page 141). Depending on how a program is compiled, it is possible for both unchecked and fully checked floating point arithmetic to be performed.

Further details on the operation of the FPU can be found in *Transputer Instruction Set - A Compiler Writers' Guide.* 

T801-20 T801-30 Operation Single length Double length Single length Double length 350 ns 350 ns 233 ns add 233 ns 350 ns 233 ns 233 ns subtract 350 ns 367 ns multiply 550 ns 1000 ns 667 ns 1600 ns divide 850 ns 567 ns 1067 ns

Table 5.2 Typical floating point operation times for IMS T801

Timing is for operations where both operands are normalised fp numbers.

## 6 System services

System services include all the necessary logic to initialise and sustain operation of the device. They also include error handling and analysis facilities.

#### 6.1 Power

Power is supplied to the device via the VCC and GND pins. The supply must be decoupled close to the chip by at least one 100 nF low inductance (e.g. ceramic) capacitor between VCC and GND. Four layer boards are recommended; if two layer boards are used, extra care should be taken in decoupling.

Input voltages must not exceed specification with respect to VCC and GND, even during power-up and power-down ramping, otherwise *latchup* can occur. CMOS devices can be permanently damaged by excessive periods of latchup.

#### 6.2 CapPlus, CapMinus

The internally derived power supply for internal clocks requires an external low leakage, low inductance  $1\mu F$  capacitor to be connected between **CapPlus** and **CapMinus**. A ceramic capacitor is preferred, with an impedance less than 3 Ohms between 100 KHz and 10 MHz. If a polarised capacitor is used the negative terminal should be connected to **CapMinus**. Total PCB track length should be less than 50 mm. The connections must not touch power supplies or other noise sources.

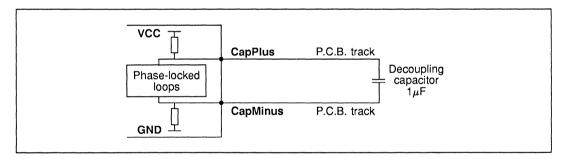


Figure 6.1 Recommended PLL decoupling

#### 6.3 Clockin

Transputer family components use a standard clock frequency, supplied by the user on the **ClockIn** input. The nominal frequency of this clock for all transputer family components is 5 MHz, regardless of device type, transputer word length or processor cycle time. High frequency internal clocks are derived from **ClockIn**, simplifying system design and avoiding problems of distributing high speed clocks externally.

A number of transputer devices may be connected to a common clock, or may have individual clocks providing each one meets the specified stability criteria. In a multi-clock system the relative phasing of **ClockIn** clocks is not important, due to the asynchronous nature of the links. Mark/space ratio is unimportant provided the specified limits of **ClockIn** pulse widths are met.

Oscillator stability is important. ClockIn must be derived from a crystal oscillator; RC oscillators are not sufficiently stable. ClockIn must not be distributed through a long chain of buffers. Clock edges must be monotonic and remain within the specified voltage and time limits.

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTE
TDCLDCH	ClockIn pulse width low	40			ns	1
TDCHDCL	Clockin pulse width high	40	ł		ns	1
TDCLDCL	ClockIn period	į	200		ns	1,2,4
TDCerror	ClockIn timing error			±0.5	ns	1,3
TDC1DC2	Difference in ClockIn for 2 linked devices			400	ppm	1,4
TDCr	ClockIn rise time			10	ns	1,5
TDCf	ClockIn fall time			8	ns	1.5

Table 6.1 Input clock

#### Notes

- 1 These parameters are not tested.
- 2 Measured between corresponding points on consecutive falling edges.
- 3 Variation of individual falling edges from their nominal times.
- 4 This value allows the use of 200 ppm crystal oscillators for two devices connected together by a link.
- 5 Clock transitions must be monotonic within the range VIH to VIL (table 11.3).

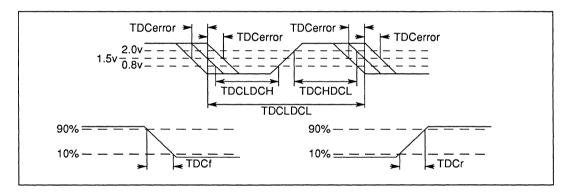


Figure 6.2 ClockIn timing

## 6.4 ProcSpeedSelect0-2

Processor speed of the IMS T801 is variable in discrete steps. The desired speed can be selected, up to the maximum rated for a particular component, by the three speed select lines **ProcSpeedSelect0-2**. The pins are tied high or low, according to table 6.2, for the various speeds. The frequency of **ClockIn** for the speeds given in table 6.2 is 5 MHz. There are six valid speed select combinations.

6 System services 153

Proc Speed Select2	Proc Speed Select1	Proc Speed Select0	Processor Clock Speed MHz	Processor Cycle Time ns	Notes
0	0	0	20.0	50.0	
0	0	1	22.5	44.4	
0	1	0	25.0	40.0	1
0	1	1	30.0	33.3	
1	0	0	35.0	28.6	
1	0	1			Invalid
1 1	1	0	17.5	57.1	1
1	1	1			Invalid

Table 6.2 Processor speed selection

Note: Inclusion of a speed selection in this table does not imply immediate availability.

#### 6.5 Reset

Reset can go high with VCC, but must at no time exceed the maximum specified voltage for VIH. After VCC is valid ClockIn should be running for a minimum period TDCVRL before the end of Reset. The falling edge of Reset initialises the transputer and starts the bootstrap routine. Link outputs are forced low during reset; link inputs and EventReq should be held low. Memory request (DMA) must not occur whilst Reset is high but can occur before bootstrap (page 167).

If **BootFromRom** is high, bootstrapping will take place immediately after **Reset** goes low, using data from external memory; otherwise the transputer will await an input from any link. The processor will be in the low priority state.

### 6.6 Bootstrap

The transputer can be bootstrapped either from a link or from external ROM. To facilitate debugging, **Boot-FromRom** may be dynamically changed but must obey the specified timing restrictions. It is sampled once only by the transputer, before the first instruction is executed after **Reset** is taken low.

If **BootFromRom** is connected high (e.g. to **VCC**) the transputer starts to execute code from the top two bytes in external memory, at address #7FFFFFFE. This location should contain a backward jump to a program in ROM. Following this access, **BootFromRom** may be taken low if required. The processor is in the low priority state, and the *W* register points to *MemStart* (page 157).

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTE
TPVRH	Power valid before Reset	10			ms	
TRHRL	Reset pulse width high	8			ClockIn	1
TDCVRL	ClockIn running before Reset end	10	1		ms	2
TAHRH	Analyse setup before Reset	3			ms	
TRLAL	Analyse hold after Reset end	1			ClockIn	1
TBRVRL	BootFromRom setup	0			ms	
TRLBRX	BootFromRom hold after Reset	0	1		ms	3
TALBRX	BootFromRom hold after Analyse				l	3

Table 6.3 Reset and Analyse

### Notes

- 1 Full periods of Clockin TDCLDCL required.
- 2 At power-on reset.
- 3 Must be stable until after end of bootstrap period. See Bootstrap section.

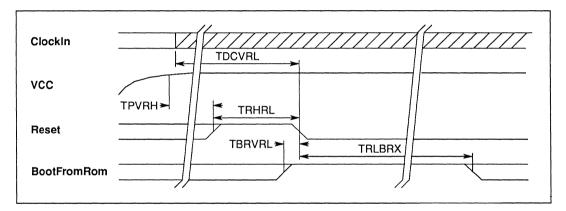


Figure 6.3 Transputer reset timing with Analyse low

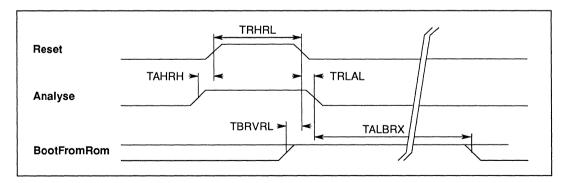


Figure 6.4 Transputer reset and analyse timing

If BootFromRom is connected low (e.g. to GND) the transputer will wait for the first bootstrap message to arrive on any one of its links. The transputer is ready to receive the first byte on a link within two processor cycles TPCLPCL after Reset goes low.

If the first byte received (the control byte) is greater than 1 it is taken as the quantity of bytes to be input. The following bytes, to that quantity, are then placed in internal memory starting at location MemStart. Following reception of the last byte the transputer will start executing code at MemStart as a low priority process. BootFromRom may be taken high after reception of the last byte, if required. The memory space immediately above the loaded code is used as work space. Messages arriving on other links after the control byte has been received and on the bootstrapping link after the last bootstrap byte will be retained until a process inputs from them.

#### 6.7 Peek and poke

Any location in internal or external memory can be interrogated and altered when the transputer is waiting for a bootstrap from link. If the control byte is 0 then eight more bytes are expected on the same link. The first four byte word is taken as an internal or external memory address at which to poke (write) the second four byte word. If the control byte is 1 the next four bytes are used as the address from which to peek (read) a word of data; the word is sent down the output channel of the same link.

Following such a peek or poke, the transputer returns to its previously held state. Any number of accesses may be made in this way until the control byte is greater than 1, when the transputer will commence reading its bootstrap program. Any link can be used, but addresses and data must be transmitted via the same link as the control byte.

#### 6.8 **Analyse**

If **Analyse** is taken high when the transputer is running, the transputer will halt at the next descheduling point (page 140). From Analyse being asserted, the processor will halt within three time slice periods plus the time taken for any high priority process to complete. As much of the transputer status is maintained as is necessary to permit analysis of the halted machine. Processor flags Error and HaltOnError are not altered at reset, whether Analyse is asserted or not. Memory refresh continues.

Input links will continue with outstanding transfers. Output links will not make another access to memory for data but will transmit only those bytes already in the link buffer. Providing there is no delay in link acknowledgement, the links should be inactive within a few microseconds of the transputer halting.

Reset should not be asserted before the transputer has halted and link transfers have ceased. When Reset is taken low whilst Analyse is high, neither the memory configuration sequence nor the block of eight refresh cycles will occur; the previous memory configuration will be used for any external memory accesses. If BootFromRom is high the transputer will bootstrap as soon as Analyse is taken low, otherwise it will await a control byte on any link. If Analyse is taken low without Reset going high the transputer state and operation are undefined. After the end of a valid Analyse sequence the registers have the values given in table 6.4.

Table 6.4 Register values after Analyse

- MemStart if bootstrapping from a link, or the external memory bootstrap address if bootstrapping from ROM.
- W MemStart if bootstrapping from ROM, or the address of the first free word after the bootstrap program if bootstrapping from link.
- The value of I when the processor halted.
- В The value of W when the processor halted, together with the priority of the process when the transputer was halted (i.e. the W descriptor).
- C The ID of the bootstrapping link if bootstrapping from link.

#### 6.9 ErrorOut

The **ErrorOut** pin is connected directly to the internal *Error* flag and follows the state of that flag. If **ErrorOut** is high it indicates an error in one of the processes caused, for example, by arithmetic overflow, divide by zero, array bounds violation or software setting the flag directly (page 141). It can also be set from the floating point unit under certain circumstances (page 141, 149). Once set, the *Error* flag is only cleared by executing the instruction *testerr*. The error is not cleared by processor reset, in order that analysis can identify any errant transputer (page 155).

A process can be programmed to stop if the *Error* flag is set; it cannot then transmit erroneous data to other processes, but processes which do not require that data can still be scheduled. Eventually all processes which rely, directly or indirectly, on data from the process in error will stop through lack of data.

By setting the *HaltOnError* flag the transputer itself can be programmed to halt if *Error* becomes set. If *Error* becomes set after *HaltOnError* has been set, all processes on that transputer will cease but will not necessarily cause other transputers in a network to halt. Setting *HaltOnError* after *Error* will not cause the transputer to halt; this allows the processor reset and analyse facilities to function with the flags in indeterminate states.

An alternative method of error handling is to have the errant process or transputer cause all transputers to halt. This can be done by applying the **ErrorOut** output signal of the errant transputer to the **EventReq** pin of a suitably programmed master transputer. Since the process state is preserved when stopped by an error, the master transputer can then use the analyse function to debug the fault. When using such a circuit, note that the *Error* flag is in an indeterminate state on power up; the circuit and software should be designed with this in mind.

Error checks can be removed completely to optimise the performance of a proven program; any unexpected error then occurring will have an arbitrary undefined effect.

If a high priority process pre-empts a low priority one, status of the *Error* and *HaltOnError* flags is saved for the duration of the high priority process and restored at the conclusion of it. Status of the *Error* flag is transmitted to the high priority process but the *HaltOnError* flag is cleared before the process starts. Either flag can be altered in the process without upsetting the error status of any complex operation being carried out by the pre-empted low priority process.

In the event of a transputer halting because of *HaltOnError*, the links will finish outstanding transfers before shutting down. If **Analyse** is asserted then all inputs continue but outputs will not make another access to memory for data. Memory refresh will continue to take place.

After halting due to the *Error* flag changing from 0 to 1 whilst *HaltOnError* is set, register *I* points two bytes past the instruction which set *Error*. After halting due to the **Analyse** pin being taken high, register *I* points one byte past the instruction being executed. In both cases *I* will be copied to register *A*.

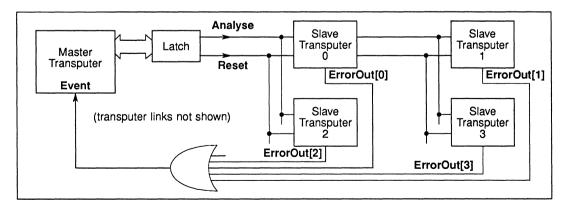


Figure 6.5 Error handling in a multi-transputer system

# 7 Memory

The IMS T801 can access 4 Gbytes of external memory space. The IMS T801 also has 4 Kbytes of fast internal static memory for high rates of data throughput. Each internal memory access takes one processor cycle **ProcClockOut** (page 162). Internal and external memory are part of the same linear address space.

IMS T801 memory is byte addressed, with words aligned on four-byte boundaries. The least significant byte of a word is the lowest addressed byte.

The bits in a byte are numbered 0 to 7, with bit 0 the least significant. The bytes are numbered from 0, with byte 0 the least significant. In general, wherever a value is treated as a number of component values, the components are numbered in order of increasing numerical significance, with the least significant component numbered 0. Where values are stored in memory, the least significant component value is stored at the lowest (most negative) address.

Internal memory starts at the most negative address #80000000 and extends to #80000FFF. User memory begins at #80000070; this location is given the name *MemStart*.

The reserved area of internal memory below MemStart is used to implement link and event channels.

Two words of memory are reserved for timer use, *TPtrLoc0* for high priority processes and *TPtrLoc1* for low priority processes. They either indicate the relevant priority timer is not in use or point to the first process on the timer gueue at that priority level.

Values of certain processor registers for the current low priority process are saved in the reserved *IntSaveLoc* locations when a high priority process pre-empts a low priority one. Other locations are reserved for extended features such as block moves and floating point operations.

External memory space starts at #80001000 and extends up through #00000000 to #7FFFFFFF. ROM bootstrapping code must be in the most positive address space, starting at #7FFFFFFE. Address space immediately below this is conventionally used for ROM based code.

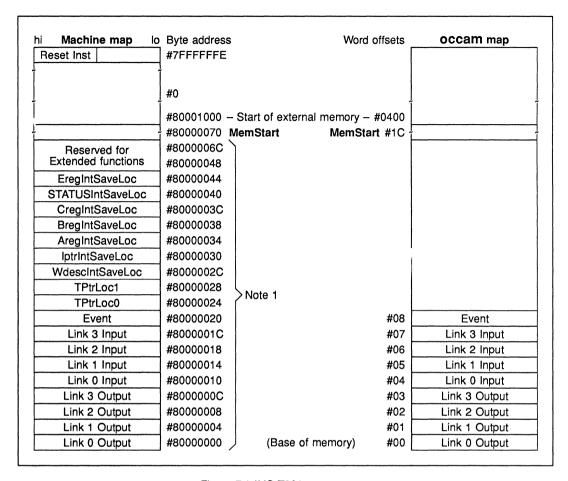


Figure 7.1 IMS T801 memory map

#### Notes

1 These locations are used as auxiliary processor registers and should not be manipulated by the user. Like processor registers, their contents may be useful for implementing debugging tools (Analyse, page 155). For details see *Transputer Instruction Set - A Compiler Writers' Guide*.

# 8 External memory interface

The IMS T801 External Memory Interface (EMI) allows access to a 32 bit address space via separate address and data buses.

The external memory cycle is divided into four Tstates with the following functions:

- T1 Address and control setup time.
- T2 Data setup time.
- T3 Data read/write.
- T4 Data and address hold after access.

Each **Tstate** is half a processor cycle **TPCLPCL** long. An external memory cycle is always a complete number of cycles **TPCLPCL** in length. The start of **T1** always coincides with a rising edge of **ProcClockOut**. **T2** can be extended indefinitely by adding externally generated wait states of one complete processor cycle each.

During an internal memory access cycle the external memory interface address bus MemA2-31 reflects the word address used to access internal RAM, notMemWrB0-3 and notMemCE are inactive and the data bus MemD0-31 is tristated. This is true unless and until a DMA (memory request) activity takes place, when the MemA2-31, MemD0-31, notMemCE and notMemWrB0-3 signals will be placed in a high impedance state by the transputer.

Bus activity is not adequate to trace the internal operation of the transputer in full, but may be used for hardware debugging in conjuction with peek and poke (page 155).

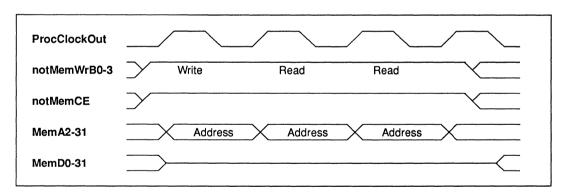


Figure 8.1 IMS T801 bus activity for 3 internal memory cycles

#### 8.1 Pin functions

# 8.1.1 MemA2-31

External memory addresses are output on a non-multiplexed 30 bit bus. The address is valid at the start of T1 and remains so until the end of T4.

#### 8.1.2 MemD0-31

The non-multiplexed data bus is 32 bits wide. The data bus is high impedance except when the transputer is writing data. If only one byte is being written, the unused 24 bits of the bus are high impedance at that time.

If the data setup time for read or write is too short it can be extended by inserting wait states at the end of T2.

# 8.1.3 notMemCE

The active low signal notMemCE is used to enable external memory on both read and write cycles.

Table 8.1 notMemCE to ProcClockOut skew

		T801-30		T801-25		T801-20		
SYMBOL	PARAMETER	MIN	MAX	MIN	MAX	MIN	MAX	NOTE
TPCHEL	notMemCE falling from ProcClockOut rising	6	10	8	12	10	14	1
TPCLEH	ProcClockOut falling to notMemCE rising	6	10	8	12	10	14	1

#### Notes

1 Units are ns.

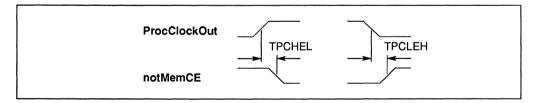


Figure 8.2 IMS T801 skew of notMemCE to ProcClockOut

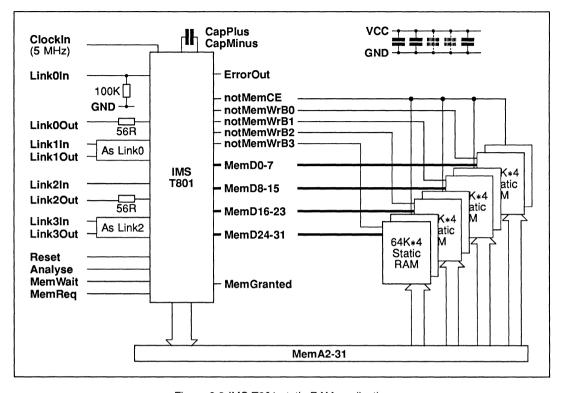


Figure 8.3 IMS T801 static RAM application

# 8.1.4 notMemWrB0-3

Four write enables **notMemWrB0-3** are provided, one to write each byte of a word. When writing a word, the four appropriate write enables are asserted; when writing a byte only the appropriate write enable is asserted.

#### 8.1.5 MemWait

Wait states can be selected by taking MemWait high. Externally generated wait states of one complete processor cycle can be added to extend the duration of T2 indefinitely.

# 8.1.6 MemReg, MemGranted

Direct memory access (DMA) can be requested at any time by driving the asynchronous MemReq input high.

**MemGranted** follows the timing of the bus being tristated and can be used to signal to the device requesting the DMA that it has control of the bus. Note that **MemGranted** changes on the falling edge of **ProcClockOut** and can therefore be sampled to establish control of the bus on the rising edge of **ProcClockOut**.

# 8.1.7 ProcClockOut

This clock is derived from the internal processor clock, which is in turn derived from ClockIn. Its period is equal to one internal microcode cycle time, and can be derived from the formula

#### TPCLPCL = TDCLDCL / PLLx

where TPCLPCL is the ProcClockOut Period, TDCLDCL is the ClockIn Period and PLLx is the phase lock loop factor for the relevant speed part, obtained from the ordering details (Ordering section).

Edges of the various external memory strobes are synchronised by, but do not all coincide with, rising or falling edges of ProcClockOut.

	Table	8.2	Pro	cClo	ckOut
--	-------	-----	-----	------	-------

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTE
TPCLPCL	ProcClockOut period	a-1	а	a+1	ns	1
TPCHPCL	ProcClockOut pulse width high	<b>b</b> -2.5	b	b+2.5	ns	2
TPCLPCH	ProcClockOut pulse width low	ļ	С		ns	3
TPCstab	ProcClockOut stability			4	%	4

- 1 a is TDCLDCL/PLLx.
- 2 b is 0.5\*TPCLPCL (half the processor clock period).
- 3 c is TPCLPCL-TPCHPCL.
- 4 Stability is the variation of cycle periods between two consecutive cycles, measured at corresponding points on the cycles.

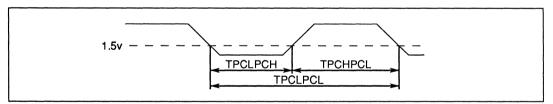


Figure 8.4 IMS T801 ProcClockOut timing

# 8.2 Read cycle

Read cycle data may be set up on the bus at any time after the start of T1, but must be valid when the IMS T801 reads it during T4. Data can be removed any time after the rising edge of **notMemCE**, but must be off the bus no later than the middle of T1, which allows for bus turn-around time before the data lines are driven at the start of T2 in a processor write cycle.

Byte addressing is carried out internally by the IMS T801 for read cycles.

		T801-30		T801-25		T801-20		
SYMBOL	PARAMETER	MIN	MAX	MIN	MAX	MIN	MAX	NOTE
TAVEL	Address valid before chip enable low	6		8		10		1,3
TELEH	Chip enable low	48	53	58	64	72	78	1,3
TEHEL	Delay before chip enable re-assertion	14		16		20		1,2,3
TEHAX	Address hold after chip enable high	6		8		10		1,3
TELDrV	Data valid from chip enable low	0	34	0	40	0	47	3
TAVDrV	Data valid from address valid	0	40	0	48	0	57	3
TDrVEH	Data setup before chip enable high	14		18		25		3
TEHDrZ	Data hold after chip enable high	0	14	0	16	0	20	3
TWEHEL	Write enable setup before chip enable	14		16		20		3,4
	low	İ				l		
TPCHEL	ProcClockOut high to chip enable low	6		8		10		1,3

Table 8.3 Read cycle

- 1 This parameter is common to read and write cycles and to byte-wide memory accesses.
- 2 These values assume back-to-back external memory accesses.
- 3 Units are ns.
- 4 Timing is for all four write enables notMemWrB0-3.

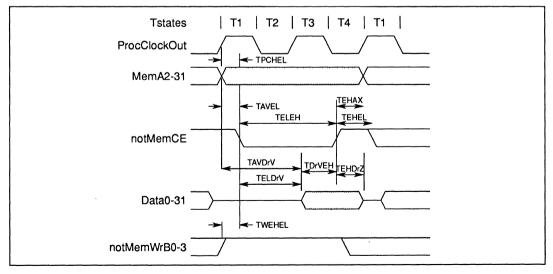


Figure 8.5 IMS T801 external read cycle

# 8.3 Write cycle

For write cycles the relevant bytes in memory are addressed by the write enables **notMemWrB0-3**. If a particular byte is not to be written, then the corresponding data outputs are tristated. **notMemWrB0** addresses the least significant byte.

The write enables are gated with the chip enable signal **notMemCE**, allowing them to be used without **notMemCE** for simple designs.

Data may be strobed into memory using **notMemWrB0-3** without the use of **notMemCE**, as the write enables go high between consecutive external memory write cycles.

Write data is placed on the data bus at the start of T2 and removed at the end of T4. The write cycle is completed with notMemCE going high.

		T801-30		T801-25		T801-20		
SYMBOL	PARAMETER	MIN	MAX	MIN	MAX	MIN	MAX	NOTE
TDwVEH	Data setup before chip enable high	33		40		50		1
TEHDwZ	Data hold after write	6	10	8	12	10	15	1
TDwZEL	Write data invalid to next chip enable	6		8		10		1
TWELEL	Write enable setup to chip enable low	-1	0	-2	0	-3	0	1,2
TEHWEH	Write enable hold after chip enable high	0	1	0	2	0	3	1,2

Table 8.4 Write cycle

- 1 Units are ns.
- 2 Timing is for all four write enables notMemWrB0-3.

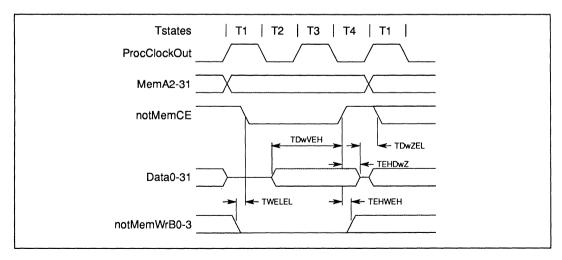


Figure 8.6 IMS T801 external write cycle

#### 8.4 Wait

Taking MemWait high with the timing shown in the diagram will extend the duration of T2 by one processor cycle TPCLPCL. One wait state comprises the pair W1 and W2. MemWait is sampled during T2, and should not change state in this region. If MemWait is still high when sampled in W2 then another wait period will be inserted. This can continue indefinitely. Internal memory access is unaffected by the number of wait states selected.

The wait state generator can be a simple digital delay line, synchronised to **notMemCE**. The **Single Wait State Generator** circuit in figure 8.7 can be extended to provide two or more wait states, as shown in figure 8.8.

		T801-30		T801-25		T801-20		
SYMBOL	PARAMETER	MIN	MAX	MIN	MAX	MIN	MAX	NOTE
TPCHWtH	MemWait asserted after ProcClockOut high		16		20		25	1
TAVWtH	MemWait asserted after Address valid		16		20		25	1
TPCHWtL	Wait low after ProcClockOut high	22		28		35		1

Table 8.5 Memory wait

## Notes

1 Units are ns.

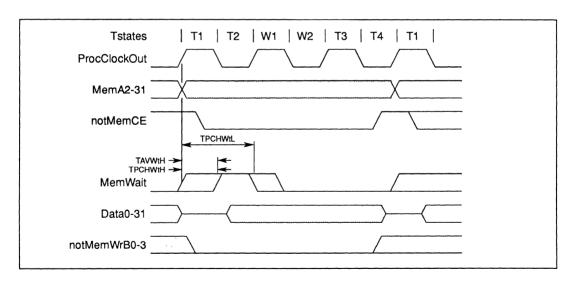


Figure 8.7 IMS T801 memory wait timing

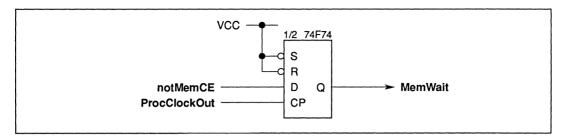


Figure 8.7 Single wait state generator

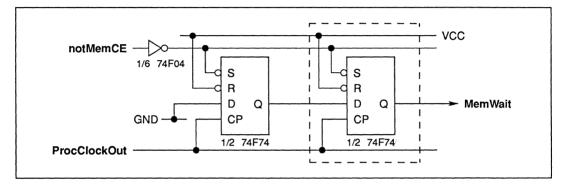


Figure 8.8 Extendable wait state generator

# 8.5 Direct memory access

Direct memory access (DMA) can be requested at any time by driving the asynchronous MemReq input high. MemReq is sampled during T1 of the processor cycle and the DMA device will then have control of the bus at the beginning of the next processor cycle, (after one ProcClockOut for internal accesses and two ProcClockOut cycles for external memory accesses, without wait states). When the processor transfers control of the bus the signals MemA2-31, notMemWrB0-3 and notMemCE are tristated and MemGranted is asserted high. MemGranted follows the timing of the bus being tristated and can be used to signal to the device requesting the DMA that it has control of the bus. Note that MemGranted changes on the falling edge of ProcClockOut and can therefore be sampled to establish control of the bus on the rising edge of ProcClockOut. During the DMA cycles, MemReq is sampled during each high phase of ProcClockOut and after it is taken low, control of the bus will be returned to the processor within two ProcClockOut cycles.

The processor is still able to access its internal memory while the DMA transfer proceeds, however when an external memory request is made the processor is forced to wait until the end of the DMA request. The DMA device has no access to the transputer's internal memory.

While control of the bus is being transferred from the processor to the DMA device, an extra clock phase, (one quarter of a **ProcClockOut** cycle) is allowed before the DMA transfer begins to ensure that the **not-MemCE** and **notMemWrB0-3** signals have been driven high before being tristated. This normally removes the requirement for external pull-up resistors.

DMA allows a bootstrap program to be loaded into external memory for execution after reset. If MemReq is asserted high during reset, MemGranted will be asserted high allowing access to the external memory before the bootstrap sequence begins. MemReq must be asserted for at least one period of TDCLDCL of ClockIn before Reset is asserted. The DMA control circuitry should ensure that correct operation will result if Reset should interrupt a normal DMA cycle.

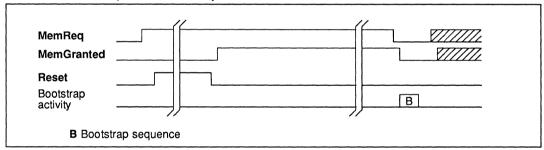


Figure 8.9 IMS T801 DMA sequence at reset

		T801-30		T801-25		T801-20		
SYMBOL	PARAMETER	MIN	MAX	MIN	MAX	MIN	MAX	NOTE
TMRHMGH	Memory request response time	58	а	70	а	85	а	1,2
TMRLMGL	Memory request end response time	60	66	75	80	90	100	2
TAZMGH	Address bus tristate before MemGranted	0		0		0		2
TDZMGH	Data bus tristate before MemGranted	0		0		0		2

Table 8.6 Memory request

- 1 Maximum response time a depends on whether an external memory cycle is in progress. Maximum time is (2 processor cycles) + (number of wait state cycles) for word access.
- 2 Units are ns.

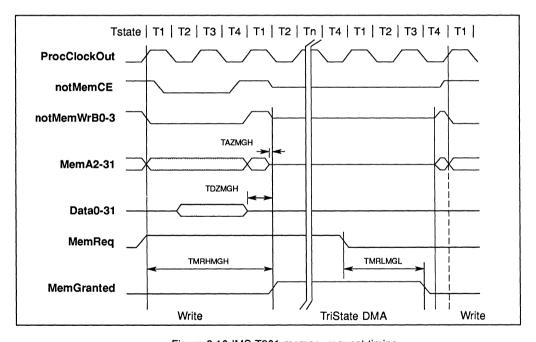


Figure 8.10 IMS T801 memory request timing

#### 9 Events

**EventReq** and **EventAck** provide an asynchronous handshake interface between an external event and an internal process. When an external event takes **EventReq** high the external event channel (additional to the external link channels) is made ready to communicate with a process. When both the event channel and the process are ready the processor takes **EventAck** high and the process, if waiting, is scheduled. **EventAck** is removed after **EventReq** goes low.

**EventWaiting** is asserted high by the transputer when a process executes an input on the event channel; typically with the occam **EVENT** ? **ANY** instruction. It remains high whilst the transputer is waiting for or servicing **EventReq** and is returned low when **EventAck** goes high. The **EventWaiting** pin changes near the falling edge of **ProcClockOut** and can therefore be sampled by the rising edge of **ProcClockOut**.

The **EventWaiting** pin can only be asserted by executing an *in* instruction on the event channel. The **EventWaiting** pin is not asserted high when an enable channel (*enbc*) instruction is executed on the Event channel (during an ALT construct in occam, for example). The **EventWaiting** pin can be asserted by executing the occam input on the event channel (such as **Event** ? **ANY**), provided that this does not occur as a guard in an alternative process. The **EventWaiting** pin can not be used to signify that an alternative process (ALT) is waiting on an input from the event channel.

**EventWaiting** allows a process to control external logic; for example, to clock a number of inputs into a memory mapped data latch so that the event request type can be determined.

Only one process may use the event channel at any given time. If no process requires an event to occur **EventAck** will never be taken high. Although **EventReq** triggers the channel on a transition from low to high, it must not be removed before **EventAck** is high. **EventReq** should be low during **Reset**; if not it will be ignored until it has gone low and returned high. **EventAck** is taken low when **Reset** occurs.

If the process is a high priority one and no other high priority process is running, the latency is as described on page 136. Setting a high priority task to wait for an event input allows the user to interrupt a transputer program running at low priority. The time taken from asserting **EventReq** to the execution of the microcode interrupt handler in the CPU is four cycles. The following functions take place during the four cycles:

- Cycle 1 Sample EventReq at pad on the rising edge of ProcClockOut and synchronise.
- Cycle 2 Edge detect the synchronised EventReq and form the interrupt request.
- **Cycle 3** Sample interrupt vector for microcode ROM in the CPU.
- Cycle 4 Execute the interrupt routine for Event rather than the next instruction.

Table	9.1	Event	

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTE
TVHKH	Event request response	0			ns	
TKHVL	Event request hold	0	İ		ns	
TVLKL	Delay before removal of event acknowledge	0		<b>a</b> +5	ns	1
TKLVH	Delay before re-assertion of event request	0			ns	
TKHEWL	Event acknowledge to end of event waiting	0			ns	
TKLEWH	End of event acknowledge to event waiting	0			ns	

# Notes

1 a is 3 processor cycles TPCLPCL.

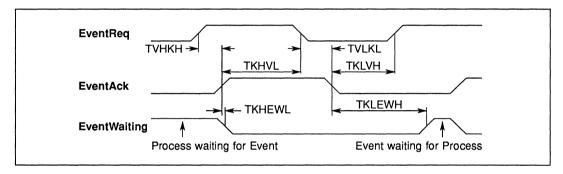


Figure 9.1 IMS T801 event timing

#### 10 Links

Four identical INMOS bi-directional serial links provide synchronized communication between processors and with the outside world. Each link comprises an input channel and output channel. A link between two transputers is implemented by connecting a link interface on one transputer to a link interface on the other transputer. Every byte of data sent on a link is acknowledged on the input of the same link, thus each signal line carries both data and control information.

The quiescent state of a link output is low. Each data byte is transmitted as a high start bit followed by a one bit followed by eight data bits followed by a low stop bit. The least significant bit of data is transmitted first. After transmitting a data byte the sender waits for the acknowledge, which consists of a high start bit followed by a zero bit. The acknowledge signifies both that a process was able to receive the acknowledged data byte and that the receiving link is able to receive another byte. The sending link reschedules the sending process only after the acknowledge for the final byte of the message has been received.

The IMS T801 links allow an acknowledge packet to be sent before the data packet has been fully received. This overlapped acknowledge technique is fully compatible with all other INMOS transputer links.

The IMS T801 links support the standard INMOS communication speed of 10 Mbits/sec. In addition they can be used at 20 Mbits/sec for IMS T801-20 and IMS T801-25. Links are not synchronised with **ClockIn** or **ProcClockOut** and are insensitive to their phases. Thus links from independently clocked systems may communicate, providing only that the clocks are nominally identical and within specification.

Links are TTL compatible and intended to be used in electrically quiet environments, between devices on a single printed circuit board or between two boards via a backplane. Direct connection may be made between devices separated by a distance of less than 300 millimetres. For longer distances a matched 100 Ohm transmission line should be used with series matching resistors **RM**. When this is done the line delay should be less than 0.4 bit time to ensure that the reflection returns before the next data bit is sent.

Buffers may be used for very long transmissions. If so, their overall propagation delay should be stable within the skew tolerance of the link, although the absolute value of the delay is immaterial.

Link speeds can be set by LinkSpeed. LinkSpeed allows Links 0, 1, 2 or 3 to be set to 10 or 20 Mbits/sec. Table 10.1 shows uni-directional and bi-directional data rates in Kbytes/sec for each link speed. Data rates are quoted for a transputer using internal memory, and will be affected by a factor depending on the number of external memory accesses and the length of the external memory cycle.

Table 10.1	Speed	Settings for	Transputer	Links
------------	-------	--------------	------------	-------

Link		Kbytes/sec					
Special	Mbits/sec	Uni	Bi				
0	10	910	1250				
1	20	1740	2350				

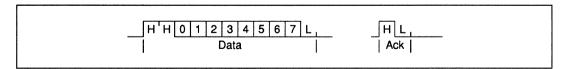


Figure 10.1 IMS T801 link data and acknowledge packets

SYMBOL	PARAMETER		NOM	MAX	UNITS	NOTE
TJQr	LinkOut rise time			20	ns	1
TJQf	LinkOut fall time		1	10	ns	1
TJDr	LinkIn rise time			20	ns	1
TJDf	LinkIn fall time		1	20	ns	1
TJQJD	Buffered edge delay	0	1		ns	
TJBskew	Variation in TJQJD 20 Mbits	s		3	ns	2
1	10 Mbits	s		10	ns	2
CLIZ	LinkIn capacitance @ f=1MH	z		7	pF	1
CLL	LinkOut load capacitance			50	pF	
RM	Series resistor for 100Ω transmission line		56		ohms	

Table 10.2 Link

- 1 These parameters are sampled, but are not 100% tested.
- 2 This is the variation in the total delay through buffers, transmission lines, differential receivers etc., caused by such things as short term variation in supply voltages and differences in delays for rising and falling edges.

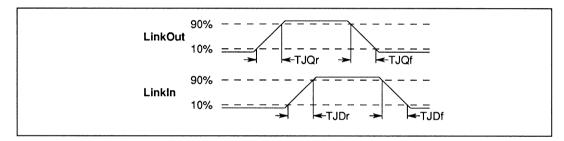


Figure 10.2 IMS T801 link timing

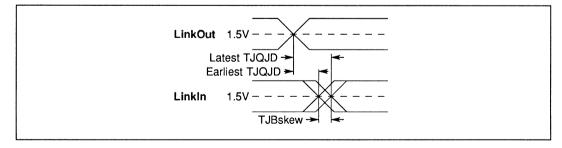


Figure 10.3 IMS T801 buffered link timing

**10 Links** 173

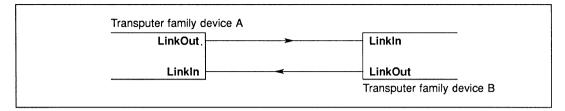


Figure 10.4 IMS T801 Links directly connected

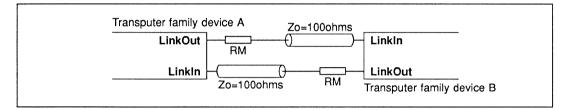


Figure 10.5 IMS T801 Links connected by transmission line

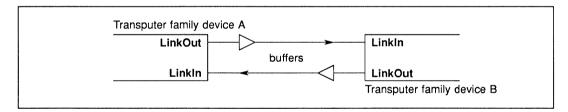


Figure 10.6 IMS T801 Links connected by buffers

# 11 Electrical specifications

#### 11.1 DC electrical characteristics

Table 11.1 Absolute maximum ratings

SYMBOL	PARAMETER	MIN	MAX	UNITS	NOTE
VCC	DC supply voltage	0	7.0	٧	1,2,3
VI, VO	Voltage on input and output pins	-0.5	VCC+0.5	V	1,2,3
II	Input current		±25	mA	4
OSCT	Output short circuit time (one pin)		1	S	2
TS	Storage temperature	-65	150	°C	2
TA	Ambient temperature under bias	-55	125	°C	2
PDmax	Maximum allowable dissipation		2	W	

#### Notes

- 1 All voltages are with respect to GND.
- 2 This is a stress rating only and functional operation of the device at these or any other conditions beyond those indicated in the operating sections of this specification is not implied. Stresses greater than those listed may cause permanent damage to the device. Exposure to absolute maximum rating conditions for extended periods may affect reliability.
- 3 This device contains circuitry to protect the inputs against damage caused by high static voltages or electrical fields. However, it is advised that normal precautions be taken to avoid application of any voltage higher than the absolute maximum rated voltages to this high impedance circuit. Unused inputs should be tied to an appropriate logic level such as VCC or GND.
- 4 The input current applies to any input or output pin and applies when the voltage on the pin is between GND and VCC.

Table 11.2 Operating conditions

SYMBOL	PARAMETER	MIN	MAX	UNITS	NOTE
VCC	DC supply voltage	4.75	5.25	٧	1
VI, VO	Input or output voltage	0	VCC	V	1,2
CL	Load capacitance on any pin		60	pF	
TA	Operating temperature range	0	70	°C	3

- 1 All voltages are with respect to GND.
- 2 Excursions beyond the supplies are permitted but not recommended; see DC characteristics.
- 3 Air flow rate 400 linear ft/min transverse air flow.

SYMBOL	PARAME	TER	MIN	MAX	UNITS	NOTE
VIH	High level input voltage		2.0	VCC+0.5	٧	1,2
VIL	Low level input voltage		-0.5	0.8	٧	1,2
11	Input current	@ GND <vi<vcc< td=""><td></td><td>±10</td><td>μΑ</td><td>1,2</td></vi<vcc<>		±10	μΑ	1,2
VOH	Output high voltage	@ IOH=2mA	VCC-1		٧	1,2
VOL	Output low voltage	@ IOL=4mA		0.4	٧	1,2
IOS	Output short circuit current	@ GND <vo<vcc< td=""><td>36</td><td>65</td><td>mA</td><td>1,2,3,6  </td></vo<vcc<>	36	65	mA	1,2,3,6
	•		65	100	mA	1,2,4,6
IOZ	Tristate output current	@ GND <vo<vcc< td=""><td></td><td>±10</td><td>μΑ</td><td>1,2</td></vo<vcc<>		±10	μΑ	1,2
PD	Power dissipation	_		1.2	W	2,5
CIN	Input capacitance	@ f=1MHz		7	pF	6
COZ	Output capacitance	@ f=1MHz		10	pF	6

Table 11.3 DC characteristics

#### Notes

- 1 All voltages are with respect to GND.
- 2 Parameters for IMS T801-S measured at 4.75V<VCC<5.25V and 0°C<TA<70°C. Input clock frequency = 5 MHz.
- 3 Current sourced from non-link outputs excluding ProcClockOut.
- 4 Current sourced from link outputs and ProcClockOut.
- 5 Power dissipation varies with output loading and program execution. Power dissipation for processor operating at 20 MHz.
- 6 This parameter is sampled and not 100% tested.

# 11.2 Equivalent circuits

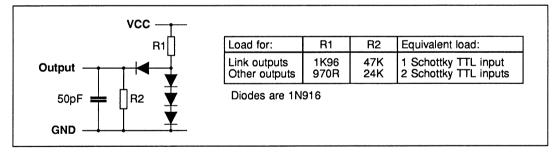


Figure 11.1 Load circuit for AC measurements

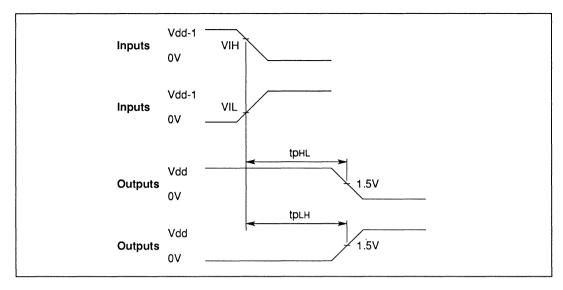


Figure 11.2 AC measurements timing waveforms

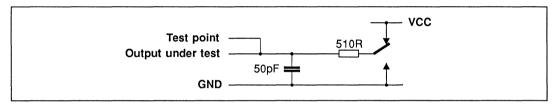


Figure 11.3 Tristate load circuit for AC measurements

# 11.3 AC timing characteristics

Table 11.4 Input, output edges

SYMBOL	PARAMETER	MIN	MAX	UNITS	NOTE
TDr	Input rising edges	2	20	ns	1,2,3
TDf	Input falling edges	2	20	ns	1,2,3
TQr	Output rising edges		25	ns	1,4
TQf	Output falling edges		15	ns	1,4

- 1 Non-link pins; see section on links.
- 2 All inputs except Clockin; see section on Clockin.
- 3 These parameters are not tested.
- 4 These parameters are sampled, but are not 100% tested.

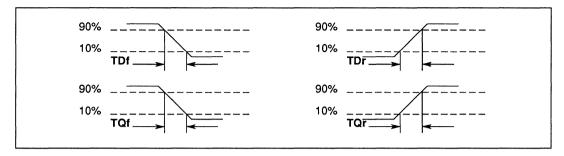


Figure 11.4 IMS T801 input and output edge timing

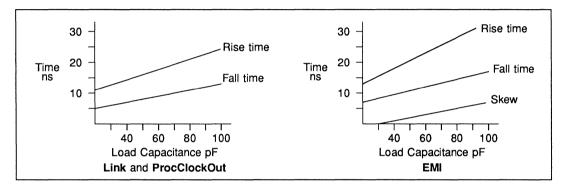


Figure 11.5 Typical rise/fall times

# Notes

1 Skew is measured between notMemCE with a standard load (2 Schottky TTL inputs and 30 pF) and notMemCE with a load of 2 Schottky TTL inputs and varying capacitance.

#### 11.4 Power rating

Internal power dissipation  $P_{INT}$  of transputer and peripheral chips depends on VCC, as shown in figure 11.6.  $P_{INT}$  is substantially independent of temperature.

Total power dissipation  $P_D$  of the chip is

$$P_D = P_{INT} + P_{IO}$$

where  $P_{IO}$  is the power dissipation in the input and output pins; this is application dependent.

Internal working temperature  $T_J$  of the chip is

$$T_J = T_A + \theta J_A * P_D$$

where  $T_A$  is the external ambient temperature in °C and  $\theta J_A$  is the junction-to-ambient thermal resistance in °C/W.  $\theta J_A$  for each package is given in the Packaging Specifications section.

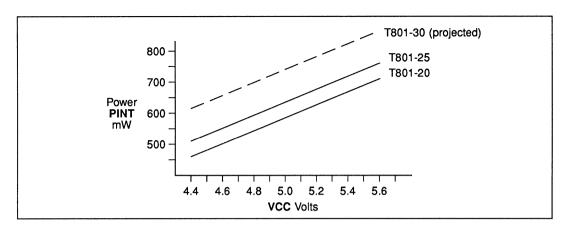


Figure 11.6 IMS T801 internal power dissipation vs VCC

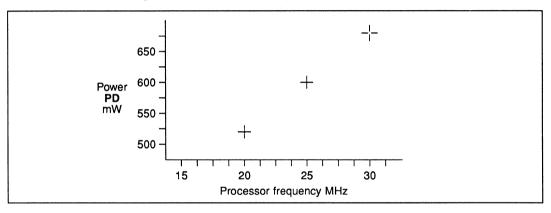


Figure 11.7 IMS T801 typical power dissipation with processor speed

## 12 Performance

The performance of the transputer is measured in terms of the number of bytes required for the program, and the number of (internal) processor cycles required to execute the program. The figures here relate to OCCam programs. For the same function, other languages should achieve approximately the same performance as OCCam.

With transputers incorporating an FPU, this type of performance calculation is straight forward when considering only integer data types. However, when floating point calculations using the **REAL32** and **REAL64** data types are present in the program, complications arise due to the concurrency inherent in the transputer's design whereby integer calculations can be overlapped with floating point calculations. A more comprehensive guide to the impact of this concurrency on transputer performance can be found in the *Transputer Instruction Set - A Compiler Writers' Guide*.

#### 12.1 Performance overview

These figures are averages obtained from detailed simulation, and should be used only as an initial guide; they assume operands are of type **INT**. The abbreviations in table 12.1 are used to represent the quantities indicated. In the replicator section of the table, figures in braces {} are not necessary if the number of replications is a compile time constant. To estimate performance, add together the time for the variable references and the time for the operation.

Table 12.1 Key to performance table

np	number of component processes
ne	number of processes earlier in queue
r	1 if INT parameter or array parameter, 0 if not
ts	number of table entries (table size)
w	width of constant in nibbles
р	number of places to shift
Eg	expression used in a guard
Et	timer expression used in a guard
Tb	most significant bit set of multiplier ((-1) if the multiplier is 0)
Tbp	most significant bit set in a positive multiplier when counting from zero ((-1) if the multiplier is 0)
Tbc	most significant bit set in the two's complement of a negative multiplier
nsp	Number of scalar parameters in a procedure
nap	Number of array parameters in a procedure

Table 12.2 Performance

	Size (bytes)	Time (cycles)
Names		
variables		
in expression	1.1+r	2.1+2(r)
assigned to or input to	1.1+r	1.1+(r)
in PROC or FUNCTION call,		,
corresponding to an INT parameter	1.1+r	1.1+(r)
channels	1.1	2.1
Array Variables (for single dimension arrays)		
constant subscript	0	0
variable subscript	5.3	7.3
expression subscript	5.3	7.3
Declarations		
CHAN OF protocol	3.1	3.1
[size]CHAN OF protocol	9.4	2.2 + 20.2* <b>size</b>
PROC	body+2	0
Primitives		-
assignment	0	0
input	4	26.5
	1 1	26.5
output STOP	2	
	0	25
SKIP	0	0
Arithmetic operators		
+ -	1	1
*	2	39
/	2	40
REM	2	38
<b>&gt;&gt; &lt;&lt;</b>	2	3+ <b>p</b>
Modulo Arithmetic operators		
PLUS	2	2
MINUS	1 1	1 1
TIMES (fast multiply, positive operand)	l i	4+Tbp
TIMES (fast multiply, negative operand)	1 1	5+Tbc
	·	51135
Boolean operators OR	1 4	
AND NOT	4	8 2
	'	
Comparison operators		
= constant	0	1
= variable	2	3
<> constant	1 1	3
<> variable	3	5
> <	1 1	2
>= <=	2	4
Bit operators		
/\`\/ <b>&gt;&lt;</b> ~	2	2
Expressions		
constant in expression	w	w
check if error	"4	6
OHOUR II OHUI		1 0

12 Performance 181

Table 12.3 Performance

	Size (bytes)	Time (cycles)
Timers		
timer input	2	3
timer AFTER		
if past time	) 2	4
with empty timer queue	2 2 2	31
non-empty timer queue	2	38+ <b>ne</b> *9
ALT (timer)		
with empty timer queue	6	52
non-empty timer queue	6	59+ <b>ne</b> *9
timer alt guard	8+2 <b>Eg</b> +2 <b>Et</b>	34+2Eg+2Et
Constructs		
SEQ	0	0
IF	1.3	1.4
if guard	3	4.3
ALT (non timer)	6	26
alt channel guard	10.2+2 <b>Eg</b>	20+2 <b>Eg</b>
skip alt guard	_8+2 <b>Eg</b>	10+2 <b>Eg</b>
PAR	11.5+( <b>np</b> -1)*7.5	19.5+( <b>np</b> -1)*30.5
WHILE	4	12
Procedure or function call		
	3.5+(nsp-2)*1.1	16.5+(nsp-2)*1.1
	+nap∗2.3	+nap*2.3
Replicators		
replicated SEQ	7.3{+5.1}	(-3.8)+15.1*count{+7.1}
replicated IF	12.3{+5.1}	(-2.6)+19.4*count{+7.1}
replicated ALT	24.8{+10.2}	25.4+33.4*count{+14.2}
replicated timer ALT	24.8{+10.2}	62.4+33.4*count(+14.2)
replicated PAR	39.1(+5.1)	(-6.4)+70.9∗count(+7.1)

# 12.2 Fast multiply, TIMES

The IMS T801 has a fast integer multiplication instruction *product*. For a positive multiplier its execution time is 4+**Tbp** cycles, and for a negative multiplier 5+**Tbc** cycles (table 12.1). The time taken for a multiplication by zero is 3 cycles.

Implementations of high level languages on the transputer may take advantage of this instruction. For example, the OCCAM modulo arithmetic operator **TIMES** is implemented by the instruction and the right-hand operand is treated as the multiplier. The fast multiplication instruction is also used in high level language implementations for the multiplication implicit in multi-dimensional array access.

# 12.3 Arithmetic

A set of functions are provided within the development system to support the efficient implementation of multiple length integer arithmetic. In the IMS T801, floating point arithmetic is taken care of by the FPU. In table 12.4 n gives the number of places shifted and all arguments and results are assumed to be local. Full details of these functions are provided in the OCCAM reference manual, supplied as part of the development system and available as a separate publication.

When calculating the execution time of the predefined maths functions, no time needs to be added for calling overhead. These functions are compiled directly into special purpose instructions which are designed to support the efficient implementation of multiple length integer arithmetic and floating point arithmetic.

Function		Cycles	+ cycles for parameter access †
LONGADD		2	7
LONGSUM		3	8
LONGSUB		2	7
LONGDIFF		3	8
LONGPROD		34	8
LONGDIV		36	8
SHIFTRIGHT	(n<32)	4+ <b>n</b>	8
	(n>=32)	<b>n-</b> 27	
SHIFTLEFT	(n<32)	4+ <b>n</b>	8
	(n>=32)	<b>n-</b> 27	
NORMALISE	(n<32)	n+6	7
	( <b>n</b> >=32)	n-25	
	(n=64)	4	
ASHIFTRIGHT		SHIFTRIGHT+2	5
ASHIFTLEFT		SHIFTLEFT+4	5
ROTATERIGHT		SHIFTRIGHT	7
ROTATELEFT		SHIFTLEFT	7
FRACMUL		LONGPROD+4	5

Table 12.4 Arithmetic performance

<sup>†</sup> Assuming local variables.

12 Performance 183

# 12.4 Floating point operations

All references to **REAL32** or **REAL64** operands within programs compiled for the IMS T801 normally produce the following performance figures.

	Size (bytes)	REAL32 Time (cycles)	REAL64 Time (cycles)
Names			
variables			
in expression	3.1	3 3	5 5
assigned to or input to	3.1	3	5
in PROC or FUNCTION call,			
corresponding to a REAL			
parameter	1.1+r	1.1+r	1.1+r
Arithmetic operators			
+ -	2	7	7
*	2	11	20
/	2 2 2 11	17	32
REM	11	19	34
Comparison operators			
<b>=</b>	2	4	4
<>	2 3 2 3	6	6
> <	2	6 5 7	6 5 7
>= <=	3	7	7
Conversions			
REAL32 to -	2		3
REAL64 to -	2	6	
To INT32 from -	2 2 5 18 3	9	9
To INT64 from -	18	32	32
INT32 to -	3	7	7
INT64 to -	14	24	22

Table 12.5 Floating point performance

# 12.4.1 Floating point functions

These functions are provided by the development system. They are compiled directly into special purpose instructions designed to support the efficient implementation of some of the common mathematical functions of other languages. The functions provide ABS and SQRT for both REAL32 and REAL64 operand types.

		+ cycles for parameter access †		
<b>Function</b>	Cycles	REAL32	REAL 64	
ABS	2	8		
SQRT	118	8		
DABS	2		12	
DSQRT	244		12	

Table 12.6 IMS T801 floating point arithmetic performance

<sup>†</sup> Assuming local variables.

#### 12.4.2 Special purpose functions and procedures

The functions and procedures given in tables 12.8 and 12.9 are provided by the development system to give access to the special instructions available on the IMS T801. Table 12.7 shows the key to the table.

Table 12.7 Key to special performance table

Tb	most significant bit set in the word counting from zero
n	number of words per row (consecutive memory locations)
r	number of rows in the two dimensional move
nr	number of bits to reverse

Table 12.8 Special purpose functions performance

Function	Cycles	+ cycles for parameter access †
BITCOUNT	2+ <b>Tb</b>	2
CRCBYTE	11	8
CRCWORD	35	8
BITREVNBIT	5+nr	4
BITREVWORD	36	2

t Assuming local variables.

Table 12.9 Special purpose procedures performance

Procedure	Cycles	+ cycles for parameter access †
MOVE 2D	8+(2 <b>n</b> +23)* <b>r</b>	8
DRAW2D	8+(2n+23)*r	8
CLIP2D	8+(2 <b>n</b> +23)* <b>r</b>	8

† Assuming local variables.

# 12.5 Effect of external memory

Extra processor cycles may be needed when program and/or data are held in external memory, depending both on the operation being performed, and on the speed of the external memory. After a processor cycle which initiates a write to memory, the processor continues execution at full speed until at least the next memory access.

Whilst a reasonable estimate may be made of the effect of external memory, the actual performance will depend upon the exact nature of the given sequence of operations.

External memory is characterized by the number of extra processor cycles per external memory cycle, denoted as **e**. The value of **e** for the IMS T801 is greater than or equal to 1.

If a program is stored in external memory, and **e** has the value 2 or 3, then no extra cycles need be estimated for linear code sequences. For larger values of **e**, the number of extra cycles required for linear code sequences may be estimated at (**e**-3)/4. A transfer of control may be estimated as requiring **e**+3 cycles.

These estimates may be refined for various constructs. In table 12.10  $\bf n$  denotes the number of components in a construct. In the case of  ${\bf IF}$ , the  $\bf n$ 'th conditional is the first to evaluate to  ${\bf TRUE}$ , and the costs include the costs of the conditionals tested. The number of bytes in an array assignment or communication is denoted by  $\bf b$ .

	IMS T	801
	Program off chip	Data off chip
Boolean expressions	<b>e</b> -2	0
IF	3 <b>en</b> -8	en
Replicated IF	(6 <b>e</b> -4) <b>n</b> +7	(5 <b>e</b> -2) <b>n</b> +8
Replicated SEQ	(3 <b>e</b> -3) <b>n</b> +2	(4e-2)n
PAR	(3 <b>e</b> -1) <b>n</b> +8	3 <b>en</b> +4
Replicated PAR	(10 <b>e</b> -8) <b>n</b> +8	16en-12
ALT	(2 <b>e</b> -4)n+6 <b>e</b>	(2e-2)n+10e-8
Array assignment and communication in one transputer	0	max (2e, e(b/2))

Table 12.10 External memory performance

The following simulation results illustrate the effect of storing program and/or data in external memory. The results are normalized to 1 for both program and data on chip. The first program (Sieve of Erastosthenes) is an extreme case as it is dominated by small, data access intensive loops; it contains no concurrency, communication, or even multiplication or division. The second program is the pipeline algorithm for Newton Raphson square root computation.

	Program	e=2	e=3	e=4	e=5	On chip
Program off chip	1	1.3	1.5	1.7	1.9	1
•	2	1.1	1.2	1.2	1.3	1
Data off chip	1	1.5	1.8	2.1	2.3	1
·	2	1.2	1.4	1.6	1.7	1
Program and data off chip	1	1.8	2.2	2.7	3.2	1
,	2	1.3	1.6	1.8	2.0	1

Table 12.11 IMS T801 external memory performance

# 12.6 Interrupt latency

If the process is a high priority one and no other high priority process is running, the latency is as described in table 12.12. The timings given are in full processor cycles **TPCLPCL**; the number of **Tm** states is also given where relevant. Maximum latency assumes all memory accesses are internal ones.

	Тур	ical	Maximum		
	TPCLPCL	Tm	TPCLPCL	Tm	
IMS T801 with FPU in use	19	38	78	156	
IMS T801 with FPU not in use	19	38	58	116	

Table 12.12 Interrupt latency

# 13 Package specifications

# 13.1 100 pin grid array package

一("	ndex 1	2	3	4	5	6	7	8	9	10
A	A21	A23	A25	A26	A30	A31	D2	D5	D6	D13
В	A5	A9	A11	A24	A29	GND	D3	D7	vcc	D14
c	A4	A6	A8	A22	A10	DO	D4	D9	D12	D16
D	GND	A2	A3	A7	A27	D1	D8	D10	D15	D17
E	A17	A19	A18	A20	A28	D11	D18	D19	D20	D21
F	A16	A15	A14	A13	Reset	Error Out	D24	GND	D23	D22
G	A12	not Mem WrB2	not Mem WrB0	GND	Link In1	Link Speed	D31	D27	D26	D25
н	not Mem WrB3	Mem Wait	Mem Req	Link Out3	Link In0	Proc Clock Out	GND	Proc Speed1	D30	D28
J	not Mem WrB1	Mem Granted	Event Req	Link In2	Link Out1	Event Waiting	Clockin	Boot From Rom	Analyse	D29
ĸ	not Mem CE	Event Ack	Link In3	Link Out2	Link Out0	vcc	Cap Plus	Cap Minus	Proc Speed0	Proc Speed2

Figure 13.1 IMS T801 100 pin grid array package pinout - top view

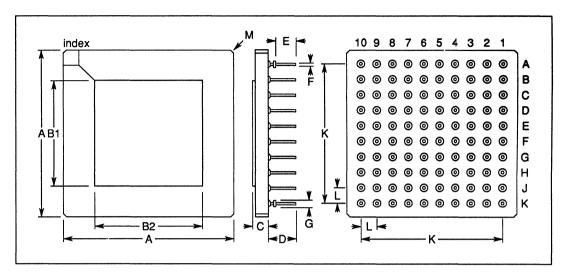


Figure 13.2 100 pin grid array package dimensions

Table 13.1	100 pin	grid array	package	dimensions
------------	---------	------------	---------	------------

	Millim	etres	Inc	hes	
DIM	NOM	TOL	NOM	TOL	Notes
Α	26.924	±0.254	1.060	±0.010	
B1	17.019	±0.127	0.670	±0.005	
B2	18.796	±0.127	0.740	±0.005	
С	2.456	±0.278	0.097	±0.011	
D	4.572	±0.127	0.180	±0.005	
E	3.302	±0.127	0.130	±0.005	
F	0.457	±0.051	0.018	±0.002	Pin diameter
G	1.143	±0.127	0.045	±0.005	Flange diameter
K	22.860	±0.127	0.900	±0.005	
L	2.540	±0.127	0.100	±0.005	
М	0.508		0.020		Chamfer

Table 13.2 100 pin grid array package junction to ambient thermal resistance

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTE
θJA	At 400 linear ft/min transverse air flow			35	°C/W	

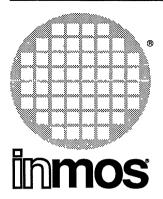
# 14 Ordering

This section indicates the designation of speed and package selections for the various devices. Speed of **ClockIn** is 5 MHz for all parts. Transputer processor cycle time is nominal; it can be calculated more exactly using the phase lock loop factor **PLLx**, as detailed in the external memory section.

For availability contact local INMOS sales office or authorised distributor.

Table 14.1 IMS T801 ordering details

INMOS designation	Processor clock speed	Processor cycle time	PLLx	Package
IMS T801-G20S	20.0 MHz	50 ns	4.0	Ceramic Pin Grid
IMS T801-G25S	25.0 MHz	40 ns	5.0	Ceramic Pin Grid
IMS T801-G30S	30.0 MHz	33 ns	6.0	Ceramic Pin Grid
IMS T801-G20M	20.0 MHz	50 ns	4.0	Ceramic Pin Grid



# IMS T800 transputer

# **Engineering Data**

# **FEATURES**

32 bit architecture

33 ns internal cycle time

30 MIPS (peak) instruction rate

4.3 Mflops (peak) instruction rate

64 bit on-chip floating point unit which conforms to IEEE 754

4 Kbytes on-chip static RAM

120 Mbytes/sec sustained data rate to internal memory

4 Gbytes directly addressable external memory

40 Mbytes/sec sustained data rate to external memory 630 ns response to interrupts

Four INMOS serial links 5/10/20 Mbits/sec

Bi-directional data rate of 2.4 Mbytes/sec per link High performance graphics support with block move

instructions

Boot from ROM or communication links

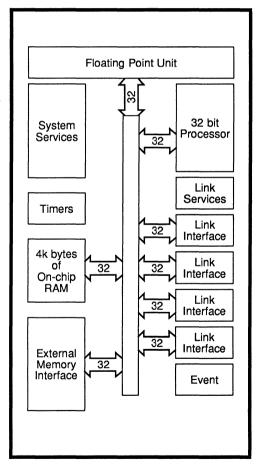
Single 5 MHz clock input

Single +5V ±5% power supply

MIL-STD-883C processing is available

# **APPLICATIONS**

Scientific and mathematical applications
High speed multi processor systems
High performance graphics processing
Supercomputers
Workstations and workstation clusters
Digital signal processing
Accelerator processors
Distributed databases
System simulation
Telecommunications
Robotics
Fault tolerant systems
Image processing
Pattern recognition
Artificial intelligence



42 1406 03 May 1989

#### 1 Introduction

The IMS T800 transputer is a 32 bit CMOS microcomputer with a 64 bit floating point unit and graphics support. It has 4 Kbytes on-chip RAM for high speed processing, a configurable memory interface and four standard INMOS communication links. The instruction set achieves efficient implementation of high level languages and provides direct support for the OCCAM model of concurrency when using either a single transputer or a network. Procedure calls, process switching and typical interrupt latency are sub-microsecond.

For convenience of description, the IMS T800 operation is split into the basic blocks shown in figure 1.1.

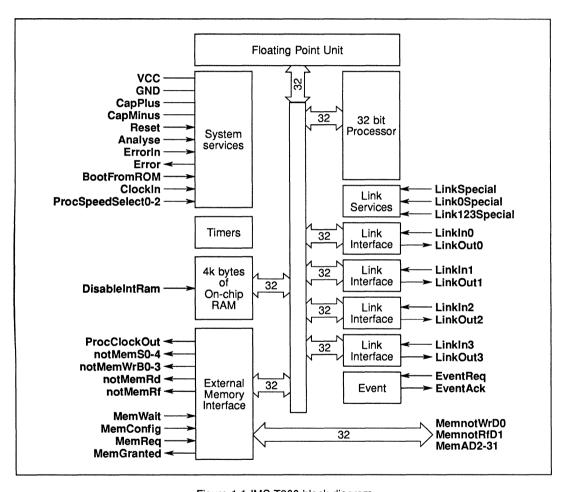


Figure 1.1 IMS T800 block diagram

The processor speed of a device can be pin-selected in stages from 17.5 MHz up to the maximum allowed for the part. A device running at 30 MHz achieves an instruction throughput of 30 MIPS peak and 15 MIPS sustained. The extended temperature version of the device complies with MIL-STD-883C.

The IMS T800 provides high performance arithmetic and floating point operations. The 64 bit floating point unit provides single and double length operation to the ANSI-IEEE 754-1985 standard for floating point arithmetic. It is able to perform floating point operations concurrently with the processor, sustaining a rate of 2.2 Mflops at a processor speed of 20 MHz and 3.3 Mflops at 30 MHz.

1 Introduction 191

High performance graphics support is provided by microcoded block move instructions which operate at the speed of memory. The two-dimensional block move instructions provide for contiguous block moves as well as block copying of either non-zero bytes of data only or zero bytes only. Block move instructions can be used to provide graphics operations such as text manipulation, windowing, panning, scrolling and screen updating.

Cyclic redundancy checking (CRC) instructions are available for use on arbitrary length serial data streams, to provide error detection where data integrity is critical. Another feature of the IMS T800, useful for pattern recognition, is the facility to count bits set in a word.

The IMS T800 can directly access a linear address space of 4 Gbytes. The 32 bit wide memory interface uses multiplexed data and address lines and provides a data rate of up to 4 bytes every 100 nanoseconds (40 Mbytes/sec) for a 30 MHz device. A configurable memory controller provides all timing, control and DRAM refresh signals for a wide variety of mixed memory systems.

System Services include processor reset and bootstrap control, together with facilities for error analysis. Error signals may be daisy-chained in multi-transputer systems.

The standard INMOS communication links allow networks of transputer family products to be constructed by direct point to point connections with no external logic. The IMS T800 links support the standard operating speed of 10 Mbits/sec, but also operate at 5 or 20 Mbits/sec. Each link can transfer data bi-directionally at up to 2.35 Mbytes/sec.

The transputer is designed to implement the OCCAM language, detailed in the OCCAM Reference Manual, but also efficiently supports other languages such as C, Pascal and Fortran. Access to the transputer at machine level is seldom required, but if necessary refer to the *Transputer Instruction Set - A Compiler Writers' Guide*.

This data sheet supplies hardware implementation and characterisation details for the IMS T800. It is intended to be read in conjunction with the Transputer Architecture chapter, which details the architecture of the transputer and gives an overview of occam.

# 2 Pin designations

Table 2.1 IMS T800 system services

Pin	In/Out	Function
VCC, GND		Power supply and return
CapPlus, CapMinus		External capacitor for internal clock power supply
Clockin	in	Input clock
ProcSpeedSelect0-2	l in	Processor speed selectors
Reset	in	System reset
Error	out	Error indicator
Errorin	in	Error daisychain input
Analyse	in	Error analysis
BootFromRom	in	Boot from external ROM or from link
DisableIntRAM	in	Disable internal RAM
DoNotWire		Must not be wired

Table 2.2 IMS T800 external memory interface

Pin	In/Out	Function
ProcClockOut	out	Processor clock
MemnotWrD0	in/out	Multiplexed data bit 0 and write cycle warning
MemnotRfD1	in/out	Multiplexed data bit 1 and refresh warning
MemAD2-31	in/out	Multiplexed data and address bus
notMemRd	out	Read strobe
notMemWrB0-3	out	Four byte-addressing write strobes
notMemS0-4	out	Five general purpose strobes
notMemRf	out	Dynamic memory refresh indicator
MemWait	in	Memory cycle extender
MemReq	in	Direct memory access request
MemGranted	out	Direct memory access granted
MemConfig	in	Memory configuration data input

Table 2.3 IMS T800 event

Pin	In/Out	Function
EventReq	in	Event request
EventAck	out	Event request acknowledge

Table 2.4 IMS T800 link

Pin	In/Out	Function
Linkln0-3	in	Four serial data input channels
LinkOut0-3	out	Four serial data output channels
LinkSpecial	in	Select non-standard speed as 5 or 20 Mbits/sec
Link0Special	in	Select special speed for Link 0
Link123Special	in	Select special speed for Links 1,2,3

Signal names are prefixed by **not** if they are active low, otherwise they are active high. Pinout details for various packages are given on page 256.

## 3 Processor

The 32 bit processor contains instruction processing logic, instruction and work pointers, and an operand register. It directly accesses the high speed 4 Kbyte on-chip memory, which can store data or program. Where larger amounts of memory or programs in ROM are required, the processor has access to 4 Gbytes of memory via the External Memory Interface (EMI).

# 3.1 Registers

The design of the transputer processor exploits the availability of fast on-chip memory by having only a small number of registers; six registers are used in the execution of a sequential process. The small number of registers, together with the simplicity of the instruction set, enables the processor to have relatively simple (and fast) data-paths and control logic. The six registers are:

The workspace pointer which points to an area of store where local variables are kept.

The instruction pointer which points to the next instruction to be executed.

The operand register which is used in the formation of instruction operands.

The A, B and C registers which form an evaluation stack.

A, B and C are sources and destinations for most arithmetic and logical operations. Loading a value into the stack pushes B into C, and A into B, before loading A. Storing a value from A, pops B into A and C into B.

Expressions are evaluated on the evaluation stack, and instructions refer to the stack implicitly. For example, the *add* instruction adds the top two values in the stack and places the result on the top of the stack. The use of a stack removes the need for instructions to respecify the location of their operands. Statistics gathered from a large number of programs show that three registers provide an effective balance between code compactness and implementation complexity.

No hardware mechanism is provided to detect that more than three values have been loaded onto the stack. It is easy for the compiler to ensure that this never happens.

Any location in memory can be accessed relative to the workpointer register, enabling the workspace to be of any size.

Further register details are given in Transputer Instruction Set - A Compiler Writers' Guide.

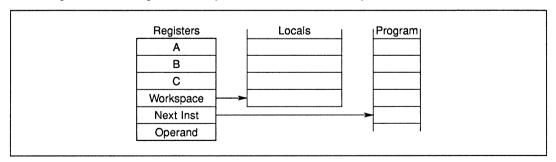


Figure 3.1 Registers

#### 3.2 Instructions

The instruction set has been designed for simple and efficient compilation of high-level languages. All instructions have the same format, designed to give a compact representation of the operations occurring most frequently in programs.

Each instruction consists of a single byte divided into two 4-bit parts. The four most significant bits of the byte are a function code and the four least significant bits are a data value.

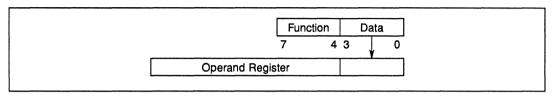


Figure 3.2 Instruction format

# 3.2.1 Direct functions

The representation provides for sixteen functions, each with a data value ranging from 0 to 15. Ten of these, shown in table 3.1, are used to encode the most important functions.

load constant	add constant	
load local	store local	load local pointer
load non-local	store non-local	
jump	conditional jump	call

Table 3.1 Direct functions

The most common operations in a program are the loading of small literal values and the loading and storing of one of a small number of variables. The *load constant* instruction enables values between 0 and 15 to be loaded with a single byte instruction. The *load local* and *store local* instructions access locations in memory relative to the workspace pointer. The first 16 locations can be accessed using a single byte instruction.

The *load non-local* and *store non-local* instructions behave similarly, except that they access locations in memory relative to the *A* register. Compact sequences of these instructions allow efficient access to data structures, and provide for simple implementations of the static links or displays used in the implementation of high level programming languages such as OCCam, C, Fortran, Pascal or ADA.

## 3.2.2 Prefix functions

Two more function codes allow the operand of any instruction to be extended in length; *prefix* and *negative prefix*.

All instructions are executed by loading the four data bits into the least significant four bits of the operand register, which is then used as the instruction's operand. All instructions except the prefix instructions end by clearing the operand register, ready for the next instruction.

The *prefix* instruction loads its four data bits into the operand register and then shifts the operand register up four places. The *negative prefix* instruction is similar, except that it complements the operand register before shifting it up. Consequently operands can be extended to any length up to the length of the operand register by a sequence of prefix instructions. In particular, operands in the range -256 to 255 can be represented using one prefix instruction.

3 Processor 195

The use of prefix instructions has certain beneficial consequences. Firstly, they are decoded and executed in the same way as every other instruction, which simplifies and speeds instruction decoding. Secondly, they simplify language compilation by providing a completely uniform way of allowing any instruction to take an operand of any size. Thirdly, they allow operands to be represented in a form independent of the processor wordlength.

## 3.2.3 Indirect functions

The remaining function code, *operate*, causes its operand to be interpreted as an operation on the values held in the evaluation stack. This allows up to 16 such operations to be encoded in a single byte instruction. However, the prefix instructions can be used to extend the operand of an *operate* instruction just like any other. The instruction representation therefore provides for an indefinite number of operations.

Encoding of the indirect functions is chosen so that the most frequently occurring operations are represented without the use of a prefix instruction. These include arithmetic, logical and comparison operations such as add, exclusive or and greater than. Less frequently occurring operations have encodings which require a single prefix operation.

## 3.2.4 Expression evaluation

Evaluation of expressions sometimes requires use of temporary variables in the workspace, but the number of these can be minimised by careful choice of the evaluation order.

Program	Mnem	onic
x := 0	ldc	0
	stl	X
x := #24	pfix Idc	2 4
	ldc	4
	stl	X
x := y + z	Idl	У
	Idl	y z
	add	
	stl	X

Table 3.2 Expression evaluation

## 3.2.5 Efficiency of encoding

Measurements show that about 70% of executed instructions are encoded in a single byte; that is, without the use of prefix instructions. Many of these instructions, such as *load constant* and *add* require just one processor cycle.

The instruction representation gives a more compact representation of high level language programs than more conventional instruction sets. Since a program requires less store to represent it, less of the memory bandwidth is taken up with fetching instructions. Furthermore, as memory is word accessed the processor will receive four instructions for every fetch.

Short instructions also improve the effectiveness of instruction pre-fetch, which in turn improves processor performance. There is an extra word of pre-fetch buffer, so the processor rarely has to wait for an instruction fetch before proceeding. Since the buffer is short, there is little time penalty when a jump instruction causes the buffer contents to be discarded.

## 3.3 Processes and concurrency

A process starts, performs a number of actions, and then either stops without completing or terminates complete. Typically, a process is a sequence of instructions. A transputer can run several processes in parallel (concurrently). Processes may be assigned either high or low priority, and there may be any number of each (page 197).

The processor has a microcoded scheduler which enables any number of concurrent processes to be executed together, sharing the processor time. This removes the need for a software kernel.

At any time, a concurrent process may be

Active - Being executed.

- On a list waiting to be executed.

Inactive - Ready to input.

Ready to output.

- Waiting until a specified time.

The scheduler operates in such a way that inactive processes do not consume any processor time. It allocates a portion of the processor's time to each process in turn. Active processes waiting to be executed are held in two linked lists of process workspaces, one of high priority processes and one of low priority processes (page 197). Each list is implemented using two registers, one of which points to the first process in the list, the other to the last. In the Linked Process List figure 3.3, process S is executing and P, Q and R are active, awaiting execution. Only the low priority process queue registers are shown; the high priority process ones perform in a similar manner.

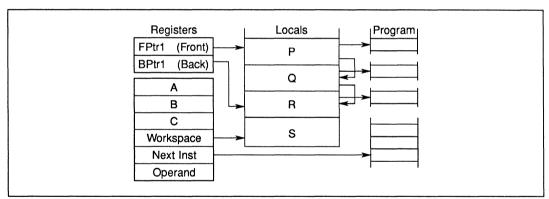


Figure 3.3 Linked process list

Table 3.3 Priority queue control registers

Function	High Priority	Low Priority
Pointer to front of active process list	Fptr0	Fptr1
Pointer to back of active process list	Bptr0	Bptr1

Each process runs until it has completed its action, but is descheduled whilst waiting for communication from another process or transputer, or for a time delay to complete. In order for several processes to operate in parallel, a low priority process is only permitted to run for a maximum of two time slices before it is forcibly descheduled at the next descheduling point (page 201). The time slice period is 5120 cycles of the external 5 MHz clock, giving ticks approximately 1 ms apart.

3 Processor 197

A process can only be descheduled on certain instructions, known as descheduling points (page 201). As a result, an expression evaluation can be guaranteed to execute without the process being timesliced part way through.

Whenever a process is unable to proceed, its instruction pointer is saved in the process workspace and the next process taken from the list. Process scheduling pointers are updated by instructions which cause scheduling operations, and should not be altered directly. Actual process switch times are less than 1  $\mu$ s, as little state needs to be saved and it is not necessary to save the evaluation stack on rescheduling.

The processor provides a number of special operations to support the process model, including *start process* and *end process*. When a main process executes a parallel construct, *start process* instructions are used to create the necessary additional concurrent processes. A *start process* instruction creates a new process by adding a new workspace to the end of the scheduling list, enabling the new concurrent process to be executed together with the ones already being executed. When a process is made active it is always added to the end of the list, and thus cannot pre-empt processes already on the same list.

The correct termination of a parallel construct is assured by use of the *end process* instruction. This uses a workspace location as a counter of the parallel construct components which have still to terminate. The counter is initialised to the number of components before the processes are *started*. Each component ends with an *end process* instruction which decrements and tests the counter. For all but the last component, the counter is non zero and the component is descheduled. For the last component, the counter is zero and the main process continues.

# 3.4 Priority

The IMS T800 supports two levels of priority. Priority 1 (low priority) processes are executed whenever there are no active priority 0 (high priority) processes.

High priority processes are expected to execute for a short time. If one or more high priority processes are able to proceed, then one is selected and runs until it has to wait for a communication, a timer input, or until it completes processing.

If no process at high priority is able to proceed, but one or more processes at low priority are able to proceed, then one is selected.

Low priority processes are periodically timesliced to provide an even distribution of processor time between computationally intensive tasks.

If there are **n** low priority processes, then the maximum latency from the time at which a low priority process becomes active to the time when it starts processing is 2**n**-2 timeslice periods. It is then able to execute for between one and two timeslice periods, less any time taken by high priority processes. This assumes that no process monopolises the transputer's time; i.e. it has a distribution of descheduling points (page 201).

Each timeslice period lasts for 5120 cycles of the external 5 MHz input clock (approximately 1 ms at the standard frequency of 5 MHz).

If a high priority process is waiting for an external channel to become ready, and if no other high priority process is active, then the interrupt latency (from when the channel becomes ready to when the process starts executing) is typically 19 processor cycles, a maximum of 78 cycles (assuming use of on-chip RAM). If the floating point unit is not being used at the time then the maximum interrupt latency is only 58 cycles. To ensure this latency, certain instructions are interruptable.

## 3.5 Communications

Communication between processes is achieved by means of channels. Process communication is point-to-point, synchronised and unbuffered. As a result, a channel needs no process queue, no message queue and no message buffer.

A channel between two processes executing on the same transputer is implemented by a single word in memory; a channel between processes executing on different transputers is implemented by point-to-point links. The processor provides a number of operations to support message passing, the most important being input message and output message.

The *input message* and *output message* instructions use the address of the channel to determine whether the channel is internal or external. Thus the same instruction sequence can be used for both, allowing a process to be written and compiled without knowledge of where its channels are connected.

The process which first becomes ready must wait until the second one is also ready. A process performs an input or output by loading the evaluation stack with a pointer to a message, the address of a channel, and a count of the number of bytes to be transferred, and then executing an *input message* or *output message* instruction. Data is transferred if the other process is ready. If the channel is not ready or is an external one the process will deschedule.

#### 3.6 Block move

The block move on the transputer moves any number of bytes from any byte boundary in memory, to any other byte boundary, using the smallest possible number of word read, and word or part-word writes.

A block move instruction can be interrupted by a high priority process. On interrupt, block move is completed to a word boundary, independent of start position. When restarting after interrupt, the last word written is written again. This appears as an unnecessary read and write in the simplest case of word aligned block moves, and may cause problems with FIFOs. This problem can be overcome by incrementing the saved destination (BregIntSaveLoc) and source pointer (CregIntSaveLoc) values by BytesPerWord during the high priority process.

#### 3.7 Timers

The transputer has two 32 bit timer clocks which 'tick' periodically. The timers provide accurate process timing, allowing processes to deschedule themselves until a specific time.

One timer is accessible only to high priority processes and is incremented every microsecond, cycling completely in approximately 4295 seconds. The other is accessible only to low priority processes and is incremented every 64 microseconds, giving exactly 15625 ticks in one second. It has a full period of approximately 76 hours.

Table 3.4 Timer registers

Clock0	Current value of high priority (level 0) process clock
Clock1	Current value of low priority (level 1) process clock
TNextReg0	Indicates time of earliest event on high priority (level 0) timer queue
TNextReg1	Indicates time of earliest event on low priority (level 1) timer queue

The current value of the processor clock can be read by executing a *load timer* instruction. A process can arrange to perform a *timer input*, in which case it will become ready to execute after a specified time has been reached. The *timer input* instruction requires a time to be specified. If this time is in the 'past' then the instruction has no effect. If the time is in the 'future' then the process is descheduled. When the specified time is reached the process is scheduled again.

3 Processor 199

Figure 3.4 shows two processes waiting on the timer queue, one waiting for time 21, the other for time 31.

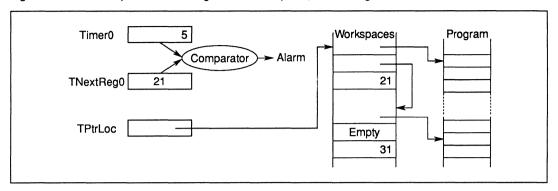


Figure 3.4 Timer registers

.

# 4 Instruction set summary

The Function Codes table 4.8. gives the basic function code set (page 194). Where the operand is less than 16, a single byte encodes the complete instruction. If the operand is greater than 15, one prefix instruction (*pfix*) is required for each additional four bits of the operand. If the operand is negative the first prefix instruction will be *nfix*.

Function Memory **Mnemonic** code code #4 #43 ldc #3 #35 ldc is coded as pfix #3 #2 #23 ldc #5 #4 #45 ldc. #987 is coded as #9 #2 #29 pfix pfix #28 #8 #2 ldc #7 #4 #47 #FFFFFE1) -31 ldc (ldc is coded as #1 #6 #61 nfix ldc #1 #4 #41

Table 4.1 prefix coding

Tables 4.9 to 4.27 give details of the operation codes. Where an operation code is less than 16 (e.g. *add*: operation code **05**), the operation can be stored as a single byte comprising the *operate* function code **F** and the operand (**5** in the example). Where an operation code is greater than 15 (e.g. *ladd*: operation code **16**), the *prefix* function code **2** is used to extend the instruction.

Mnem	onic	Function code	Memory code
add is coded as	(op. code #5)		#F5
opr	add	#F	#F5
ladd is coded as	(op. code #16)		#21F6
pfix	#1	#2	#21
opr	#6	#F	#F6

Table 4.2 operate coding

In the Floating Point Operation Codes tables 4.21 to 4.27, a selector sequence code (page 209) is indicated in the Memory Code column by **s**. The code given in the Operation Code column is the indirection code, the operand for the *ldc* instruction.

The FPU and processor operate concurrently, so the actual throughput of floating point instructions is better than that implied by simply adding up the instruction times. For full details see *Transputer Instruction Set - A Compiler Writers' Guide*.

The Processor Cycles column refers to the number of periods **TPCLPCL** taken by an instruction executing in internal memory. The number of cycles is given for the basic operation only; where the memory code for an instruction is two bytes, the time for the *prefix* function (one cycle) should be added. For a 20 MHz transputer one cycle is 50 ns. Some instruction times vary. Where a letter is included in the cycles column it is interpreted from table 4.3.

Table 4.3 Instruction set interpretation

Ident	Interpretation
b	Bit number of the highest bit set in register A. Bit 0 is the least significant bit.
m	Bit number of the highest bit set in the absolute value of register <i>A</i> .  Bit 0 is the least significant bit.
n	Number of places shifted.
w	Number of words in the message. Part words are counted as full words. If the message is not word aligned the number of words is increased to include the part words at either end of the message.
р	Number of words per row.
r	Number of rows.

The **DE** column of the tables indicates the descheduling/error features of an instruction as described in table 4.4.

Table 4.4 Instruction features

Ident	Feature	See page:
D	The instruction is a descheduling point	201
E	The instruction will affect the Error flag	202, 216
F	The instruction will affect the FP_Error flag	209, 202

# 4.1 Descheduling points

The instructions in table 4.5 are the only ones at which a process may be descheduled (page 196). They are also the ones at which the processor will halt if the **Analyse** pin is asserted (page 215).

Table 4.5 Descheduling point instructions

	input message timer alt wait jump	output message timer input loop end	output byte stop on error end process	output word alt wait stop process	
١	jump	ioop ena	ena process	stop process	

# 4.2 Error instructions

The instructions in table 4.6 are the only ones which can affect the *Error* flag (page 216) directly. Note, however, that the floating point unit error flag *FP\_Error* is set by certain floating point instructions (page 202), and that *Error* can be set from this flag by *fpcheckerror*.

Table 4.6 Error setting instructions

add	add constant	subtract	remainder
multiply	fractional multiply	divide	
long add	long subtract	long divide	
set error	testerr	fpcheckerror	check count from 1
check word	check subscript from 0	check single	

# 4.3 Floating point errors

The instructions in table 4.7 are the only ones which can affect the floating point error flag *FP\_Error* (page 209). *Error* is set from this flag by *fpcheckerror* if *FP\_Error* is set.

Table 4.7 Floating point error setting instructions

fpadd fpldnladdsn fpremfirst fpuseterror	fpsub fpldnladddb fpusqrtfirst fpuclearerror	fpmul fpldnlmulsn fpgt fptesterror	fpdiv fpldnlmuldb fpeq
fpuseterror fpuexpincby32 fpur32tor64 fprtoi32	fpuciearerror fpuexpdecby32 fpur64tor32 fpuabs	iplesiendi fpumulby2 fpucki32 fpint	fpudivby2 fpucki64

Function Code	Memory Code	Mnemonic	Processor Cycles	Name	D E
0	0X	j	3	jump	D
1	1X	ldlp	1	load local pointer	
2	2X	pfix	1	prefix	1
3	3X	Idni	2	load non-local	
4	4X	ldc	1	load constant	
5	5X	Idnlp	1	load non-local pointer	
6	6X	nfix	1	negative prefix	
7	7X	idi	2	load local	
8	8X	adc	1	add constant	E
9	9X	call	7	call	
Α	AX	cj	2	conditional jump (not taken)	1
			4	conditional jump (taken)	
В	BX	ajw	1	adjust workspace	
С	CX	eqc	2	equals constant	
D	DX	stl	1	store local	
Ε	EX	stnl	2	store non-local	
F	FX	opr	-	operate	

Table 4.8 IMS T800 function codes

Table 4.9 IMS T800 arithmetic/logical operation codes

Operation Code	Memory Code	Mnemonic	Processor Cycles	Name	D E
46	24F6	and	1	and	
4B	24FB	or	1	or	
33	23F3	xor	1	exclusive or	
32	23F2	not	1	bitwise not	
41	24F1	shl	n+2	shift left	
40	24F0	shr	n+2	shift right	
05	F5	add	1	add	E
0C	FC	sub	1	subtract	E
53	25F3	mul	38	multiply	E
72	27F2	fmul	35	fractional multiply (no rounding)	E
			40	fractional multiply (rounding)	E
2C	22FC	div	39	divide	E
1F	21FF	rem	37	remainder	E
09	F9	gt	2	greater than	1 1
04	F4	diff	1	difference	
52	25F2	sum	1	sum	
08	F8	prod	b+4	product for positive register A	
			<b>m</b> +5	product for negative register A	

Operation Code	Memory Code	Mnemonic	Processor Cycles	Name	D E
16	21F6	ladd	2	long add	Е
38	23F8	Isub	2	long subtract	E
37	23F7	Isum	3	long sum	
4F	24FF	ldiff	3	long diff	
31	23F1	lmul	33	long multiply	
1A	21FA	ldiv	35	long divide	E
36	23F6	Ishi	n+3	long shift left (n<32)	
			n-28	long shift left(n≥32)	
35	23F5	Ishr	n+3	long shift right (n<32)	
			n-28	long shift right (n≥32)	
19	21F9	norm	n+5	normalise (n<32)	
			n-26	normalise (n≥32)	
			3	normalise (n=64)	

Table 4.10 IMS T800 long arithmetic operation codes

Table 4.11 IMS T800 general operation codes

Operation Code	Memory Code	Mnemonic	Processor Cycles	Name	D E
00	F0	rev	1	reverse	
ЗА	23FA	xword	4	extend to word	1
56	25F6	cword	5	check word	E
1D	21FD	xdble	2	extend to double	
4C	24FC	csngl	3	check single	E
42	24F2	mint	1	minimum integer	
5A	25FA	dup	1	duplicate top of stack	

Table 4.12 IMS T800 2D block move operation codes

Operation Code	Memory Code	Mnemonic	Processor Cycles	Name	D E
5B	25FB	move2dinit	8	initialise data for 2D block move	
5C	25FC	move2dall	(2p+23)*r	2D block copy	
5D	25FD	move2dnonzero	(2p+23)*r	2D block copy non-zero bytes	
5E	25FE	move2dzero	(2 <b>p</b> +23)* <b>r</b>	2D block copy zero bytes	

Table 4.13 IMS T800 CRC and bit operation codes

Operation Code	Memory Code	Mnemonic	Processor Cycles	Name	D E
74	27F4	crcword	35	calculate crc on word	T
75	27F5	crcbyte	11	calculate crc on byte	
76	27F6	bitcnt	<b>b</b> +2	count bits set in word	
77	27F7	bitrevword	36	reverse bits in word	Ì
78	27F8	bitrevnbits	n+4	reverse bottom n bits in word	i

24FA

move

4A

Operation Code	Memory Code	Mnemonic	Processor Cycles	Name	D E
02	F2	bsub	1	byte subscript	
0A	FA	wsub	2	word subscript	
81	28F1	wsubdb	3	form double word subscript	
34	23F4	bcnt	2	byte count	
3F	23FF	wcnt	5	word count	-
01	F1	lb	5	load byte	
3B	23FB	sb	4	store byte	

Table 4.14 IMS T800 indexing/array operation codes

Table 4.15 IMS T800 timer handling operation codes

2w+8

move message

Operation Code	Memory Code	Mnemonic	Processor Cycles	Name	D E
22	22F2	ldtimer	2	load timer	
2B	22FB	tin	30	timer input (time future)	D
			4	timer input (time past)	D
4E	24FE	talt	4	timer alt start	
51	25F1	taltwt	15	timer alt wait (time past)	D
			48	timer alt wait (time future)	D
47	24F7	enbt	8	enable timer	İ
2E	22FE	dist	23	disable timer	

Table 4.16 IMS T800 input/output operation codes

Operation Code	Memory Code	Mnemonic	Processor Cycles	Name	D E
07	F7	in	2w+19	input message	D
0B	FB	out	2 <b>w</b> +19	output message	D
0F	FF	outword	23	output word	D
0E	FE	outbyte	23	output byte	D
43	24F3	alt	2	alt start	
44	24F4	altwt	5	alt wait (channel ready)	D
			17	alt wait (channel not ready)	D
45	24F5	altend	4	alt end	
49	24F9	enbs	3	enable skip	
30	23F0	diss	4	disable skip	
12	21F2	resetch	3	reset channel	
48	24F8	enbc	7	enable channel (ready)	
			5	enable channel (not ready)	
2F	22FF	disc	8	disable channel	

Operation Code	Memory Code	Mnemonic	Processor Cycles	Name	D E
20	22F0	ret	5	return	
1B	21FB	ldpi	2	load pointer to instruction	
3C	23FC	gajw	2	general adjust workspace	
06	F6	gcall	4	general call	
21	21   22F1   lend	lend	10	loop end (loop)	D
			5	loop end (exit)	D

Table 4.17 IMS T800 control operation codes

Table 4.18 IMS T800 scheduling operation codes

Operation Code	Memory Code	Mnemonic	Processor Cycles	Name	D E
0D	FD	startp	12	start process	D
03	F3	endp	13	end process	D
39	23F9	runp	10	run process	1 1
15	21F5	stopp	11	stop process	
1E	21FE	ldpri	1	load current priority	

Table 4.19 IMS T800 error handling operation codes

Operation Code	Memory Code	Mnemonic	Processor Cycles	Name	D E
13	21F3	csub0	2	check subscript from 0	E
4D	24FD	ccnt1	3	check count from 1	E
29	22F9	testerr	2	test error false and clear (no error)	
			3	test error false and clear (error)	
10	21F0	seterr	1	set error	E
55	25F5	stoperr	2	stop on error (no error)	D
57	25F7	cirhaiterr	1	clear halt-on-error	
58	25F8	sethalterr	1	set halt-on-error	
59	25F9	testhalterr	2	test halt-on-error	

Table 4.20 IMS T800 processor initialisation operation codes

Operation Code	Memory Code	Mnemonic	Processor Cycles	Name	D E
2A	22FA	testpranal	2	test processor analysing	
3E	23FE	saveh	4	save high priority queue registers	
3D	23FD	savel	4	save low priority queue registers	
18	21F8	sthf	1	store high priority front pointer	
50	25F0	sthb	1	store high priority back pointer	
1C	21FC	stlf	1	store low priority front pointer	
17	21F7	stlb	1	store low priority back pointer	
54	25F4	sttimer	1	store timer	

Operation Code	Memory Code	Mnemonic	Processor Cycles	Name	D E
8E	28FE	fpldnlsn	2	fp load non-local single	
8 <b>A</b>	28FA	fpldnldb	3	fp load non-local double	
86	28F6	fpldnlsni	4	fp load non-local indexed single	
82	28F2	fpldnldbi	6	fp load non-local indexed double	
9F	29FF	fpldzerosn	2	load zero single	İ
A0	2AF0	fpldzerodb	2	load zero double	
AA	2AFA	fpldnladdsn	8/11	fp load non local & add single	F
A6	2AF6	fpldnladddb	9/12	fp load non local & add double	F
AC	2AFC	fpldnlmulsn	13/20	fp load non local & multiply single	F
A8	2AF8	fpldnlmuldb	21/30	fp load non local & multiply double	F
88	28F8	fpstnlsn	2	fp store non-local single	
84	28F4	fpstnldb	3	fp store non-local double	
9E	29FE	fpstnli32	4	store non-local int32	

Table 4.21 IMS T800 floating point load/store operation codes

Processor cycles are shown as Typical/Maximum cycles.

Table 4.22 IMS T800 floating point general operation codes

Operation Code	Memory Code	Mnemonic	Processor Cycles	Name	D E
AB	2AFB	fpentry	1	floating point unit entry	
A4	2AF4	fprev	1	fp reverse	
A3	2AF3	fpdup	1	fp duplicate	

Table 4.23 IMS T800 floating point rounding operation codes

Operation Code	Memory Code	Mnemonic	Processor Cycles	Name	D E
22	S	fpurn	1	set rounding mode to round nearest	
06	s	fpurz	1	set rounding mode to round zero	
04	s	fpurp	1	set rounding mode to round positive	
05	s	fpurm	1	set rounding mode to round minus	

Table 4.24 IMS T800 floating point error operation codes

Operation Code	Memory Code	Mnemonic	Processor Cycles	Name	D E
83	28F3	fpchkerror	1	check fp error	E
9C	29FC.	fptesterror	2	test fp error false and clear	F
23	S	fpuseterror	1	set fp error	F
9C	S	fpuclearerror	1	clear fp error	F

Operation Code	Memory Code	Mnemonic	Processor Cycles	Name	D E
94	29F4	fpgt	4/6	fp greater than	F
95	29F5	fpeq	3/5	fp equality	F
92	29F2	fpordered	3/4	fp orderability	
91	29F1	fpnan	2/3	fp NaN	
93	29F3	fpnotfinite	2/2	fp not finite	
0E	s	fpuchki32	3/4	check in range of type int32	F
0F	s	fpuchki64	3/4	check in range of type int64	F

Table 4.25 IMS T800 floating point comparison operation codes

Processor cycles are shown as Typical/Maximum cycles.

Table 4.26 IMS T800 floating point conversion operation codes

Operation Code	Memory Code	Mnemonic	Processor Cycles	Name	D E
07	S	fpur32tor64	3/4	real32 to real64	F
08	s	fpur64tor32	6/9	real64 to real32	F
9D	29FD	fprtoi32	7/9	real to int32	F
96	29F6	fpi32tor32	8/10	int32 to real32	
98	29F8	fpi32tor64	8/10	int32 to real64	
9A	29FA	fpb32tor64	8/8	bit32 to real64	1
0D	s	fpunoround	2/2	real64 to real32, no round	
A1	2AF1	fpint	5/6	round to floating integer	F

Processor cycles are shown as Typical/Maximum cycles.

Table 4.27 IMS T800 floating point arithmetic operation codes

Operation	Memory		Processo	or cycles		D
Code	Code	Mnemonic	Single	Double	Name	E
87	28F7	fpadd	6/9	6/9	fp add	F
89	28F9	fpsub	6/9	6/9	fp subtract	F
8B	28FB	fpmul	11/18	18/27	fp multiply	F
8C	28FC	fpdiv	16/28	31/43	fp divide	F
0B	s	fpuabs	2/2	2/2	fp absolute	F
8F	28FF	fpremfirst	36/46	36/46	fp remainder first step	F
90	29F0	fpremstep	32/36	32/36	fp remainder iteration	
01	s	fpusqrtfirst	27/29	27/29	fp square root first step	F
02	s	fpusqrtstep	42/42	42/42	fp square root step	
03	s	fpusqrtlast	8/9	8/9	fp square root end	
0A	s	fpuexpinc32	6/9	6/9	multiply by 2 <sup>32</sup>	F
09	s	fpuexpdec32	6/9	6/9	divide by 2 <sup>32</sup>	F
12	s	fpumulby2	6/9	6/9	multiply by 2.0	F
11	S	fpudivby2	6/9	6/9	divide by 2.0	F

Processor cycles are shown as Typical/Maximum cycles.

# 5 Floating point unit

The 64 bit FPU provides single and double length arithmetic to floating point standard ANSI-IEEE 754-1985. It is able to perform floating point arithmetic concurrently with the central processor unit (CPU), sustaining 3.3 Mflops on a 30 MHz device. All data communication between memory and the FPU occurs under control of the CPU.

The FPU consists of a microcoded computing engine with a three deep floating point evaluation stack for manipulation of floating point numbers. These stack registers are *FA*, *FB* and *FC*, each of which can hold either 32 bit or 64 bit data; an associated flag, set when a floating point value is loaded, indicates which. The stack behaves in a similar manner to the CPU stack (page 193).

As with the CPU stack, the FPU stack is not saved when rescheduling (page 196) occurs. The FPU can be used in both low and high priority processes. When a high priority process interrupts a low priority one the FPU state is saved inside the FPU. The CPU will service the interrupt immediately on completing its current operation. The high priority process will not start, however, before the FPU has completed its current operation.

Points in an instruction stream where data need to be transferred to or from the FPU are called *synchronisation points*. At a synchronisation point the first processing unit to become ready will wait until the other is ready. The data transfer will then occur and both processors will proceed concurrently again. In order to make full use of concurrency, floating point data source and destination addresses can be calculated by the CPU whilst the FPU is performing operations on a previous set of data. Device performance is thus optimised by minimising the CPU and FPU idle times.

The FPU has been designed to operate on both single length (32 bit) and double length (64 bit) floating point numbers, and returns results which fully conform to the ANSI-IEEE 754-1985 floating point arithmetic standard. Denormalised numbers are fully supported in the hardware. All rounding modes defined by the standard are implemented, with the default being round to nearest.

The basic addition, subtraction, multiplication and division operations are performed by single instructions. However, certain less frequently used floating point instructions are selected by a value in register A (when allocating registers, this should be taken into account). A *load constant* instruction *ldc* is used to load register A; the *floating point entry* instruction *fpentry* then uses this value to select the floating point operation. This pair of instructions is termed a *selector sequence*.

Names of operations which use *fpentry* begin with *fpu*. A typical usage, returning the absolute value of a floating point number, would be

ldc fpuabs; fpentry;

Since the indirection code for fpuabs is **0B**, it would be encoded as

Table 5.1 fpentry coding

Mnem	nonic	Function code	Memory code
ldc	fpuabs	#4	#4B
fpentry is coded as	(op. code #AB)		#2AFB
pfix opr	#A #B	#2 #F	#2A #FB

The *remainder* and *square root* instructions take considerably longer than other instructions to complete. In order to minimise the interrupt latency period of the transputer they are split up to form instruction sequences. As an example, the instruction sequence for a single length square root is

fpusqrtfirst; fpusqrtstep; fpusqrtstep; fpusqrtlast;

The FPU has its own error flag *FP\_Error*. This reflects the state of evaluation within the FPU and is set in circumstances where invalid operations, division by zero or overflow exceptions to the ANSI-IEEE 754-1985 standard would be flagged (page 202). *FP\_Error* is also set if an input to a floating point operation is infinite or is not a number (NaN). The *FP\_Error* flag can be set, tested and cleared without affecting the main *Error* flag, but can also set *Error* when required (page 202). Depending on how a program is compiled, it is possible for both unchecked and fully checked floating point arithmetic to be performed.

Further details on the operation of the FPU can be found in *Transputer Instruction Set - A Compiler Writers' Guide.* 

T800-20 T800-30 Operation Single length Double length Single length Double length add 350 ns 350 ns 233 ns 233 ns subtract 350 ns 350 ns 233 ns 233 ns multiply 550 ns 1000 ns 367 ns 667 ns divide 850 ns 1600 ns 567 ns 1067 ns

Table 5.2 Typical floating point operation times for IMS T800

Timing is for operations where both operands are normalised fp numbers.

# 6 System services

System services include all the necessary logic to initialise and sustain operation of the device. They also include error handling and analysis facilities.

#### 6.1 Power

Power is supplied to the device via the VCC and GND pins. Several of each are provided to minimise inductance within the package. All supply pins must be connected. The supply must be decoupled close to the chip by at least one 100 nF low inductance (e.g. ceramic) capacitor between VCC and GND. Four layer boards are recommended; if two layer boards are used, extra care should be taken in decoupling.

Input voltages must not exceed specification with respect to VCC and GND, even during power-up and power-down ramping, otherwise *latchup* can occur. CMOS devices can be permanently damaged by excessive periods of latchup.

# 6.2 CapPlus, CapMinus

The internally derived power supply for internal clocks requires an external low leakage, low inductance  $1\mu F$  capacitor to be connected between **CapPlus** and **CapMinus**. A ceramic capacitor is preferred, with an impedance less than 3 Ohms between 100 KHz and 10 MHz. If a polarised capacitor is used the negative terminal should be connected to **CapMinus**. Total PCB track length should be less than 50 mm. The connections must not touch power supplies or other noise sources.

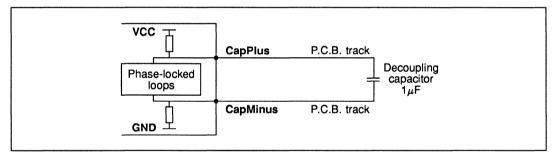


Figure 6.1 Recommended PLL decoupling

## 6.3 Clockin

Transputer family components use a standard clock frequency, supplied by the user on the **ClockIn** input. The nominal frequency of this clock for all transputer family components is 5 MHz, regardless of device type, transputer word length or processor cycle time. High frequency internal clocks are derived from **ClockIn**, simplifying system design and avoiding problems of distributing high speed clocks externally.

A number of transputer devices may be connected to a common clock, or may have individual clocks providing each one meets the specified stability criteria. In a multi-clock system the relative phasing of **ClockIn** clocks is not important, due to the asynchronous nature of the links. Mark/spage ratio is unimportant provided the specified limits of **ClockIn** pulse widths are met.

Oscillator stability is important. ClockIn must be derived from a crystal oscillator; RC oscillators are not sufficiently stable. ClockIn must not be distributed through a long chain of buffers. Clock edges must be monotonic and remain within the specified voltage and time limits.

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTE
TDCLDCH	ClockIn pulse width low	40			ns	1
TDCHDCL	ClockIn pulse width high	40			ns	1
TDCLDCL	ClockIn period		200		ns	1,2,4
TDCerror	ClockIn timing error			±0.5	ns	1,3
TDC1DC2	Difference in ClockIn for 2 linked devices			400	ppm	1,4
TDCr	ClockIn rise time			10	ns	1,5
TDCf	ClockIn fall time			8	ns	1,5

Table 6.1 Input clock

## Notes

- 1 These paramters are not tested.
- 2 Measured between corresponding points on consecutive falling edges.
- 3 Variation of individual falling edges from their nominal times.
- 4 This value allows the use of 200ppm crystal oscillators for two devices connected together by a link.
- 5 Clock transitions must be monotonic within the range VIH to VIL (table 11.3).

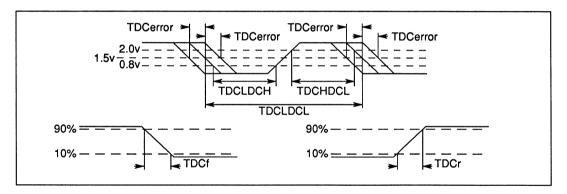


Figure 6.2 ClockIn timing

# 6.4 ProcSpeedSelect0-2

Processor speed of the IMS T800 is variable in discrete steps. The desired speed can be selected, up to the maximum rated for a particular component, by the three speed select lines **ProcSpeedSelect0-2**. The pins are tied high or low, according to the table below, for the various speeds. The pins are arranged so that the IMS T425 can be plugged directly into a board designed for a IMS T800.

Only six of the possible speed select combinations are currently used; the other two are not valid speed selectors. The frequency of **ClockIn** for the speeds given in the table is 5 MHz.

System services 213

Proc Speed Select2	Proc Speed Select1	Proc Speed Select0	Processor Clock Speed MHz	Processor Cycle Time ns	Notes
0	0	0	20.0	50.0	
0	0	1	22.5	44.4	
0	1	0	25.0	40.0	
0	1	1	30.0	33.3	
1	0	0	35.0	28.6	
1	0	1			Invalid
1	1	0	17.5	57.1	
1	1	1			Invalid

Table 6.2 Processor speed selection

Note: Inclusion of a speed selection in this table does not imply immediate availability.

## 6.5 Reset

6

Reset can go high with VCC, but must at no time exceed the maximum specified voltage for VIH. After VCC is valid ClockIn should be running for a minimum period TDCVRL before the end of Reset. The falling edge of Reset initialises the transputer, triggers the memory configuration sequence and starts the bootstrap routine. Link outputs are forced low during reset; link inputs and EventReq should be held low. Memory request (DMA) must not occur whilst Reset is high but can occur before bootstrap (page 238).

After the end of **Reset** there will be a delay of 144 periods of **ClockIn** (figure 6.3). Following this, the **MemWrD0**, **MemRfD1** and **MemAD2-31** pins will be scanned to check for the existence of a pre-programmed memory interface configuration (page 228). This lasts for a further 144 periods of **ClockIn**. Regardless of whether a configuration was found, 36 configuration read cycles will then be performed on external memory using the default memory configuration (page 230), in an attempt to access the external configuration ROM. A delay will then occur, its period depending on the actual configuration. Finally eight complete and consecutive refresh cycles will initialise any dynamic RAM, using the new memory configuration. If the memory configuration does not enable refresh of dynamic RAM the refresh cycles will be replaced by an equivalent delay with no external memory activity.

If **BootFromRom** is high bootstrapping will then take place immediately, using data from external memory; otherwise the transputer will await an input from any link. The processor will be in the low priority state.

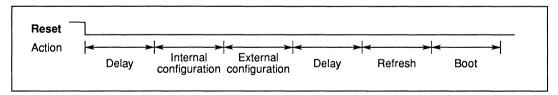


Figure 6.3 IMS T800 post-reset sequence

## 6.6 Bootstrap

The transputer can be bootstrapped either from a link or from external ROM. To facilitate debugging, **Boot-FromRom** may be dynamically changed but must obey the specified timing restrictions. It is sampled once only by the transputer, before the first instruction is executed after **Reset** is taken low.

If **BootFromRom** is connected high (e.g. to **VCC**) the transputer starts to execute code from the top two bytes in external memory, at address #7FFFFFFE. This location should contain a backward jump to a program in ROM. Following this access, **BootFromRom** may be taken low if required. The processor is in the low priority

state, and the W register points to MemStart (page 217).

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTE
TPVRH	Power valid before Reset	10			ms	
TRHRL	Reset pulse width high	8			ClockIn	1
TDCVRL	ClockIn running before Reset end	10			ms	2
TAHRH	Analyse setup before Reset	3			ms	
TRLAL	Analyse hold after Reset end	1			ClockIn	1
TBRVRL	BootFromRom setup	0			ms	
TRLBRX	BootFromRom hold after Reset	0			ms	3
TALBRX	BootFromRom hold after Analyse					3

Table 6.3 Reset and Analyse

- 1 Full periods of Clockin TDCLDCL required.
- 2 At power-on reset.
- 3 Must be stable until after end of bootstrap period. See Bootstrap section.

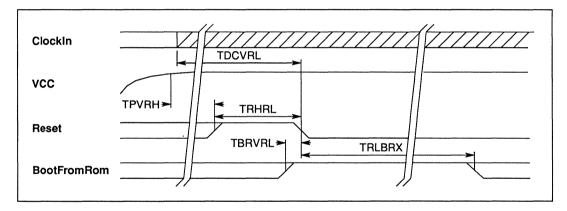


Figure 6.4 Transputer reset timing with Analyse low

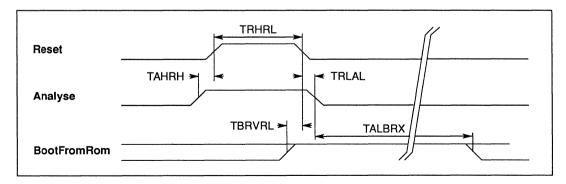


Figure 6.5 Transputer reset and analyse timing

System services 215

If **BootFromRom** is connected low (e.g. to **GND**) the transputer will wait for the first bootstrap message to arrive on any one of its links. The transputer is ready to receive the first byte on a link within two processor cycles **TPCLPCL** after **Reset** goes low.

If the first byte received (the control byte) is greater than 1 it is taken as the quantity of bytes to be input. The following bytes, to that quantity, are then placed in internal memory starting at location *MemStart*. Following reception of the last byte the transputer will start executing code at *MemStart* as a low priority process. **BootFromRom** may be taken high after reception of the last byte, if required. The memory space immediately above the loaded code is used as work space. Messages arriving on other links after the control byte has been received and on the bootstrapping link after the last bootstrap byte will be retained until a process inputs from them.

## 6.7 Peek and poke

Any location in internal or external memory can be interrogated and altered when the transputer is waiting for a bootstrap from link. If the control byte is 0 then eight more bytes are expected on the same link. The first four byte word is taken as an internal or external memory address at which to poke (write) the second four byte word. If the control byte is 1 the next four bytes are used as the address from which to peek (read) a word of data; the word is sent down the output channel of the same link.

Following such a peek or poke, the transputer returns to its previously held state. Any number of accesses may be made in this way until the control byte is greater than 1, when the transputer will commence reading its bootstrap program. Any link can be used, but addresses and data must be transmitted via the same link as the control byte.

## 6.8 Analyse

If **Analyse** is taken high when the transputer is running, the transputer will halt at the next descheduling point (page 201). From **Analyse** being asserted, the processor will halt within three time slice periods plus the time taken for any high priority process to complete. As much of the transputer status is maintained as is necessary to permit analysis of the halted machine. Processor flags **Error** and **HaltOnError** are not altered at reset, whether **Analyse** is asserted or not. Memory refresh continues.

Input links will continue with outstanding transfers. Output links will not make another access to memory for data but will transmit only those bytes already in the link buffer. Providing there is no delay in link acknowledgement, the links should be inactive within a few microseconds of the transputer halting.

Reset should not be asserted before the transputer has halted and link transfers have ceased. When Reset is taken low whilst Analyse is high, neither the memory configuration sequence nor the block of eight refresh cycles will occur; the previous memory configuration will be used for any external memory accesses. If BootFromRom is high the transputer will bootstrap as soon as Analyse is taken low, otherwise it will await a control byte on any link. If Analyse is taken low without Reset going high the transputer state and operation are undefined. After the end of a valid Analyse sequence the registers have the values given in table 6.4.

Table 6.4 Register values after Analyse

- I MemStart if bootstrapping from a link, or the external memory bootstrap address if bootstrapping from ROM.
- W MemStart if bootstrapping from ROM, or the address of the first free word after the bootstrap program if bootstrapping from link.
- A The value of I when the processor halted.
- B The value of W when the processor halted, together with the priority of the process when the transputer was halted (i.e. the W descriptor).
- C The ID of the bootstrapping link if bootstrapping from link.

# 6.9 Error, Errorin

The **Error** pin carries the OR'ed output of the internal *Error* flag and the **ErrorIn** input. If **Error** is high it indicates either that **ErrorIn** is high or that an error was detected in one of the processes. An internal error can be caused, for example, by arithmetic overflow, divide by zero, array bounds violation or software setting the flag directly (page 202). It can also be set from the floating point unit under certain circumstances (page 202, 209). Once set, the *Error* flag is only cleared by executing the instruction *testerr*. The error is not cleared by processor reset, in order that analysis can identify any errant transputer (page 215).

A process can be programmed to stop if the *Error* flag is set; it cannot then transmit erroneous data to other processes, but processes which do not require that data can still be scheduled. Eventually all processes which rely, directly or indirectly, on data from the process in error will stop through lack of data. **ErrorIn** does not directly affect the status of a processor in any way.

By setting the *HaltOnError* flag the transputer itself can be programmed to halt if *Error* becomes set. If *Error* becomes set after *HaltOnError* has been set, all processes on that transputer will cease but will not necessarily cause other transputers in a network to halt. Setting *HaltOnError* after *Error* will not cause the transputer to halt; this allows the processor reset and analyse facilities to function with the flags in indeterminate states.

An alternative method of error handling is to have the errant process or transputer cause all transputers to halt. This can be done by 'daisy-chaining' the **ErrorIn** and **Error** pins of a number of processors and applying the final **Error** output signal to the **EventReq** pin of a suitably programmed master transputer. Since the process state is preserved when stopped by an error, the master transputer can then use the analyse function to debug the fault. When using such a circuit, note that the *Error* flag is in an indeterminate state on power up; the circuit and software should be designed with this in mind.

Error checks can be removed completely to optimise the performance of a proven program; any unexpected error then occurring will have an arbitrary undefined effect.

If a high priority process pre-empts a low priority one, status of the *Error* and *HaltOnError* flags is saved for the duration of the high priority process and restored at the conclusion of it. Status of both flags is transmitted to the high priority process. Either flag can be altered in the process without upsetting the error status of any complex operation being carried out by the pre-empted low priority process.

In the event of a transputer halting because of *HaltOnError*, the links will finish outstanding transfers before shutting down. If **Analyse** is asserted then all inputs continue but outputs will not make another access to memory for data. Memory refresh will continue to take place.

After halting due to the *Error* flag changing from 0 to 1 whilst *HaltOnError* is set, register *I* points two bytes past the instruction which set *Error*. After halting due to the **Analyse** pin being taken high, register *I* points one byte past the instruction being executed. In both cases *I* will be copied to register *A*.

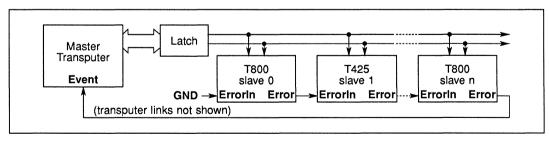


Figure 6.6 Error handling in a multi-transputer system

# 7 Memory

The IMS T800 has 4 Kbytes of fast internal static memory for high rates of data throughput. Each internal memory access takes one processor cycle **ProcClockOut** (page 219). The transputer can also access 4 Gbytes of external memory space. Internal and external memory are part of the same linear address space. Internal RAM can be disabled by holding **DisableIntRAM** high. All internal addresses are then mapped to external RAM. This pin should not be altered after **Reset** has been taken low.

IMS T800 memory is byte addressed, with words aligned on four-byte boundaries. The least significant byte of a word is the lowest addressed byte.

The bits in a byte are numbered 0 to 7, with bit 0 the least significant. The bytes are numbered from 0, with byte 0 the least significant. In general, wherever a value is treated as a number of component values, the components are numbered in order of increasing numerical significance, with the least significant component numbered 0. Where values are stored in memory, the least significant component value is stored at the lowest (most negative) address.

Internal memory starts at the most negative address #80000000 and extends to #80000FFF. User memory begins at #80000070; this location is given the name *MemStart*.

The reserved area of internal memory below MemStart is used to implement link and event channels.

Two words of memory are reserved for timer use, *TPtrLoc0* for high priority processes and *TPtrLoc1* for low priority processes. They either indicate the relevant priority timer is not in use or point to the first process on the timer queue at that priority level.

Values of certain processor registers for the current low priority process are saved in the reserved *IntSaveLoc* locations when a high priority process pre-empts a low priority one. Other locations are reserved for extended features such as block moves and floating point operations.

External memory space starts at #80001000 and extends up through #00000000 to #7FFFFFFF. Memory configuration data and ROM bootstrapping code must be in the most positive address space, starting at #7FFFFF6C and #7FFFFFFE respectively. Address space immediately below this is conventionally used for ROM based code.

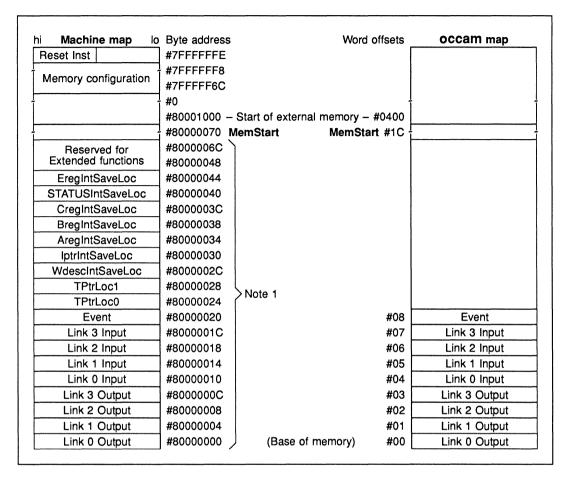


Figure 7.1 IMS T800 memory map

#### Notes

1 These locations are used as auxiliary processor registers and should not be manipulated by the user. Like processor registers, their contents may be useful for implementing debugging tools (Analyse, page 215). For details see Transputer Instruction Set - A Compiler Writers' Guide.

# 8 External memory interface

The External Memory Interface (EMI) allows access to a 32 bit address space, supporting dynamic and static RAM as well as ROM and EPROM. EMI timing can be configured at **Reset** to cater for most memory types and speeds, and a program is supplied with the Transputer Development System to aid in this configuration.

There are 17 internal configurations which can be selected by a single pin connection (page 228). If none are suitable the user can configure the interface to specific requirements, as shown in page 230.

#### 8.1 ProcClockOut

This clock is derived from the internal processor clock, which is in turn derived from **Clockin**. Its period is equal to one internal microcode cycle time, and can be derived from the formula

## TPCLPCL = TDCLDCL / PLLx

where TPCLPCL is the ProcClockOut Period, TDCLDCL is the ClockIn Period and PLLx is the phase lock loop factor for the relevant speed part, obtained from the ordering details (Ordering section).

The time value Tm is used to define the duration of Tstates and, hence, the length of external memory cycles; its value is exactly half the period of one ProcClockOut cycle (0.5\*TPCLPCL), regardless of mark/space ratio of ProcClockOut.

Edges of the various external memory strobes coincide with rising or falling edges of **ProcClockOut**. It should be noted, however, that there is a skew associated with each coincidence. The value of skew depends on whether coincidence occurs when the **ProcClockOut** edge and strobe edge are both rising, when both are falling or if either is rising when the other is falling. Timing values given in the strobe tables show the best and worst cases. If a more accurate timing relationship is required, the exact **Tstate** timing and strobe edge to **ProcClockOut** relationships should be calculated and the correct skew factors applied from the edge skew timing table 8.4.

The timing parameters in the following tables are based on full characterisation of the 17 MHz and 20 MHz parts. Data for higher speeds is based on tests on a limited number of samples and may change when full characterisation is completed.

#### 8.2 Tstates

The external memory cycle is divided into six **Tstates** with the following functions:

- T1 Address setup time before address valid strobe.
- T2 Address hold time after address valid strobe.
- T3 Read cycle tristate or write cycle data setup.
- T4 Extendable data setup time.
- T5 Read or write data.
- T6 Data hold.

Under normal conditions each **Tstate** may be from one to four periods **Tm** long, the duration being set during memory configuration. The default condition on **Reset** is that all **Tstates** are the maximum four periods **Tm** long to allow external initialisation cycles to read slow ROM.

Period T4 can be extended indefinitely by adding externally generated wait states.

An external memory cycle is always an even number of periods Tm in length and the start of T1 always coincides with a rising edge of ProcClockOut. If the total configured quantity of periods Tm is an odd number, one extra period Tm will be added at the end of T6 to force the start of the next T1 to coincide with

a rising edge of ProcClockOut. This period is designated E in configuration diagrams (figure 8.11).

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTE
TPCLPCL	ProcClockOut period	<b>a</b> -2	а	a+2	ns	1,5
TPCHPCL	ProcClockOut pulse width high	<b>b</b> -11.5	b	<b>b</b> +3.5	ns	2,5
TPCLPCH	ProcClockOut pulse width low		С	İ	ns	3,5
Tm	ProcClockOut half cycle	b-1	b	b+1	ns	2,5
TPCstab	ProcClockOut stability			8	%	4.5

Table 8.1 ProcClockOut

#### Notes

- 1 a is TDCLDCL/PLLx.
- 2 b is 0.5\*TPCLPCL (half the processor clock period).
- 3 c is TPCLPCL-TPCHPCL.
- 4 Stability is the variation of cycle periods between two consecutive cycles, measured at corresponding points on the cycles.
- 5 This parameter is sampled and not 100% tested.

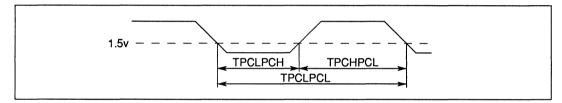


Figure 8.1 IMS T800 ProcClockOut timing

#### 8.3 Internal access

During an internal memory access cycle the external memory interface bus **MemAD2-31** reflects the word address used to access internal RAM, **MemnotWrD0** reflects the read/write operation and **MemnotRfD1** is high; all control strobes are inactive. This is true unless and until a memory refresh cycle or DMA (memory request) activity takes place, when the bus will carry the appropriate external address or data.

The bus activity is not adequate to trace the internal operation of the transputer in full, but may be used for hardware debugging in conjuction with peek and poke (page 215).

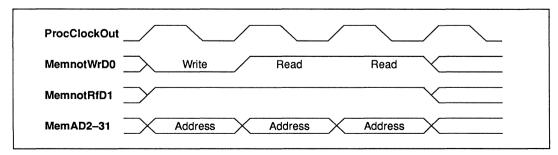


Figure 8.2 IMS T800 bus activity for internal memory cycle

#### 8.4 MemAD2-31

External memory addresses and data are multiplexed on one bus. Only the top 30 bits of address are output on the external memory interface, using pins **MemAD2-31**. They are normally output only during **Tstates T1** and **T2**, and should be latched during this time. Byte addressing is carried out internally by the transputer for read cycles. For write cycles the relevant bytes in memory are addressed by the write strobes **notMemWrB0-3**.

The data bus is 32 bits wide. It uses MemAD2-31 for the top 30 bits and MemnotRfD1 and MemnotWrD0 for the lower two bits. Read cycle data may be set up on the bus at any time after the start of T3, but must be valid when the transputer reads it at the end of T5. Data may be removed any time during T6, but must be off the bus no later than the end of that period.

Write data is placed on the bus at the start of T3 and removed at the end of T6. If T6 is extended to force the next cycle Tmx (page 221) to start on a rising edge of ProcClockOut, data will be valid during this time also.

## 8.5 MemnotWrD0

During T1 and T2 this pin will be low if the cycle is a write cycle, otherwise it will be high. During Tstates T3 to T6 it becomes bit 0 of the data bus. In both cases it follows the general timing of MemAD2-31.

#### 8.6 MemnotRfD1

During T1 and T2, this pin is low if the address on MemAD2-31 is a refresh address, otherwise it is high. During Tstates T3 to T6 it becomes bit 1 of the data bus. In both cases it follows the general timing of MemAD2-31.

## 8.7 notMemRd

For a read cycle the read strobe **notMemRd** is low during **T4** and **T5**. Data is read by the transputer on the rising edge of this strobe, and may be removed immediately afterward. If the strobe duration is insufficient it may be extended by adding extra periods **Tm** to either or both of the **Tstates T4** and **T5**. Further extension may be obtained by inserting wait states at the end of **T4**.

In the read cycle timing diagrams **ProcClockOut** is included as a guide only; it is shown with each **Tstate** configured to one period **Tm**.

#### 8.8 notMemS0-4

To facilitate control of different types of memory and devices, the EMI is provided with five strobe outputs, four of which can be configured by the user. The strobes are conventionally assigned the functions shown in the read and write cycle diagrams, although there is no compulsion to retain these designations.

notMemS0 is a fixed format strobe. Its leading edge is always coincident with the start of T2 and its trailing edge always coincident with the end of T5.

The leading edge of **notMemS1** is always coincident with the start of **T2**, but its duration may be configured to be from zero to 31 periods **Tm**. Regardless of the configured duration, the strobe will terminate no later than the end of **T6**. The strobe is sometimes programmed to extend beyond the normal end of **Tmx**. When wait states are inserted into an EMI cycle the end of **Tmx** is delayed, but the potential active duration of the strobe is not altered. Thus the strobe can be configured to terminate relatively early under certain conditions (page 236). If **notMemS1** is configured to be zero it will never go low.

notMemS2, notMemS3 and notMemS4 are identical in operation. They all terminate at the end of T5, but the start of each can be delayed from one to 31 periods Tm beyond the start of T2. If the duration of one of these strobes would take it past the end of T5 it will stay high. This can be used to cause a strobe to become active only when wait states are inserted. If one of these strobes is configured to zero it will never go low. Figure 8.5 shows the effect of Wait on strobes in more detail; each division on the scale is one period Tm.

lable	8.2	Read

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTE
TaZdV	Address tristate to data valid	0			ns	
TdVRdH	Data setup before read	25	İ		ns	
TRdHdX	Data hold after read	0			ns	
TS0LRdL	notMemS0 before start of read	a-4	а	a+4	ns	1
TS0HRdH	End of read from end of notMemS0	-4	l	4	ns	1
TRdLRdH	Read period	<b>b</b> -3		b+5	ns	2

- 1 a is total of T2+T3 where T2, T3 can be from one to four periods Tm each in length.
- 2 b is total of T4+Twalt+T5 where T4, T5 can be from one to four periods Tm each in length and Twalt may be any number of periods Tm in length.

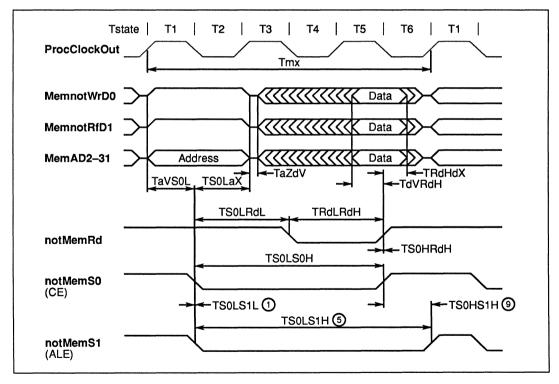


Figure 8.3 IMS T800 external read cycle: static memory

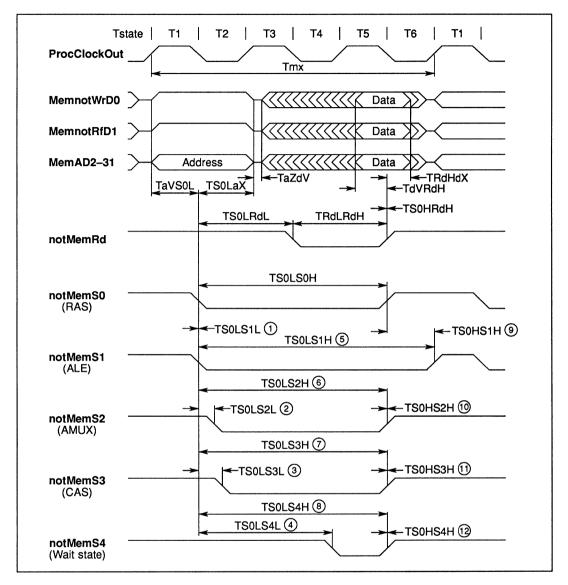


Figure 8.4 IMS T800 external read cycle: dynamic memory

SYMBOL	n	PARAMETER	MIN	NOM	MAX	UNITS	NOTE
TaVS0L		Address setup before notMemS0	<b>a</b> -8			ns	1
TS0LaX		Address hold after notMemS0	<b>b</b> -8	b	<b>b</b> +8	ns	2
TS0LS0H		notMemS0 pulse width low	<b>c</b> -5		<b>c</b> +6	ns	3
TS0LS1L	1	notMemS1 from notMemS0	-4		4	ns	
TS0LS1H	5	notMemS1 end from notMemS0	<b>d</b> -1		<b>d</b> +9	ns	4,6
TS0HS1H	9	notMemS1 end from notMemS0 end	<b>e</b> -8		e+4	ns	5,6
TS0LS2L	2	notMemS2 delayed after notMemS0	<b>f</b> -6		f+5	ns	7
TS0LS2H	6	notMemS2 end from notMemS0	<b>c</b> -5		c+7	ns	3
TS0HS2H	10	notMemS2 end from notMemS0 end	-4		7	ns	
TS0LS3L	3	notMemS3 delayed after notMemS0	<b>f</b> -6		<b>f</b> +5	ns	7
TS0LS3H	7	notMemS3 end from notMemS0	<b>c</b> -5		<b>c</b> +7	ns	3
TS0HS3H	11	notMemS3 end from notMemS0 end	-4		7	ns	
TS0LS4L	4	notMemS4 delayed after notMemS0	<b>f</b> -6		f+5	ns	7
TS0LS4H	8	notMemS4 end from notMemS0	<b>c</b> -5		c+7	ns	3
TS0HS4H	12	notMemS4 end from notMemS0 end	-4		7	ns	
Tmx		Complete external memory cycle		g			8

Table 8.3 IMS T800 strobe timing

- 1 a is T1 where T1 can be from one to four periods Tm in length.
- 2 b is T2 where T2 can be from one to four periods Tm in length.
- 3 c is total of T2+T3+T4+Twait+T5 where T2, T3, T4, T5 can be from one to four periods Tm each in length and Twait may be any number of periods Tm in length.
- 4 d can be from zero to 31 periods Tm in length.
- 5 e can be from -27 to +4 periods Tm in length.
- 6 If the configuration would cause the strobe to remain active past the end of **T6** it will go high at the end of **T6**. If the strobe is configured to zero periods **Tm** it will remain high throughout the complete cycle **Tmx**.
- 7 f can be from zero to 31 periods Tm in length. If this length would cause the strobe to remain active past the end of T5 it will go high at the end of T5. If the strobe value is zero periods Tm it will remain low throughout the complete cycle Tmx.
- 8 g is one complete external memory cycle comprising the total of T1+T2+T3+T4+Twait+T5+T6 where T1, T2, T3, T4, T5 can be from one to four periods Tm each in length, T6 can be from one to five periods Tm in length and Twait may be zero or any number of periods Tm in length.

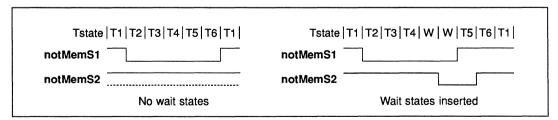


Figure 8.5 IMS T800 effect of wait states on strobes

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTE
TPCHS0H	notMemS0 rising from ProcClockOut rising	-6		4	ns	
TPCLS0H	notMemS0 rising from ProcClockOut falling	-5		10	ns	
TPCHS0L	notMemS0 falling from ProcClockOut rising	-8		3	ns	
TPCLS0L	notMemS0 falling from ProcClockOut falling	-5		7	ns	

Table 8.4 Strobe S0 to ProcClockOut skew

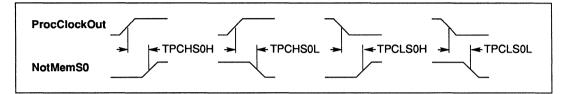


Figure 8.6 IMS T800 skew of notMemS0 to ProcClockOut

#### 8.9 notMemWrB0-3

Because the transputer uses word addressing, four write strobes are provided; one to write each byte of the word. If a particular byte is not to be written, then the corresponding data outputs are tristated. **notMemWrB0** addresses the least significant byte.

The transputer has both early and late write cycle modes. For a late write cycle the relevant write strobes notMemWrB0-3 are low during T4 and T5; for an early write they are also low during T3. Data should be latched into memory on the rising edge of the strobes in both cases, although it is valid until the end of T6. If the strobe duration is insufficient, it may be extended at configuration time by adding extra periods Tm to either or both of Tstates T4 and T5 for both early and late modes. For an early cycle they may also be added to T3. Further extension may be obtained by inserting wait states at the end of T4. If the data hold time is insufficient, extra periods Tm may be added to T6 to extend it.

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTE
TdVWrH	Data setup before write	<b>d</b> -7		<b>d</b> +10	ns	1,5
TWrHdX	Data hold after write	<b>a</b> -10		<b>a</b> +5	ns	1,2
TS0LWrL	notMemS0 before start of early write	<b>b</b> -5		<b>b</b> +5	ns	1,3
	notMemS0 before start of late write	<b>c</b> -5		<b>c</b> +5	ns	1,4
TS0HWrH	End of write from end of notMemS0	-5		4	ns	1
TWrLWrH	Early write pulse width	<b>d</b> -4		<b>d</b> +7	ns	1,5
ĺ	Late write pulse width	<b>e</b> -4		<b>e</b> +7	ns	1,6

Table 8.5 Write

- 1 Timing is for all write strobes notMemWrB0-3.
- 2 a is T6 where T6 can be from one to five periods Tm in length.
- 3 b is T2 where T2 can be from one to four periods Tm in length.
- 4 c is total of T2+T3 where T2, T3 can be from one to four periods Tm each in length.
- 5 d is total of T3+T4+Twait+T5 where T3, T4, T5 can be from one to four periods Tm each in length and Twait may be zero or any number of periods Tm in length.
- 6 e is total of T4+Twait+T5 where T4, T5 can be from one to four periods Tm each in length and Twait may be zero or any number of periods Tm in length.

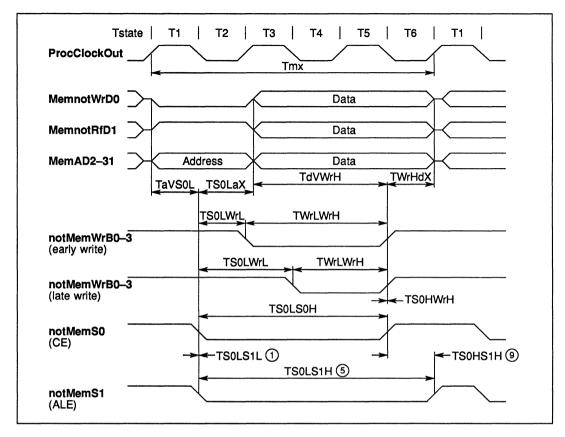


Figure 8.7 IMS T800 external write cycle

In the write cycle timing diagram ProcClockOut is included as a guide only; it is shown with each Tstate configured to one period Tm. The strobe is inactive during internal memory cycles.

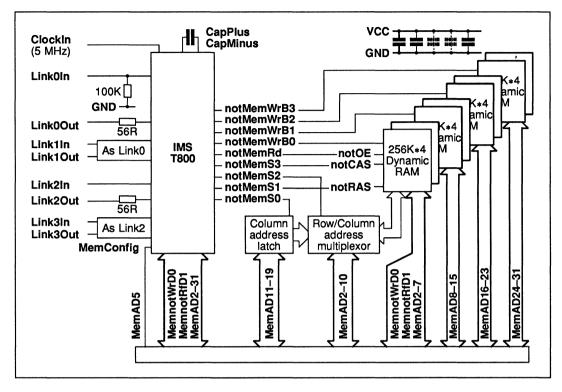


Figure 8.8 IMS T800 dynamic RAM application

# 8.10 MemConfig

**MemConfig** is an input pin used to read configuration data when setting external memory interface (EMI) characteristics. It is read by the processor on two occasions after **Reset** goes low; first to check if one of the preset internal configurations is required, then to determine a possible external configuration.

# 8.10.1 Internal configuration

The internal configuration scan comprises 64 periods TDCLDCL of ClockIn during the internal scan period of 144 ClockIn periods. MemnotWrD0, MemnotRfD1 and MemAD2-32 are all high at the beginning of the scan. Starting with MemnotWrD0, each of these lines goes low successively at intervals of two ClockIn periods and stays low until the end of the scan. If one of these lines is connected to MemConfig the preset internal configuration mode associated with that line will be used as the EMI configuration. The default configuration is that defined in the table for MemAD31; connecting MemConfig to VCC will also produce this default configuration. Note that only 17 of the possible configurations are valid, all others remain at the default configuration.

	Di	uratio	n of erio			te		Strobe coefficient		Write cycle	Refresh interval	Cycle time	
Di-	T-4	то.	Τ0	T4	75	TC		-0	-2	-4	A	Clockin	Proc
Pin	T1	T2	T3	T4	T5	T6	<b>S1</b>	<u>s2</u>	<u>s3</u>	s4	type	cycles	cycles
MemnotWrD0	1	1	1	1	1	1	30	1	3	5	late	72	3
MemnotRfD1	1	2	1	1	1	2	30	1	2	7	late	72	4
MemAD2	1	2	1	1	2	3	30	1	2	7	late	72	5
MemAD3	2	3	1	1	2	3	30	1	3	8	late	72	6
MemAD4	1	1	1	1	1	1	3	1	2	3	early	72	3
MemAD5	1	1	2	1	2	1	5	1	2	3	early	72	4
MemAD6	2	1	2	1	3	1	6	1	2	3	early	72	5
MemAD7	2	2	2	1	3	2	7	1	3	4	early	72	6
MemAD8	1	1	1	1	1	1	30	1	2	3	early	t	3
MemAD9	1	1	2	1	2	1	30	2	5	9	early	i †	4
MemAD10	2	2	2	2	4	2	30	2	3	8	late	72	7
MemAD11	3	3	3	3	3	3	30	2	4	13	late	72	9
MemAD12	1	1	2	1	2	1	4	1	2	3	early	72	4
MemAD13	2	1	2	1	2	2	5	1	2	3	early	72	5
MemAD14	2	2	2	1	3	2	6	1	3	4	early	72	6
MemAD15	2	1	2	3	3	3	8	1	2	3	early	72	7
MemAD31	4	4	4	4	4	4	31	30	30	18	late	72	12

Table 8.6 IMS T800 internal configuration coding

<sup>†</sup> Provided for static RAM only.

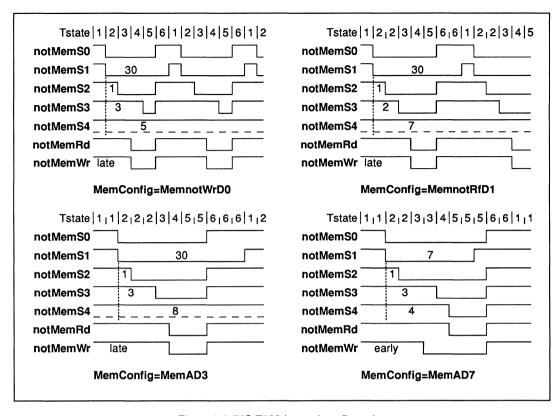


Figure 8.9 IMS T800 internal configuration

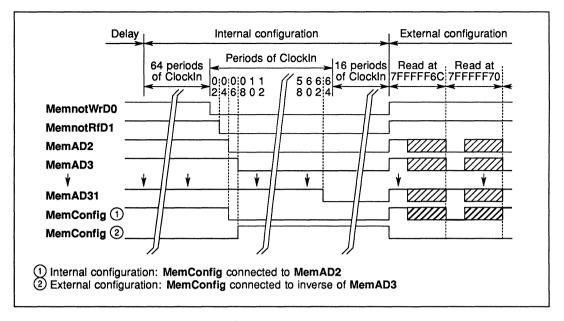


Figure 8.10 IMS T800 internal configuration scan

## 8.10.2 External configuration

If **MemConfig** is held low until **MemnotWrD0** goes low the internal configuration is ignored and an external configuration will be loaded instead. An external configuration scan always follows an internal one, but if an internal configuration occurs any external configuration is ignored.

The external configuration scan comprises 36 successive external read cycles, using the default EMI configuration preset by **MemAD31**. However, instead of data being read on the data bus as for a normal read cycle, only a single bit of data is read on **MemConfig** at each cycle. Addresses put out on the bus for each read cycle are shown in table 8.7, and are designed to address ROM at the top of the memory map. The table shows the data to be held in ROM; data required at the **MemConfig** pin is the inverse of this.

MemConfig is typically connected via an inverter to MemnotWrD0. Data bit zero of the least significant byte of each ROM word then provides the configuration data stream. By switching MemConfig between various data bus lines up to 32 configurations can be stored in ROM, one per bit of the data bus. MemConfig can be permanently connected to a data line or to GND. Connecting MemConfig to GND gives all Tstates configured to four periods; notMemS1 pulse of maximum duration; notMemS2-4 delayed by maximum; refresh interval 72 periods of ClockIn; refresh enabled; late write.

The external memory configuration table 8.7 shows the contribution of each memory address to the 13 configuration fields. The lowest 12 words (#7FFFF6C to #7FFFF98, fields 1 to 6) define the number of extra periods **Tm** to be added to each **Tstate**. If field 2 is 3 then three extra periods will be added to **T2** to extend it to the maximum of four periods.

The next five addresses (field 7) define the duration of **notMemS1** and the following fifteen (fields 8 to 10) define the delays before strobes **notMemS2-4** become active. The five bits allocated to each strobe allow durations of from 0 to 31 periods **Tm**, as described in strobes page 221.

Addresses #7FFFFEC to #7FFFFF4 (fields 11 and 12) define the refresh interval and whether refresh is to be used, whilst the final address (field 13) supplies a high bit to **MemConfig** if a late write cycle is required.

The columns to the right of the coding table show the values of each configuration bit for the four sample

external configuration diagrams. Note the inclusion of period E at the end of T6 in some diagrams. This is inserted to bring the start of the next Tstate T1 to coincide with a rising edge of ProcClockOut (page 219).

Wait states **W** have been added to show the effect of them on strobe timing; they are not part of a configuration. In each case which includes wait states, two wait periods are defined. This shows that if a wait state would cause the start of **T5** to coincide with a falling edge of **ProcClockOut**, another period **Tm** is generated by the EMI to force it to coincide with a rising edge of **ProcClockOut**. This coincidence is only necessary if wait states are added, otherwise coincidence with a falling edge is permitted. Any configuration memory access is only permitted to be extended using wait, up to a total of 14 **ClockIn** periods.

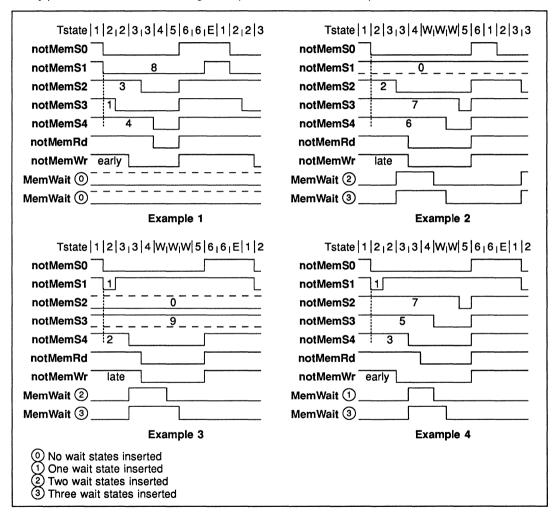


Figure 8.11 IMS T800 external configuration

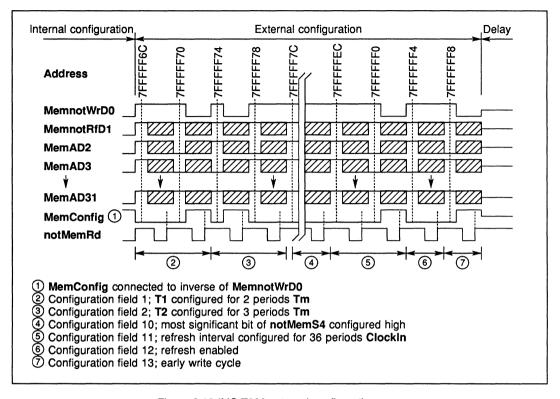


Figure 8.12 IMS T800 external configuration scan

Scan	MemAD			Exa	mple	diag	ram
cycle	address	Field	Function	1	2	3	4
1	7FFFFF6C	1	T1 least significant bit	0	0	0	0
2	7FFFFF70	1	T1 most significant bit	0	0	0	0
3	7FFFFF74	2	T2 least significant bit	1	0	0	1
4	7FFFFF78	2	T2 most significant bit	0	0	0	0
5 6 7	7FFFFF7C	3	T3 least significant bit	1	1	1	1
6	7FFFFF80	3	T3 most significant bit	0	0	0	0
	7FFFFF84	4	T4 least significant bit	0	0	0	0
8	7FFFFF88	4	T4 most significant bit	0	0	0	0
9	7FFFFF8C	5	T5 least significant bit	0	0	0	0
10	7FFFFF90	5	T5 most significant bit	0	0	0	0
11	7FFFFF94	6	T6 least significant bit	1	0	1	1
12	7FFFFF98	6	T6 most significant bit	0	0	0	0
13	7FFFFF9C	7	notMemS1 least significant bit	0	0	1	1
14	7FFFFFA0	7		0	0	0	0
15	7FFFFFA4	7		0	0	0	0
16	7FFFFA8	7		1	0	0	0
17	7FFFFFAC	7	notMemS1 most significant bit	0	0	0	0
18	7FFFFB0	8	notMemS2 least significant bit	1	0	0	1
19	7FFFFB4	8		1	1	0	1
20	7FFFFB8	8		0	0	0	1
21	7FFFFBC	8		0	0	0	0
22	7FFFFC0	8	notMemS2 most significant bit	0	0	0	0
23	7FFFFFC4	9	notMemS3 least significant bit	1	1	1	1
24	7FFFFFC8	9	-	0	1	0	0
25	7FFFFFCC	9		0	1	0	1
26	7FFFFFD0	9		0	0	1	0
27	7FFFFFD4	9	notMemS3 most significant bit	0	0	0	0
28	7FFFFD8	10	notMemS4 least significant bit	0	0	0	1
29	7FFFFDC	10	1	0	1	1	1
30	7FFFFE0	10		1	1	0	0
31	7FFFFE4	10	· · · · · · · · · · · · · · · · · · ·	0	0	0	0
32	7FFFFE8	10	notMemS4 most significant bit	0	0	0	0
33	7FFFFEC	11	Refresh Interval least significant bit	-	-	-	-
34	7FFFFFF0	11	Refresh Interval most significant bit	-	-	-	-
35	7FFFFFF4	12	Refresh Enable	-	-	-	-
36	7FFFFF8	13	Late Write	0	1	1	0

Refresh interval	Interval in μs	Field 11 encoding	Complete cycle (mS)
18	3.6	00	0.922
36	7.2	01	1.843
54	10.8	10	2.765
72	14.4	1 11	3.686

Table 8.8 IMS T800 memory refresh configuration coding

Refresh intervals are in periods of ClockIn and ClockIn frequency is 5 MHz:

Interval = 18 \* 200 = 3600 ns

Refresh interval is between successive incremental refresh addresses. Complete cycles are shown for 256 row DRAMS.

Table 8.9 Memory configuration

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTE
TMCVRdH	Memory configuration data setup	25			ns	
TRdHMCX	Memory configuration data hold	0			ns	
TS0LRdH	notMemS0 to configuration data read	<b>a</b> -12		a+12	ns	1

#### Notes

1 a is 16 periods Tm.

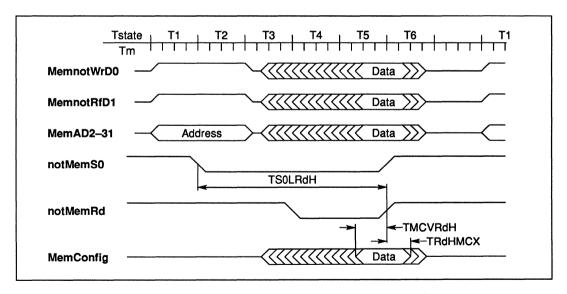


Figure 8.13 IMS T800 external configuration read cycle timing

#### 8.11 notMemRf

The IMS T800 can be operated with memory refresh enabled or disabled. The selection is made during memory configuration, when the refresh interval is also determined. Refresh cycles do not interrupt internal memory accesses, although the internal addresses cannot be reflected on the external bus during refresh.

When refresh is disabled no refresh cycles occur. During the post-**Reset** period eight dummy refresh cycles will occur with the appropriate timing but with no bus or strobe activity.

A refresh cycle uses the same basic external memory timing as a normal external memory cycle, except that it starts two periods **Tm** before the start of **T1**. If a refresh cycle is due during an external memory access, it will be delayed until the end of that external cycle. Two extra periods **Tm** (periods **R** in the diagram) will then be inserted between the end of **T6** of the external memory cycle and the start of **T1** of the refresh cycle itself. The refresh address and various external strobes become active approximately one period **Tm** before **T1**. Bus signals are active until the end of **T2**, whilst **notMemRf** remains active until the end of **T6**.

For a refresh cycle, **MemnotRfD1** goes low before **notMemRf** goes low and **MemnotWrD0** goes high with the same timing as **MemnotRfD1**. All the address lines share the same timing, but only **MemAD2-11** give the refresh address. **MemAD12-30** stay high during the address period, whilst **MemAD31** remains low. Refresh cycles generate strobes **notMemS0-4** with timing as for a normal external cycle, but **notMemRd** and **notMemWrB0-3** remain high. **MemWait** operates normally during refresh cycles.

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTE
TRfLRfH	Refresh pulse width low	<b>a</b> -2		<b>a</b> +9	ns	1
TRaVS0L	Refresh address setup before notMemS0	<b>b</b> -12			ns	
TRfLS0L	Refresh indicator setup before notMemS0	b-4	b	<b>b</b> +6	ns	2

Table 8.10 Memory refresh

#### Notes

- 1 a is total Tmx+Tm.
- 2 b is total T1+Tm where T1 can be from one to four periods Tm in length.

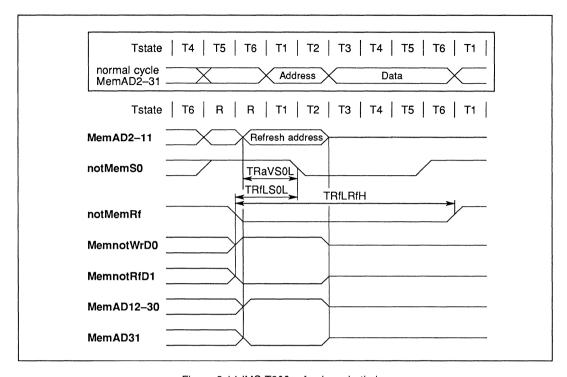


Figure 8.14 IMS T800 refresh cycle timing

## 8.12 MemWait

Taking MemWait high with the timing shown will extend the duration of T4. MemWait is sampled relative to the falling edge of ProcClockOut during a T3 period, and should not change state in this region. By convention, notMemS4 is used to synchronize wait state insertion. If this or another strobe is used, its delay should be such as to take the strobe low an even number of periods Tm after the start of T1, to coincide with a rising edge of ProcClockOut.

**MemWait** may be kept high indefinitely, although if dynamic memory refresh is used it should not be kept high long enough to interfere with refresh timing. **MemWait** operates normally during all cycles, including refresh and configuration cycles. It does not affect internal memory access in any way.

If the start of **T5** would coincide with a falling edge of **ProcClockOut** an extra wait period **Tm** (**EW**) is generated by the EMI to force coincidence with a rising edge. Rising edge coincidence is only forced if wait states are added, otherwise coincidence with a falling edge is permitted.

Table 8.11 Memory wait

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTE
TPCLWtH	Wait setup	-(0.5 <b>Tm</b> +9)			ns	1,2
TPCLWtL	Wait hold	0.5 <b>Tm</b> +10			ns	1,2
TWtLWtH	Delay before re-assertion of Wait	2Tm				

#### Notes

- 1 ProcClockOut load should not exceed 50pf.
- 2 If wait period exceeds refresh interval, refresh cycles will be lost.

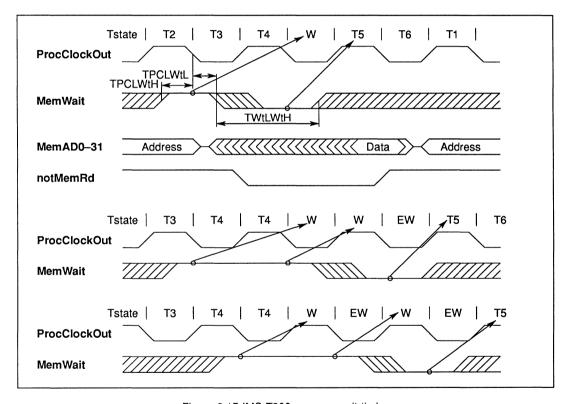


Figure 8.15 IMS T800 memory wait timing

## 8.13 MemReg. MemGranted

Direct memory access (DMA) can be requested at any time by taking the asynchronous **MemReq** input high. The transputer samples **MemReq** during the final period **Tm** of **T6** of both refresh and external memory cycles. To guarantee taking over the bus immediately following either, **MemReq** must be set up at least two periods **Tm** before the end of **T6**. In the absence of an external memory cycle, **MemReq** is sampled during every low period of **ProcClockOut**. The address bus is tristated two periods **Tm** after the **ProcClockOut** rising edge which follows the sample. **MemGranted** is asserted one period **Tm** after that.

Removal of **MemReq** is sampled during each low period of **ProcClockOut** and **MemGranted** is removed synchronously with the next falling edge of **ProcClockOut**. If accurate timing of DMA is required, **MemReq** should be set low coincident with a falling edge of **ProcClockOut**. Further external bus activity, either refresh, external cycles or reflection of internal cycles, will commence at the next rising edge of **ProcClockOut**.

Strobes are left in their inactive states during DMA. DMA cannot interrupt a refresh or external memory cycle, and outstanding refresh cycles will occur before the bus is released to DMA. DMA does not interfere with internal memory cycles in any way, although a program running in internal memory would have to wait for the end of DMA before accessing external memory. DMA cannot access internal memory. If DMA extends longer than one refresh interval (Memory Refresh Configuration Coding, table 8.8), the DMA user becomes responsible for refresh. DMA may also inhibit an internally running program from accessing external memory.

DMA allows a bootstrap program to be loaded into external RAM ready for execution after reset. If MemReq is held high throughout reset, MemGranted will be asserted before the bootstrap sequence begins. MemReq must be high at least one period TDCLDCL of ClockIn before Reset. The circuit should be designed to ensure correct operation if Reset could interrupt a normal DMA cycle.

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTE
TMRHMGH	Memory request response time	4 <b>Tm</b> -2ns		7 <b>Tm</b> +7ns		1
TMRLMGL	Memory request end response time	2 <b>Tm</b> -2ns		5 <b>Tm</b> +22ns		
TADZMGH	Bus tristate before memory granted	Tm-2ns		Tm+22ns		
TMGLADV	Bus active after end of memory granted	-10ns		Tm+2ns		

Table 8.12 Memory request

#### Notes

1 These values assume no external memory cycle is in progress. If an external cycle is active, maximum time could be (1 EMI cycle Tmx)+(1 refresh cycle TRfLRfH)+(6 periods Tm).

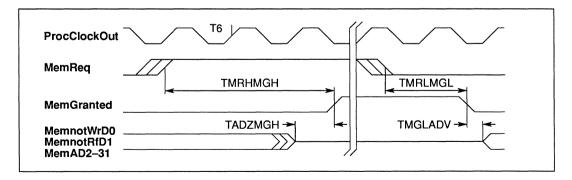


Figure 8.16 IMS T800 memory request timing

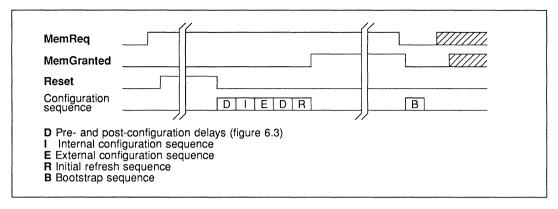


Figure 8.17 IMS T800 DMA sequence at reset

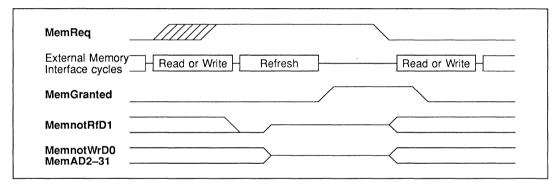


Figure 8.18 IMS T800 operation of MemReg, MemGranted with external, refresh memory cycles

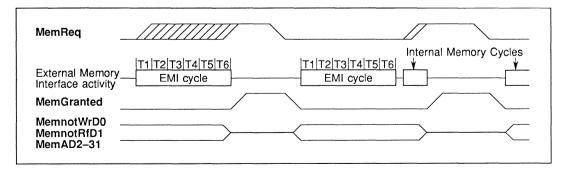


Figure 8.19 IMS T800 operation of MemReq, MemGranted with external, internal memory cycles

## 9 Events

EventReq and EventAck provide an asynchronous handshake interface between an external event and an internal process. When an external event takes EventReq high the external event channel (additional to the external link channels) is made ready to communicate with a process. When both the event channel and the process are ready the processor takes EventAck high and the process, if waiting, is scheduled. EventAck is removed after EventReq goes low.

Only one process may use the event channel at any given time. If no process requires an event to occur **EventAck** will never be taken high. Although **EventReq** triggers the channel on a transition from low to high, it must not be removed before **EventAck** is high. **EventReq** should be low during **Reset**; if not it will be ignored until it has gone low and returned high. **EventAck** is taken low when **Reset** occurs.

If the process is a high priority one and no other high priority process is running, the latency is as described on page 197. Setting a high priority task to wait for an event input allows the user to interrupt a transputer program running at low priority. The time taken from asserting **EventReq** to the execution of the microcode interrupt handler in the CPU is four cycles. The following functions take place during the four cycles:

- Cycle 1 Sample EventReg at pad on the rising edge of ProcClockOut and synchronise.
- **Cycle 2** Edge detect the synchronised **EventReq** and form the interrupt request.
- Cycle 3 Sample interrupt vector for microcode ROM in the CPU.
- **Cycle 4** Execute the interrupt routine for Event rather than the next instruction.

Table 9.1 Event

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTE
TVHKH	Event request response	0			ns	
TKHVL	Event request hold	0			ns	
TVLKL	Delay before removal of event acknowledge	0		6 <b>Tm</b> +7ns		
TKLVH	Delay before re-assertion of event request	0			ns	

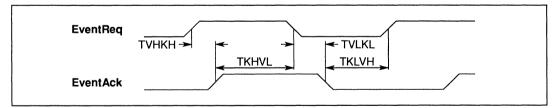


Figure 9.1 IMS T800 event timing

#### 10 Links

Four identical INMOS bi-directional serial links provide synchronized communication between processors and with the outside world. Each link comprises an input channel and output channel. A link between two transputers is implemented by connecting a link interface on one transputer to a link interface on the other transputer. Every byte of data sent on a link is acknowledged on the input of the same link, thus each signal line carries both data and control information.

The quiescent state of a link output is low. Each data byte is transmitted as a high start bit followed by a one bit followed by eight data bits followed by a low stop bit. The least significant bit of data is transmitted first. After transmitting a data byte the sender waits for the acknowledge, which consists of a high start bit followed by a zero bit. The acknowledge signifies both that a process was able to receive the acknowledged data byte and that the receiving link is able to receive another byte. The sending link reschedules the sending process only after the acknowledge for the final byte of the message has been received.

The IMS T800 links allow an acknowledge packet to be sent before the data packet has been fully received. This overlapped acknowledge technique is fully compatible with all other INMOS transputer links.

The IMS T800 links support the standard INMOS communication speed of 10 Mbits/sec. In addition they can be used at 5 or 20 Mbits/sec. Links are not synchronised with ClockIn or ProcClockOut and are insensitive to their phases. Thus links from independently clocked systems may communicate, providing only that the clocks are nominally identical and within specification.

Links are TTL compatible and intended to be used in electrically quiet environments, between devices on a single printed circuit board or between two boards via a backplane. Direct connection may be made between devices separated by a distance of less than 300 millimetres. For longer distances a matched 100 ohm transmission line should be used with series matching resistors **RM**. When this is done the line delay should be less than 0.4 bit time to ensure that the reflection returns before the next data bit is sent.

Buffers may be used for very long transmissions. If so, their overall propagation delay should be stable within the skew tolerance of the link, although the absolute value of the delay is immaterial.

Link speeds can be set by LinkSpecial, Link0Special and Link123Special. The link 0 speed can be set independently. Table 10.1 shows uni-directional and bi-directional data rates in Kbytes/sec for each link speed; LinknSpecial is to be read as Link0Special when selecting link 0 speed and as Link123Special for the others. Data rates are quoted for a transputer using internal memory, and will be affected by a factor depending on the number of external memory accesses and the length of the external memory cycle.

Link	Linkn		Kbyte	s/sec
Special	Special	Mbits/sec	Uni	Bi
0	0	10	910	1250
0	1	5	450	670
1	0	10	910	1250
1	1	20	1740	2350

Table 10.1 Speed Settings for Transputer Links

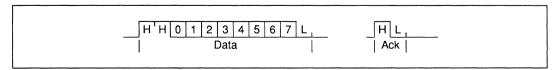


Figure 10.1 IMS T800 link data and acknowledge packets

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTE
TJQr	LinkOut rise time			20	ns	1
TJQf	LinkOut fall time		Ì	10	ns	1
TJDr	LinkIn rise time			20	ns	1 1
TJDf	LinkIn fall time			20	ns	1
TJQJD	Buffered edge delay	0			ns	
TJBskew	Variation in TJQJD 20 Mbits/s			3	ns	2
i	10 Mbits/s		1	10	ns	2
	5 Mbits/s			30	ns	2
CLIZ	LinkIn capacitance @ f=1MHz		ļ	7	pF	1
CLL	LinkOut load capacitance			50	pF	
RM	Series resistor for 100Ω transmission line		56		ohms	

Table 10.2 Link

#### Notes

- 1 These paramters are sampled, but are not 100% tested.
- 2 This is the variation in the total delay through buffers, transmission lines, differential receivers etc., caused by such things as short term variation in supply voltages and differences in delays for rising and falling edges.

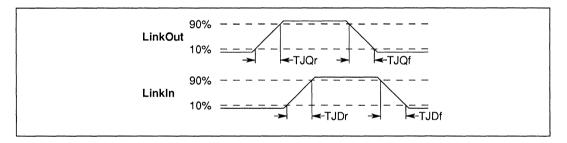


Figure 10.2 IMS T800 link timing

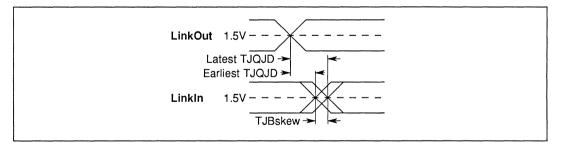


Figure 10.3 IMS T800 buffered link timing

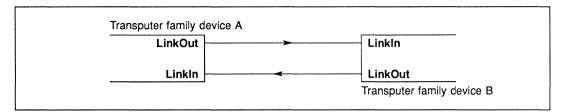


Figure 10.4 IMS T800 Links directly connected

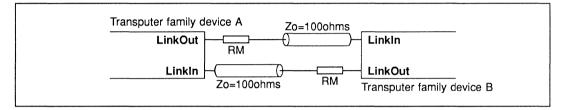


Figure 10.5 IMS T800 Links connected by transmission line

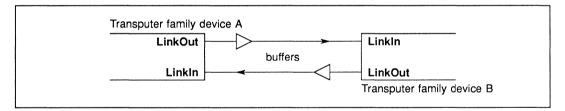


Figure 10.6 IMS T800 Links connected by buffers

## 11 Electrical specifications

#### 11.1 DC electrical characteristics

Table 11.1 Absolute maximum ratings

SYMBOL	PARAMETER	MIN	MAX	UNITS	NOTE
VCC	DC supply voltage	0	7.0	٧	1,2,3
VI, VO	Voltage on input and output pins	-0.5	VCC+0.5	V	1,2,3
	Input current		±25	mA	4
OSCT	Output short circuit time (one pin)	İ	1	s	2
TS	Storage temperature	-65	150	°C	2
TA	Ambient temperature under bias	-55	125	°C	2
PDmax	Maximum allowable dissipation	1	2	W	

#### Notes

- 1 All voltages are with respect to GND.
- 2 This is a stress rating only and functional operation of the device at these or any other conditions beyond those indicated in the operating sections of this specification is not implied. Stresses greater than those listed may cause permanent damage to the device. Exposure to absolute maximum rating conditions for extended periods may affect reliability.
- 3 This device contains circuitry to protect the inputs against damage caused by high static voltages or electrical fields. However, it is advised that normal precautions be taken to avoid application of any voltage higher than the absolute maximum rated voltages to this high impedance circuit. Unused inputs should be tied to an appropriate logic level such as VCC or GND.
- 4 The input current applies to any input or output pin and applies when the voltage on the pin is between GND and VCC.

Table 11.2 Operating conditions

SYMBOL	PARAMETER	MIN	MAX	UNITS	NOTE
VCC	DC supply voltage	4.75	5.25	V	1
VI, VO	Input or output voltage	0	VCC	V	1,2
CL	Load capacitance on any pin		60	pF	
TA	Operating temperature range IMS T800-S	0	70	°C	3
TA	Operating temperature range IMS T800-M	-55	125	°C	3

#### Notes

- 1 All voltages are with respect to GND.
- 2 Excursions beyond the supplies are permitted but not recommended; see DC characteristics.
- 3 Air flow rate 400 linear ft/min transverse air flow.

SYMBOL	PARAME	TER	MIN	MAX	UNITS	NOTE
VIH	High level input voltage		2.0	VCC+0.5	V	1,2
VIL	Low level input voltage		-0.5	0.8	l v	1,2
11	Input current	@ GND <vi<vcc< td=""><td></td><td>±10</td><td><math>\mu</math>A</td><td>1,2</td></vi<vcc<>		±10	$\mu$ A	1,2
VOH	Output high voltage	@ IOH=2mA	VCC-1		ľV	1,2
VOL	Output low voltage	@ IOL=4mA		0.4	l v	1,2
IOS	Output short circuit current	@ GND <vo<vcc< td=""><td>36</td><td>65</td><td>mA</td><td>1,2,3,6</td></vo<vcc<>	36	65	mA	1,2,3,6
	-		65	100	mA	1,2,4,6
IOZ	Tristate output current	@ GND <vo<vcc< td=""><td></td><td>±10</td><td>μA</td><td>1,2</td></vo<vcc<>		±10	μA	1,2
PD	Power dissipation			1.2	w	2,5
CIN	Input capacitance	@ f=1MHz		7	pF	6
COZ	Output capacitance	@ f=1MHz		10	pF	6

Table 11.3 DC characteristics

#### Notes

- 1 All voltages are with respect to GND.
- 2 Parameters for IMS T800-S measured at 4.75V<VCC<5.25V and 0°C<TA<70°C. Input clock frequency = 5 MHz.</p>
- 3 Current sourced from non-link outputs.
- 4 Current sourced from link outputs.
- 5 Power dissipation varies with output loading and program execution. Power dissipation for processor operating at 20 MHz.
- 6 This parameter is sampled and not 100% tested.

## 11.2 Equivalent circuits

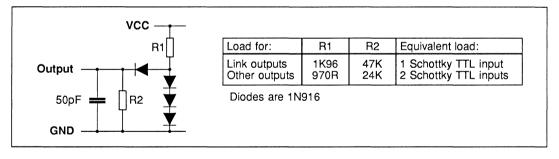


Figure 11.1 Load circuit for AC measurements

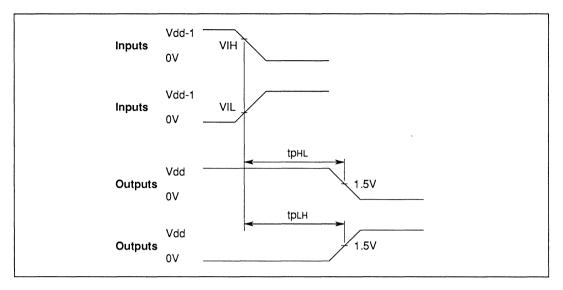


Figure 11.2 AC measurements timing waveforms

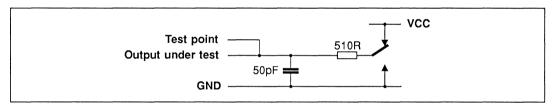


Figure 11.3 Tristate load circuit for AC measurements

## 11.3 AC timing characteristics

Table 11.4 Input, output edges

SYMBOL	PARAMETER	MIN	MAX	UNITS	NOTE
TDr	Input rising edges	2	20	ns	1,2
TDf	Input falling edges	2	20	ns	1,2
TQr	Output rising edges		25	ns	1
TQf	Output falling edges	Ì	15	ns	1 1
TS0LaHZ	Address high to tristate	a	<b>a</b> +6	ns	3
TS0LaLZ	Address low to tristate	а	<b>a</b> +6	ns	3

#### Notes

- 1 Non-link pins; see section on links.
- 2 All inputs except Clockin; see section on Clockin.
- 3 a is T2 where T2 can be from one to four periods Tm in length. Address lines include MemnotWrD0, MemnotRfD1, MemAD2-31.

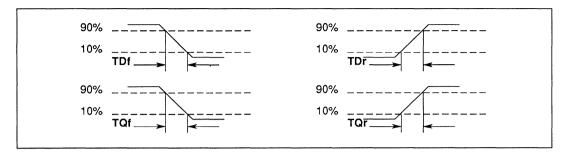


Figure 11.4 IMS T800 input and output edge timing

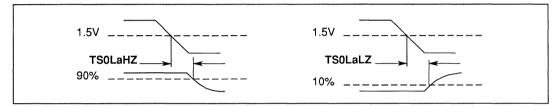
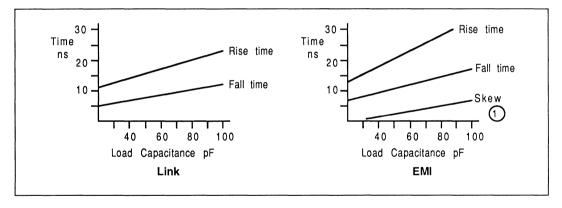


Figure 11.5 IMS T800 tristate timing relative to notMemS0



Notes

Figure 11.6 Typical rise/fall times

1 Skew is measured between **notMemS0** with a standard load (2 Schottky TTL inputs and 30pF) and **notMemS0** with a load of 2 Schottky TTL inputs and varying capacitance.

## 11.4 Power rating

Internal power dissipation  $P_{INT}$  of transputer and peripheral chips depends on VCC, as shown in figure 11.7.  $P_{INT}$  is substantially independent of temperature.

Total power dissipation  $P_D$  of the chip is

$$P_D = P_{INT} + P_{IO}$$

where  $P_{IO}$  is the power dissipation in the input and output pins; this is application dependent.

Internal working temperature  $T_J$  of the chip is

$$T_J = T_A + \theta J_A * P_D$$

where  $T_A$  is the external ambient temperature in °C and  $\theta J_A$  is the junction-to-ambient thermal resistance in °C/W.  $\theta J_A$  for each package is given in the Packaging Specifications section.

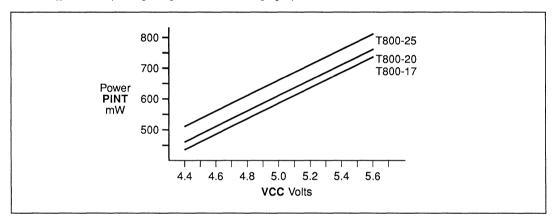


Figure 11.7 IMS T800 internal power dissipation vs VCC

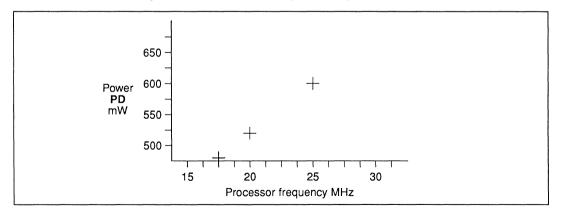


Figure 11.8 IMS T800 typical power dissipation with processor speed

## 12 Performance

The performance of the transputer is measured in terms of the number of bytes required for the program, and the number of (internal) processor cycles required to execute the program. The figures here relate to OCCam programs. For the same function, other languages should achieve approximately the same performance as occam.

With transputers incorporating an FPU, this type of performance calculation is straight forward when considering only integer data types. However, when floating point calculations using the **REAL32** and **REAL64** data types are present in the program, complications arise due to the concurrency inherent in the transputer's design whereby integer calculations can be overlapped with floating point calculations. A more comprehensive guide to the impact of this concurrency on transputer performance can be found in the *Transputer Instruction Set - A Compiler Writers' Guide*.

#### 12.1 Performance overview

These figures are averages obtained from detailed simulation, and should be used only as an initial guide; they assume operands are of type **INT**. The abbreviations in table 12.1 are used to represent the quantities indicated. In the replicator section of the table, figures in braces {} are not necessary if the number of replications is a compile time constant. To estimate performance, add together the time for the variable references and the time for the operation.

Table 12.1 Key to performance table

np	number of component processes
ne	number of processes earlier in queue
r	1 if INT parameter or array parameter, 0 if not
ts	number of table entries (table size)
w	width of constant in nibbles
р	number of places to shift
Eg	expression used in a guard
Et	timer expression used in a guard
Tb	most significant bit set of multiplier ((-1) if the multiplier is 0)
Tbp	most significant bit set in a positive multiplier when counting from zero ((-1) if the multiplier is 0)
Tbc	most significant bit set in the two's complement of a negative multiplier
nsp	Number of scalar parameters in a procedure
nap	Number of array parameters in a procedure

Table 12.2 Performance

	Size (bytes)	Time (cycles)
Names		
variables		
in expression	1.1+ <b>r</b>	2.1+2( <b>r</b> )
assigned to or input to	1.1+ <b>r</b>	1.1+(r)
in PROC or FUNCTION call,		
corresponding to an INT parameter	1.1+ <b>r</b>	1.1+(r)
channels	1.1	2.1
Array Variables (for single dimension arrays)		
constant subscript	0	0
variable subscript	5.3	7.3
expression subscript	5.3	7.3
Declarations		
CHAN OF protocol	3.1	3.1
[size]CHAN OF protocol	9.4	2.2 + 20.2*size
PROC	body+2	0
Primitives	·	
assignment	0	0
input	4	26.5
output	1	26
STOP	2	25
SKIP	0	0
Arithmetic operators		
+ -	1	1
*	2	39
/	2	40
REM	2	38
<b>&gt;&gt;                                   </b>	2	3 <b>+p</b>
Modulo Arithmetic operators		
PLUS	2	2
MINUS	1	1
TIMES (fast multiply, positive operand)	1	4+Tbp
TIMES (fast multiply, negative operand)	1	5 <b>+Tbc</b>
Boolean operators		
OR	4	8
AND NOT	1	2
Comparison operators		
= constant	0	1
= variable	2	3
<> constant	1	3
<> variable	3	5
> <	1	2
>= <=	2	4
Bit operators		
/\ \/ <b>&gt;&lt;</b> ~	2	2
Expressions		
constant in expression	w	w
check if error	4	6

12 Performance 251

Table 12.3 Performance

	Size (bytes)	Time (cycles)
Timers		
timer input	2	3
timer AFTER	1	
if past time	2	4
with empty timer queue	2 2 2	31
non-empty timer queue	2	38+ <b>ne</b> +9
ALT (timer)		
with empty timer queue	6	52
non-empty timer queue	6	59+ <b>ne</b> *9
timer alt guard	8+2 <b>Eg</b> +2 <b>Et</b>	34+2 <b>Eg</b> +2 <b>Et</b>
Constructs		
SEQ	0	0
IF	1.3	1.4
if guard	3	4.3
ALT (non timer)	6	26
alt channel guard	10.2+2 <b>Eg</b>	20+2 <b>Eg</b>
skip alt guard	8+2 <b>Eg</b>	10+2 <b>Eg</b>
PAR	11.5+( <b>np</b> -1)*7.5	19.5+( <b>np</b> -1)*30.5
WHILE	4	12
Procedure or function call		
	3.5+(nsp-2)*1.1	16.5+(nsp-2)*1.1
	+nap*2.3	+nap+2.3
Replicators		
replicated SEQ	7.3{+5.1}	(-3.8)+15.1*count{+7.1}
replicated IF	12.3{+5.1}	(-2.6)+19.4*count{+7.1}
replicated ALT	24.8{+10.2}	25.4+33.4*count{+14.2}
replicated timer ALT	24.8{+10.2}	62.4+33.4*count{+14.2}
replicated PAR	39.1{+5.1}	(-6.4)+70.9*count(+7.1)

## 12.2 Fast multiply, TIMES

The IMS T800 has a fast integer multiplication instruction *product*. For a positive multiplier its execution time is 4+**Tbp** cycles, and for a negative multiplier 5+**Tbc** cycles (table 12.1). The time taken for a multiplication by zero is 3 cycles.

Implementations of high level languages on the transputer may take advantage of this instruction. For example, the OCCAM modulo arithmetic operator **TIMES** is implemented by the instruction and the right-hand operand is treated as the multiplier. The fast multiplication instruction is also used in high level language implementations for the multiplication implicit in multi-dimensional array access.

#### 12.3 Arithmetic

A set of functions are provided within the development system to support the efficient implementation of multiple length integer arithmetic. In the IMS T800, floating point arithmetic is taken care of by the FPU. In table 12.4 n gives the number of places shifted and all arguments and results are assumed to be local. Full details of these functions are provided in the OCCAM reference manual, supplied as part of the development system and available as a separate publication.

When calculating the execution time of the predefined maths functions, no time needs to be added for calling overhead. These functions are compiled directly into special purpose instructions which are designed to support the efficient implementation of multiple length integer arithmetic and floating point arithmetic.

Function		Cycles	+ cycles for parameter access †
LONGADD		2	7
LONGSUM		3	8
LONGSUB		2	7
LONGDIFF		3	8
LONGPROD		34	8
LONGDIV		36	8
SHIFTRIGHT	(n<32)	4+n	8
	( <b>n</b> >=32)	<b>n</b> -27	
SHIFTLEFT	(n<32)	4+ <b>n</b>	8
	(n>=32)	<b>n</b> -27	
NORMALISE	(n<32)	<b>n</b> +6	7
	(n>=32)	n-25	
	( <b>n</b> =64)	4	
ASHIFTRIGHT		SHIFTRIGHT+2	5
ASHIFTLEFT		SHIFTLEFT+4	5
ROTATERIGHT		SHIFTRIGHT	7
ROTATELEFT		SHIFTLEFT	7
FRACMUL		LONGPROD+4	5

Table 12.4 Arithmetic performance

<sup>†</sup> Assuming local variables.

12 Performance 253

## 12.4 Floating point operations

All references to **REAL32** or **REAL64** operands within programs compiled for the IMS T800 normally produce the following performance figures.

· · · · · · · · · · · · · · · · · · ·					
	Size (bytes)	REAL32 Time (cycles)	REAL64 Time (cycles)		
Names					
variables					
in expression	3.1	3 3	5 5		
assigned to or input to	3.1	3	5		
in PROC or FUNCTION call,					
corresponding to a REAL					
parameter	1.1+r	1.1+r	1.1+r		
Arithmetic operators					
+ -	2	7	7		
*	2 2 2 11	11	20		
/	2	17	32		
REM	11	19	34		
Comparison operators					
=	2	4	4		
<>	2 3 2 3	6 5 7	6		
> <	2	5	6 5 7		
>= <=	3	7	7		
Conversions					
REAL32 to -	2		3		
REAL64 to -	2	6			
To INT32 from -	2 2 5 18 3	9 32	9		
To INT64 from -	18	32	32		
INT32 to -	3	7	7		
INT64 to -	14	24	22		

Table 12.5 Floating point performance

## 12.4.1 Floating point functions

These functions are provided by the development system. They are compiled directly into special purpose instructions designed to support the efficient implementation of some of the common mathematical functions of other languages. The functions provide ABS and SQRT for both REAL32 and REAL64 operand types.

		+ cycles for parameter access †			
Function	Cycles	REAL32	REAL 64		
ABS	2	8			
SQRT	118	8			
DABS	2		12		
DSQRT	244		12		

Table 12.6 IMS T800 floating point arithmetic performance

<sup>†</sup> Assuming local variables.

## 12.4.2 Special purpose functions and procedures

The functions and procedures given in tables 12.8 and 12.9 are provided by the development system to give access to the special instructions available on the IMS T800. Table 12.7 shows the key to the table.

Table 12.7 Key to special performance table

Tb	most significant bit set in the word counting from zero
n	number of words per row (consecutive memory locations)
r	number of rows in the two dimensional move
	number of bits to reverse

Table 12.8 Special purpose functions performance

Function	Cycles	+ cycles for parameter access †
BITCOUNT	2+ <b>Tb</b>	2
CRCBYTE	11	8
CRCWORD	35	8
BITREVNBIT	5+nr	4
BITREVWORD	36	2

t Assuming local variables.

Table 12.9 Special purpose procedures performance

Procedure	Cycles	+ cycles for parameter access †
MOVE2D	8+(2n+23)*r	8
DRAW2D	8+(2n+23)*r	8
CLIP2D	8+(2n+23)*r	8

† Assuming local variables.

## 12.5 Effect of external memory

Extra processor cycles may be needed when program and/or data are held in external memory, depending both on the operation being performed, and on the speed of the external memory. After a processor cycle which initiates a write to memory, the processor continues execution at full speed until at least the next memory access.

Whilst a reasonable estimate may be made of the effect of external memory, the actual performance will depend upon the exact nature of the given sequence of operations.

External memory is characterized by the number of extra processor cycles per external memory cycle, denoted as **e**. For the IMS T800, with the fastest external memory the value of **e** is 2; a typical value for a large external memory is 5.

If a program is stored in external memory, and **e** has the value 2 or 3, then no extra cycles need be estimated for linear code sequences. For larger values of **e**, the number of extra cycles required for linear code sequences may be estimated at (**e**-3)/4. A transfer of control may be estimated as requiring **e**+3 cycles.

These estimates may be refined for various constructs. In table 12.10 n denotes the number of components in a construct. In the case of IF, the n'th conditional is the first to evaluate to TRUE, and the costs include the

12 Performance 255

costs of the conditionals tested. The number of bytes in an array assignment or communication is denoted by  ${\bf b}$ .

	IMS T800		
	Program off ship	Data off chip	
Boolean expressions	<b>e</b> -2	0	
IF	3 <b>en</b> -8	en	
Replicated IF	(6 <b>e</b> -4) <b>n</b> +7	(5 <b>e</b> -2) <b>n</b> +8	
Replicated SEQ	(3 <b>e</b> -3) <b>n</b> +2	(4 <b>e</b> -2)n	
PAR	(3 <b>e</b> -1) <b>n</b> +8	3 <b>en</b> +4	
Replicated PAR	(10 <b>e</b> -8) <b>n</b> +8	16 <b>en</b> -12	
ALT	(2 <b>e</b> -4) <b>n</b> +6 <b>e</b>	(2e-2)n+10e-8	
Array assignment and communication in one transputer	0	max (2e, e(b/2))	

Table 12.10 External memory performance

The following simulation results illustrate the effect of storing program and/or data in external memory. The results are normalized to 1 for both program and data on chip. The first program (Sieve of Erastosthenes) is an extreme case as it is dominated by small, data access intensive loops; it contains no concurrency, communication, or even multiplication or division. The second program is the pipeline algorithm for Newton Raphson square root computation.

	Program	e=2	e=3	e=4	e=5	On chip
Program off chip	1	1.3	1.5	1.7	1.9	1
	2	1.1	1.2	1.2	1.3	1
Data off chip	1	1.5	1.8	2.1	2.3	1
·	2	1.2	1.4	1.6	1.7	1
Program and data off chip	1	1.8	2.2	2.7	3.2	1
	2	1.3	1.6	1.8	2.0	1

Table 12.11 IMS T800 external memory performance

## 12.6 Interrupt latency

If the process is a high priority one and no other high priority process is running, the latency is as described in table 12.12. The timings given are in full processor cycles **TPCLPCL**; the number of **Tm** states is also given where relevant. Maximum latency assumes all memory accesses are internal ones.

	Турі	cal	Maximum		
	TPCLPCL	Tm	TPCLPCL	Tm	
IMS T800 with FPU in use	19	38	78	156	
IMS T800 with FPU not in use	19	38	58	116	

Table 12.12 Interrupt latency

## 13 Package specifications

## 13.1 84 pin grid array package

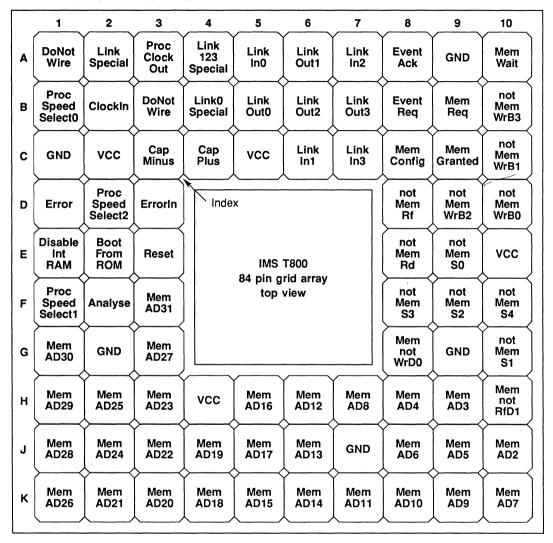


Figure 13.1 IMS T800 84 pin grid array package pinout

Figure 13.2 84 pin grid array package dimensions

Table 13.1 84 pin grid array package dimensions

	Millimetres		Inches		
DIM	NOM	TOL	NOM	TOL	Notes
Α	26.924	±0.254	1.060	±0.010	
B1	17.019	±0.127	0.670	±0.005	
B2	18.796	±0.127	0.740	±0.005	
С	2.456	±0.278	0.097	±0.011	
D	4.572	±0.127	0.180	±0.005	
E	3.302	±0.127	0.130	±0.005	
F	0.457	±0.025	0.018	$\pm 0.002$	Pin diameter
G	1.143	±0.127	0.045	±0.005	Flange diameter
K	22.860	±0.127	0.900	±0.005	
L	2.540	±0.127	0.100	±0.005	
М	0.508		0.020		Chamfer

Package weight is approximately 7.2 grams

Table 13.2 84 pin grid array package junction to ambient thermal resistance

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTE
$\theta$ JA	At 400 linear ft/min transverse air flow			35	°C/W	

#### 13.2 84 lead quad cerpack package

The leads are unformed to allow the user to form them to specific requirements.

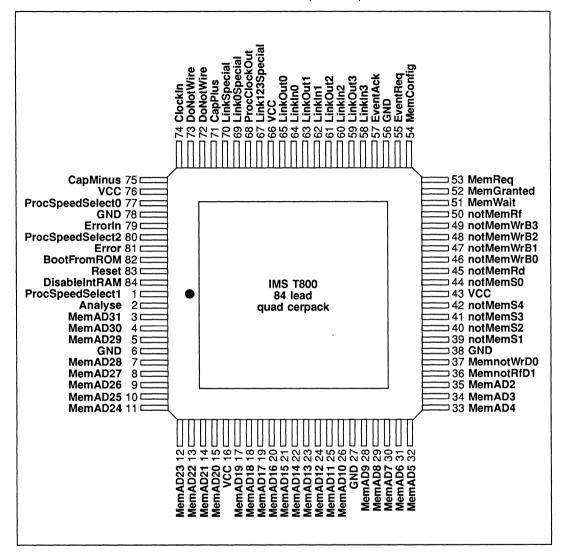


Figure 13.3 IMS T800 84 lead quad cerpack package pinout

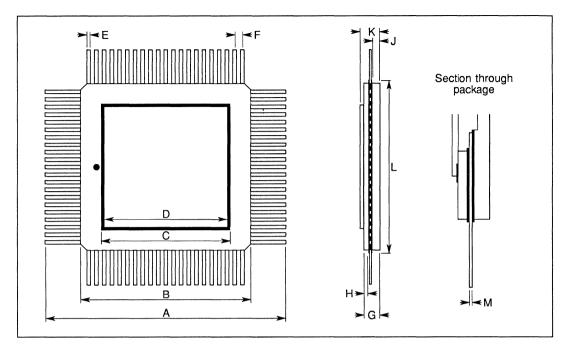


Figure 13.4 84 lead quad cerpack package dimensions

	Millimetres		Inc	Inches		
DIM	NOM	TOL	NOM	TOL	Notes	
Α	38.100	±0.508	1.500	±0.020		
В	26.924	±0.305	1.060	±0.012		
C	20.574	±0.203	0.810	±0.008		
D	19.558	±0.254	0.770	±0.010		
E	0.508		0.020			
F	1.270	±0.051	0.050	±0.002		
G	2.489	±0.305	0.098	±0.012		
Н	0.635	±0.076	0.025	±0.003		
J	1.143	±0.102	0.045	±0.004		
K	3.099		0.122		Max.	
L	27.940		1.100		Мах.	
М	0.178	±0.025	0.007	±0.001		

Table 13.3 84 lead quad cerpack package dimensions

# 14 Ordering

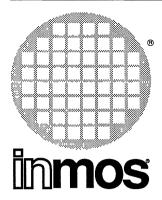
This section indicates the designation of speed and package selections for the various devices. Speed of **ClockIn** is 5 MHz for all parts. Transputer processor cycle time is nominal; it can be calculated more exactly using the phase lock loop factor **PLLx**, as detailed in the external memory section.

For availability contact local INMOS sales office or authorised distributor.

Table 14.1 IMS T800 ordering details

INMOS designation	Processor clock speed	Processor cycle time	PLLx	Package
IMS T800-G17S	17.5 MHz	57 ns	3.5	Ceramic Pin Grid
IMS T800-G20S	20.0 MHz	50 ns	4.0	Ceramic Pin Grid
IMS T800-G25S	25.0 MHz	40 ns	5.0	Ceramic Pin Grid
IMS T800-G30S	30.0 MHz	33 ns	6.0	Ceramic Pin Grid
IMS T800-G17M IMS T800-G20M	17.5 MHz 20.0 MHz	57 ns 50 ns	3.5 4.0	Ceramic Pin Grid MIL Spec Ceramic Pin Grid MIL Spec
IMS T800-Q17M IMS T800-Q20M	17.5 MHz 20.0 MHz	57 ns 50 ns	3.5 4.0	Quad Cerpack MIL Spec Quad Cerpack MIL Spec

The timing parameters in this datasheet are based on full characterisation of the 17 MHz and 20 MHz parts. Data for higher speeds is based on tests on a limited number of samples and may change when full characterisation is completed.



# IMS T425 transputer

# Advance Data

#### **FEATURES**

32 bit architecture

33 ns internal cycle time

30 MIPS (peak) instruction rate

Pin compatible with IMS T805, IMS T800 and IMS T414 Debugging support

4 Kbytes on-chip static RAM

120 Mbytes/sec sustained data rate to internal memory

4 Gbytes directly addressable external memory

40 Mbytes/sec sustained data rate to external memory 630 ns response to interrupts

Four INMOS serial links 5/10/20 Mbits/sec

High performance graphics support with block move instructions

Boot from ROM or communication links

Single 5 MHz clock input

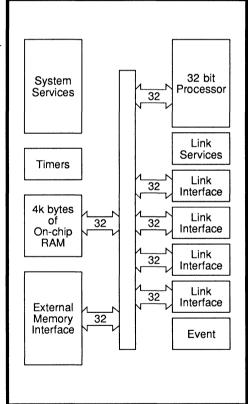
Single +5V ±5% power supply

MIL-STD-883C processing will be available

## **APPLICATIONS**

Artificial intelligence

High speed multi processor systems
High performance graphics processing
Supercomputers
Workstations and workstation clusters
Digital signal processing
Accelerator processors
Distributed databases
System simulation
Telecommunications
Robotics
Fault tolerant systems
Image processing
Pattern recognition



#### 1 Introduction

The IMS T425 transputer is a 32 bit CMOS microcomputer with graphics support. It has 4 Kbytes on-chip RAM for high speed processing, a configurable memory interface and four standard INMOS communication links. The instruction set achieves efficient implementation of high level languages and provides direct support for the OCCAM model of concurrency when using either a single transputer or a network. Procedure calls, process switching and typical interrupt latency are sub-microsecond.

For convenience of description, the IMS T425 operation is split into the basic blocks shown in figure 1.1.

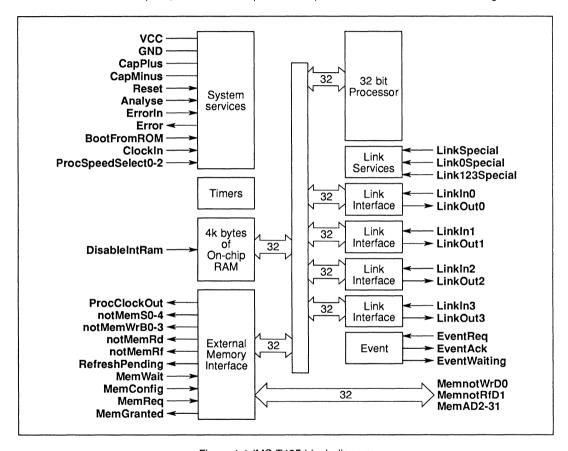


Figure 1.1 IMS T425 block diagram

The processor speed of a device can be pin-selected in stages from 17.5 MHz up to the maximum allowed for the part. A device running at 30 MHz achieves an instruction throughput of 30 MIPS peak and 15 MIPS sustained. The extended temperature version of the device complies with MIL-STD-883C.

High performance graphics support is provided by microcoded block move instructions which operate at the speed of memory. The two-dimensional block move instructions provide for contiguous block moves as well as block copying of either non-zero bytes of data only or zero bytes only. Block move instructions can be used to provide graphics operations such as text manipulation, windowing, panning, scrolling and screen updating.

Cyclic redundancy checking (CRC) instructions are available for use on arbitrary length serial data streams, to provide error detection where data integrity is critical. Another feature of the IMS T425, useful for pattern recognition, is the facility to count bits set in a word.

1 Introduction 263

The IMS T425 can directly access a linear address space of 4 Gbytes. The 32 bit wide memory interface uses multiplexed data and address lines and provides a data rate of up to 4 bytes every 100 nanoseconds (40 Mbytes/sec) for a 30 MHz device. A configurable memory controller provides all timing, control and DRAM refresh signals for a wide variety of mixed memory systems.

System Services include processor reset and bootstrap control, together with facilities for error analysis. Error signals may be daisy-chained in multi-transputer systems.

The standard INMOS communication links allow networks of transputer family products to be constructed by direct point to point connections with no external logic. The IMS T425 links support the standard operating speed of 10 Mbits/sec, but also operate at 5 or 20 Mbits/sec. Each link can transfer data bi-directionally at up to 2.35 Mbytes/sec.

The IMS T425 is pin compatible with the IMS T800 and can be plugged directly into a circuit designed for that device. It has a number of additions to improve hardware interfacing and to facilitate software initialising and debugging. The improvements have been made in an upwards-compatible manner. Software should be recompiled, although no changes to the source code are necessary.

The IMS T425-20 is also pin compatible with the IMS T414-20, as the extra inputs used are all held to ground on the IMS T414. The IMS T425-20 can thus be plugged directly into a circuit designed for a 20 MHz version of the IMS T414.

The transputer is designed to implement the OCCAM language, detailed in the OCCAM Reference Manual, but also efficiently supports other languages such as C, Pascal and Fortran. Access to the transputer at machine level is seldom required, but if necessary refer to the *Transputer Instruction Set - A Compiler Writers' Guide*. The instruction set of the IMS T425 is the same as that of the IMS T800, except that the IMS T800 floating point instructions are replaced by the IMS T414 floating point support instructions.

This data sheet supplies hardware implementation and characterisation details for the IMS T425. It is intended to be read in conjunction with the Transputer Architecture chapter, which details the architecture of the transputer and gives an overview of OCCAM.

# 2 Pin designations

Table 2.1 IMS T425 system services

Pin	In/Out	Function
VCC, GND		Power supply and return
CapPlus, CapMinus		External capacitor for internal clock power supply
Clockin	in	Input clock
ProcSpeedSelect0-2	in	Processor speed selectors
Reset	in	System reset
Error	out	Error indicator
Errorin	in	Error daisychain input
Analyse	in	Error analysis
BootFromRom	in	Boot from external ROM or from link
DisableIntRAM	in	Disable internal RAM

Table 2.2 IMS T425 external memory interface

Pin	In/Out	Function
ProcClockOut	out	Processor clock
MemnotWrD0	in/out	Multiplexed data bit 0 and write cycle warning
MemnotRfD1	in/out	Multiplexed data bit 1 and refresh warning
MemAD2-31	in/out	Multiplexed data and address bus
notMemRd	out	Read strobe
notMemWrB0-3	out	Four byte-addressing write strobes
notMemS0-4	out	Five general purpose strobes
notMemRf	out	Dynamic memory refresh indicator
RefreshPending	out	Dynamic refresh is pending
MemWait	in	Memory cycle extender
MemReq	in	Direct memory access request
MemGranted	out	Direct memory access granted
MemConfig	in	Memory configuration data input

Table 2.3 IMS T425 event

Pin	In/Out	Function
EventReq	in	Event request
EventAck	out	Event request acknowledge
EventWaiting	out	Event input requested by software

Table 2.4 IMS T425 link

Pin	In/Out	Function
Linkln0-3	in	Four serial data input channels
LinkOut0-3	out	Four serial data output channels
LinkSpecial	in	Select non-standard speed as 5 or 20 Mbits/sec
Link0Special	in	Select special speed for Link 0
Link123Special	in	Select special speed for Links 1,2,3

Signal names are prefixed by **not** if they are active low, otherwise they are active high. Pinout details for various packages are given on page 326.

# 3 Processor

The 32 bit processor contains instruction processing logic, instruction and work pointers, and an operand register. It directly accesses the high speed 4 Kbyte on-chip memory, which can store data or program. Where larger amounts of memory or programs in ROM are required, the processor has access to 4 Gbytes of memory via the External Memory Interface (EMI).

## 3.1 Registers

The design of the transputer processor exploits the availability of fast on-chip memory by having only a small number of registers; six registers are used in the execution of a sequential process. The small number of registers, together with the simplicity of the instruction set, enables the processor to have relatively simple (and fast) data-paths and control logic. The six registers are:

The workspace pointer which points to an area of store where local variables are kept.

The instruction pointer which points to the next instruction to be executed.

The operand register which is used in the formation of instruction operands.

The A, B and C registers which form an evaluation stack.

A, B and C are sources and destinations for most arithmetic and logical operations. Loading a value into the stack pushes B into C, and A into B, before loading A. Storing a value from A, pops B into A and C into B.

Expressions are evaluated on the evaluation stack, and instructions refer to the stack implicitly. For example, the *add* instruction adds the top two values in the stack and places the result on the top of the stack. The use of a stack removes the need for instructions to respecify the location of their operands. Statistics gathered from a large number of programs show that three registers provide an effective balance between code compactness and implementation complexity.

No hardware mechanism is provided to detect that more than three values have been loaded onto the stack. It is easy for the compiler to ensure that this never happens.

Any location in memory can be accessed relative to the workpointer register, enabling the workspace to be of any size.

Further register details are given in Transputer Instruction Set - A Compiler Writers' Guide.

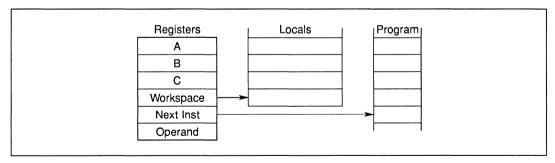


Figure 3.1 Registers

## 3.2 Instructions

The instruction set has been designed for simple and efficient compilation of high-level languages. All instructions have the same format, designed to give a compact representation of the operations occurring most frequently in programs.

Each instruction consists of a single byte divided into two 4-bit parts. The four most significant bits of the byte are a function code and the four least significant bits are a data value.

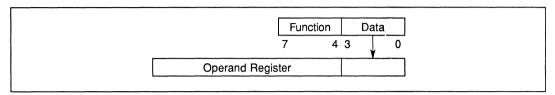


Figure 3.2 Instruction format

#### 3.2.1 Direct functions

The representation provides for sixteen functions, each with a data value ranging from 0 to 15. Ten of these, shown in table 3.1, are used to encode the most important functions.

load constant add constant
load local store local load local pointer
load non-local store non-local
iump conditional iump call

Table 3.1 Direct functions

The most common operations in a program are the loading of small literal values and the loading and storing of one of a small number of variables. The *load constant* instruction enables values between 0 and 15 to be loaded with a single byte instruction. The *load local* and *store local* instructions access locations in memory relative to the workspace pointer. The first 16 locations can be accessed using a single byte instruction.

The *load non-local* and *store non-local* instructions behave similarly, except that they access locations in memory relative to the *A* register. Compact sequences of these instructions allow efficient access to data structures, and provide for simple implementations of the static links or displays used in the implementation of high level programming languages such as OCCAM, C, Fortran, Pascal or ADA.

### 3.2.2 Prefix functions

Two more function codes allow the operand of any instruction to be extended in length; *prefix* and *negative prefix*.

All instructions are executed by loading the four data bits into the least significant four bits of the operand register, which is then used as the instruction's operand. All instructions except the prefix instructions end by clearing the operand register, ready for the next instruction.

The *prefix* instruction loads its four data bits into the operand register and then shifts the operand register up four places. The *negative prefix* instruction is similar, except that it complements the operand register before shifting it up. Consequently operands can be extended to any length up to the length of the operand register by a sequence of prefix instructions. In particular, operands in the range -256 to 255 can be represented using one prefix instruction.

The use of prefix instructions has certain beneficial consequences. Firstly, they are decoded and executed in the same way as every other instruction, which simplifies and speeds instruction decoding. Secondly, they simplify language compilation by providing a completely uniform way of allowing any instruction to take an operand of any size. Thirdly, they allow operands to be represented in a form independent of the processor wordlength.

## 3.2.3 Indirect functions

The remaining function code, *operate*, causes its operand to be interpreted as an operation on the values held in the evaluation stack. This allows up to 16 such operations to be encoded in a single byte instruction. However, the prefix instructions can be used to extend the operand of an *operate* instruction just like any other. The instruction representation therefore provides for an indefinite number of operations.

Encoding of the indirect functions is chosen so that the most frequently occurring operations are represented without the use of a prefix instruction. These include arithmetic, logical and comparison operations such as add, exclusive or and greater than. Less frequently occurring operations have encodings which require a single prefix operation.

## 3.2.4 Expression evaluation

Evaluation of expressions sometimes requires use of temporary variables in the workspace, but the number of these can be minimised by careful choice of the evaluation order.

Program	Mnem	onic
x := 0	ldc	0
	stl	X
x := #24	pfix Idc	2
	ldc	2 4 X
	stl	X
x := y + z	ldl	У
	ldl	Z
	add	
	stl	X

Table 3.2 Expression evaluation

## 3.2.5 Efficiency of encoding

Measurements show that about 70% of executed instructions are encoded in a single byte; that is, without the use of prefix instructions. Many of these instructions, such as *load constant* and *add* require just one processor cycle.

The instruction representation gives a more compact representation of high level language programs than more conventional instruction sets. Since a program requires less store to represent it, less of the memory bandwidth is taken up with fetching instructions. Furthermore, as memory is word accessed the processor will receive four instructions for every fetch.

Short instructions also improve the effectiveness of instruction pre-fetch, which in turn improves processor performance. There is an extra word of pre-fetch buffer, so the processor rarely has to wait for an instruction fetch before proceeding. Since the buffer is short, there is little time penalty when a jump instruction causes the buffer contents to be discarded.

## 3.3 Processes and concurrency

A process starts, performs a number of actions, and then either stops without completing or terminates complete. Typically, a process is a sequence of instructions. A transputer can run several processes in parallel (concurrently). Processes may be assigned either high or low priority, and there may be any number of each (page 269).

The processor has a microcoded scheduler which enables any number of concurrent processes to be executed together, sharing the processor time. This removes the need for a software kernel.

At any time, a concurrent process may be

Active - Being executed.

On a list waiting to be executed.

Inactive - Ready to input.

Ready to output.

- Waiting until a specified time.

The scheduler operates in such a way that inactive processes do not consume any processor time. It allocates a portion of the processor's time to each process in turn. Active processes waiting to be executed are held in two linked lists of process workspaces, one of high priority processes and one of low priority processes (page 269). Each list is implemented using two registers, one of which points to the first process in the list, the other to the last. In the Linked Process List figure 3.3, process S is executing and P, Q and R are active, awaiting execution. Only the low priority process queue registers are shown; the high priority process ones perform in a similar manner.

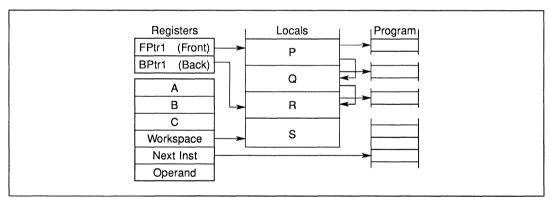


Figure 3.3 Linked process list

Table 3.3 Priority queue control registers

Function	High Priority	Low Priority
Pointer to front of active process list	Fptr0	Fptr1
Pointer to back of active process list	Bptr0	Bptr1

Each process runs until it has completed its action, but is descheduled whilst waiting for communication from another process or transputer, or for a time delay to complete. In order for several processes to operate in parallel, a low priority process is only permitted to run for a maximum of two time slices before it is forcibly descheduled at the next descheduling point (page 273). The time slice period is 5120 cycles of the external 5 MHz clock, giving ticks approximately 1 ms apart.

3 Processor 269

A process can only be descheduled on certain instructions, known as descheduling points (page 273). As a result, an expression evaluation can be guaranteed to execute without the process being timesliced part way through.

Whenever a process is unable to proceed, its instruction pointer is saved in the process workspace and the next process taken from the list. Process scheduling pointers are updated by instructions which cause scheduling operations, and should not be altered directly. Actual process switch times are less than 1  $\mu$ s, as little state needs to be saved and it is not necessary to save the evaluation stack on rescheduling.

The processor provides a number of special operations to support the process model, including *start process* and *end process*. When a main process executes a parallel construct, *start process* instructions are used to create the necessary additional concurrent processes. A *start process* instruction creates a new process by adding a new workspace to the end of the scheduling list, enabling the new concurrent process to be executed together with the ones already being executed. When a process is made active it is always added to the end of the list, and thus cannot pre-empt processes already on the same list.

The correct termination of a parallel construct is assured by use of the *end process* instruction. This uses a workspace location as a counter of the parallel construct components which have still to terminate. The counter is initialised to the number of components before the processes are *started*. Each component ends with an *end process* instruction which decrements and tests the counter. For all but the last component, the counter is non zero and the component is descheduled. For the last component, the counter is zero and the main process continues.

#### 3.4 Priority

The IMS T425 supports two levels of priority. Priority 1 (low priority) processes are executed whenever there are no active priority 0 (high priority) processes.

High priority processes are expected to execute for a short time. If one or more high priority processes are able to proceed, then one is selected and runs until it has to wait for a communication, a timer input, or until it completes processing.

If no process at high priority is able to proceed, but one or more processes at low priority are able to proceed, then one is selected.

Low priority processes are periodically timesliced to provide an even distribution of processor time between computationally intensive tasks.

If there are **n** low priority processes, then the maximum latency from the time at which a low priority process becomes active to the time when it starts processing is 2**n**-2 timeslice periods. It is then able to execute for between one and two timeslice periods, less any time taken by high priority processes. This assumes that no process monopolises the transputer's time; i.e. it has a distribution of descheduling points (page 273).

Each timeslice period lasts for 5120 cycles of the external 5 MHz input clock (approximately 1 ms at the standard frequency of 5 MHz).

If a high priority process is waiting for an external channel to become ready, and if no other high priority process is active, then the interrupt latency (from when the channel becomes ready to when the process starts executing) is typically 19 processor cycles, a maximum of 58 cycles (assuming use of on-chip RAM).

#### 3.5 Communications

Communication between processes is achieved by means of channels. Process communication is point-topoint, synchronised and unbuffered. As a result, a channel needs no process queue, no message queue and no message buffer.

A channel between two processes executing on the same transputer is implemented by a single word in memory; a channel between processes executing on different transputers is implemented by point-to-point

links. The processor provides a number of operations to support message passing, the most important being input message and output message.

The *input message* and *output message* instructions use the address of the channel to determine whether the channel is internal or external. Thus the same instruction sequence can be used for both, allowing a process to be written and compiled without knowledge of where its channels are connected.

The process which first becomes ready must wait until the second one is also ready. A process performs an input or output by loading the evaluation stack with a pointer to a message, the address of a chanhel, and a count of the number of bytes to be transferred, and then executing an *input message* or *output message* instruction. Data is transferred if the other process is ready. If the channel is not ready or is an external one the process will deschedule.

#### 3.6 Block move

The block move on the transputer moves any number of bytes from any byte boundary in memory, to any other byte boundary, using the smallest possible number of word read, and word or part-word writes.

A block move instruction can be interrupted by a high priority process. On interrupt, block move is completed to a word boundary, independent of start position. When restarting after interrupt, the last word written is written again. This appears as an unnecessary read and write in the simplest case of word aligned block moves, and may cause problems with FIFOs. This problem can be overcome by incrementing the saved destination (BregIntSaveLoc) and source pointer (CregIntSaveLoc) values by BytesPerWord during the high priority process.

#### 3.7 Timers

The transputer has two 32 bit timer clocks which 'tick' periodically. The timers provide accurate process timing, allowing processes to deschedule themselves until a specific time.

One timer is accessible only to high priority processes and is incremented every microsecond, cycling completely in approximately 4295 seconds. The other is accessible only to low priority processes and is incremented every 64 microseconds, giving exactly 15625 ticks in one second. It has a full period of approximately 76 hours.

Table 3.4 Timer registers

	Current value of high priority (level 0) process clock Current value of low priority (level 1) process clock
TNextReg0	Indicates time of earliest event on high priority (level 0) timer queue Indicates time of earliest event on low priority (level 1) timer queue

The current value of the processor clock can be read by executing a *load timer* instruction. A process can arrange to perform a *timer input*, in which case it will become ready to execute after a specified time has been reached. The *timer input* instruction requires a time to be specified. If this time is in the 'past' then the instruction has no effect. If the time is in the 'future' then the process is descheduled. When the specified time is reached the process is scheduled again.

3 Processor 271

Figure 3.4 shows two processes waiting on the timer queue, one waiting for time 21, the other for time 31.

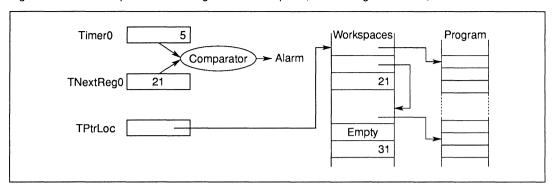


Figure 3.4 Timer registers

# 4 Instruction set summary

The Function Codes table 4.7. gives the basic function code set (page 266). Where the operand is less than 16, a single byte encodes the complete instruction. If the operand is greater than 15, one prefix instruction (*pfix*) is required for each additional four bits of the operand. If the operand is negative the first prefix instruction will be *nfix*.

Mnemonic			Function code	Memory code
ldc	#3		#4	#43
<i>ldc</i> is coded as	#35			
pfix Idc	#3 #5		#2 #4	#23 #45
<i>ldc</i> is coded as	#987			
pfix pfix Idc	#9 #8 #7		#2 #2 #4	#29 #28 #47
<i>ldc</i> is coded as	-31	(Idc	#FFFFFE1)	
nfix Idc	#1 #1		#6 #4	#61 #41

Table 4.1 prefix coding

Tables 4.8 to 4.21 give details of the operation codes. Where an operation code is less than 16 (e.g. *add*: operation code **05**), the operation can be stored as a single byte comprising the *operate* function code **F** and the operand (**5** in the example). Where an operation code is greater than 15 (e.g. *ladd*: operation code **16**), the *prefix* function code **2** is used to extend the instruction.

Mnem	nonic	Function code	Memory code
add is coded as	(op. code #5)		#F5
opr	add	#F	#F5
ladd is coded as	(op. code #16)		#21F6
pfix opr	#1 #6	#2 #F	#21 #F6

Table 4.2 operate coding

The load device identity (*Iddevid*) instruction (table 4.20) pushes the device type identity into the A register. Each product is allocated a unique group of numbers for use with the *Iddevid* instruction. The product identity numbers for the IMS T425 are 0 to 9 inclusive.

The Processor Cycles column refers to the number of periods **TPCLPCL** taken by an instruction executing in internal memory. The number of cycles is given for the basic operation only; where the memory code for an instruction is two bytes, the time for the *prefix* function (one cycle) should be added. For a 20 MHz transputer one cycle is 50 ns. Some instruction times vary. Where a letter is included in the cycles column it is interpreted from table 4.3.

Table 4.3 Instruction set interpretation

Ident	Interpretation
b	Bit number of the highest bit set in register A. Bit 0 is the least significant bit.
m	Bit number of the highest bit set in the absolute value of register A. Bit 0 is the least significant bit.
n	Number of places shifted.
w	Number of words in the message. Part words are counted as full words. If the message is not word aligned the number of words is increased to include the part words at either end of the message.
р	Number of words per row.
r	Number of rows.

The **DE** column of the tables indicates the descheduling/error features of an instruction as described in table 4.4.

Table 4.4 Instruction features

ldent	Feature	See page:
D	The instruction is a descheduling point	273
E	The instruction will affect the Error flag	274, 285

# 4.1 Descheduling points

The instructions in table 4.5 are the only ones at which a process may be descheduled (page 268). They are also the ones at which the processor will halt if the **Analyse** pin is asserted (page 284).

Table 4.5 Descheduling point instructions

input message	output message	output byte	output word	
timer alt wait	timer input	stop on error	alt wait	
jump	loop end	end process	stop process	

## 4.2 Error instructions

The instructions in table 4.6 are the only ones which can affect the Error flag (page 285) directly.

Table 4.6 Error setting instructions

add multiply	add constant fractional multiply	subtract divide	remainder
long add	long subtract	long divide	
set error	testerr		cflerr
check word	check subscript from 0	check single	check count from 1

# 4.3 Debugging support

Table 4.21 contains a number of instructions to facilitate the implementation of breakpoints. These instructions overload the operation of *j0*. Normally *j0* is a no-op which might cause descheduling. *Setj0break* enables the breakpointing facilities and causes *j0* to act as a breakpointing instruction. When breakpointing is enabled, *j0* swaps the current **lptr** and **Wptr** with an **lptr** and **Wptr** stored above MemStart. The breakpoint instruction does not cause descheduling, and preserves the state of the registers. It is possible to single step the processor at machine level using these instructions. Refer to *Support for debugging/breakpointing in transputers* (technical note 61) for more detailed information regarding debugger support.

Table 4.7 IMS T425 function codes

Function Code	Memory Code	Mnemonic	Processor Cycles	Name	D E
0	0X	j	3	jump	D
1	1X	ldlp	1	load local pointer	
2	2X	pfix	1	prefix	
3	3X	ldnl	2	load non-local	
4	4X	ldc	1	load constant	
5	5X	ldnlp	1	load non-local pointer	
6	6X	nfix	1	negative prefix	
7	7X	ldl	2	load local	
8	8X	adc	1	add constant	E
9	9X	call	7	call	
Α	AX	cj	2	conditional jump (not taken)	
			4	conditional jump (taken)	
В	вх	ajw	1	adjust workspace	
С	CX	eqc	2	equals constant	
D	DX	sti	1	store local	
E	EX	stni	2	store non-local	
F	FX	opr		operate	

Table 4.8 IMS T425 arithmetic/logical operation codes

Operation Code	Memory Code	Mnemonic	Processor	Name	D E
			Cycles	<del> </del>	-
46	24F6	and	1	and	
4B	24FB	or	1	or	
33	23F3	xor	1	exclusive or	
32	23F2	not	1	bitwise not	
41	24F1	shl	n+2	shift left	ll
40	24F0	shr	n+2	shift right	
05	F5	add	1	add	Е
0C	FC	sub	1	subtract	E
53	25F3	mul	38	multiply	E
72	27F2	fmul	35	fractional multiply (no rounding)	E
			40	fractional multiply (rounding)	E
2C	22FC	div	39	divide	E
1F	21FF	rem	37	remainder	E
09	F9	gt	2	greater than	
04	F4	diff	1	difference	
52	25F2	sum	1	sum	
08	F8	prod	b+4	product for positive register A	
			m+5	product for negative register A	

Operation Code	Memory Code	Mnemonic	Processor Cycles	Name	D E
16	21F6	ladd	2	long add	Е
38	23F8	Isub	2	long subtract	E
37	23F7	Isum	3	long sum	1 1
4F	24FF	ldiff	3	long diff	
31	23F1	lmul	33	long multiply	
1A	21FA	ldiv	35	long divide	E
36	23F6	Ishi	<b>n</b> +3	long shift left (n<32)	
			n-28	long shift left(n≥32)	
35	23F5	Ishr	<b>n</b> +3	long shift right (n<32)	
			n-28	long shift right (n≥32)	
19	21F9	norm	n+5	normalise (n<32)	
			n-26	normalise ( <b>n</b> ≥32)	
			3	normalise ( <b>n</b> =64)	

Table 4.9 IMS T425 long arithmetic operation codes

Table 4.10 IMS T425 general operation codes

Operation Code	Memory Code	Mnemonic	Processor Cycles	Name	DE
00	F0	rev	1	reverse	
3A	23FA	xword	4	extend to word	
56	25F6	cword	5	check word	E
1D	21FD	xdble	2	extend to double	
4C	24FC	csngl	3	check single	E
42	24F2	mint	1	minimum integer	
5A	25FA	dup	1	duplicate top of stack	
79	27F9	рор	1	pop processor stack	

Table 4.11 IMS T425 floating point support operation codes

Operation Code	Memory Code	Mnemonic	Processor Cycles	Name	D E
73	27F3	cflerr	3	check floating point error	E
9C	29FC	fptesterr	1	load value true (FPU not present)	
63	26F3	unpacksn	15	unpack single length fp number	
6D	26FD	roundsn	12/15	round single length fp number	
6C	26FC	postnormsn	5/30	post-normalise correction of single length fp number	
71	27F1	ldinf	1	load single length infinity	

Processor cycles are shown as Typical/Maximum cycles.

Operation Code	Memory Code	Mnemonic	Processor Cycles	Name	D E
5B	25FB	move2dinit	8	initialise data for 2D block move	
5C	25FC	move2dall	(2p+23)*r	2D block copy	İ
5D	25FD	move2dnonzero	(2 <b>p</b> +23)∗r	2D block copy non-zero bytes	1
5E	25FE	move2dzero	(2p+23)*r	2D block copy zero bytes	

Table 4.12 IMS T425 2D block move operation codes

Table 4.13 IMS T425 CRC and bit operation codes

Operation Code	Memory Code	Mnemonic	Processor Cycles	Name	D E
74	27F4	crcword	35	calculate crc on word	
75	27F5	crcbyte	11	calculate crc on byte	
76	27F6	bitcnt	<b>b</b> +2	count bits set in word	
77	27F7	bitrevword	36	reverse bits in word	
78	27F8	bitrevnbits	n+4	reverse bottom n bits in word	

Table 4.14 IMS T425 indexing/array operation codes

Operation Code	Memory Code	Mnemonic	Processor Cycles	Name	D E
02	F2	bsub	1	byte subscript	
0A	FA	wsub	2	word subscript	
81	28F1	wsubdb	3	form double word subscript	
34	23F4	bcnt	2	byte count	
3F	23FF	wcnt	5	word count	
01	F1	lb	5	load byte	
3B	23FB	sb	4	store byte	
4A	24FA	move	2 <b>w</b> +8	move message	

Table 4.15 IMS T425 timer handling operation codes

Operation Code	Memory Code	Mnemonic	Processor Cycles	Name	ΒD
22	22F2	ldtimer	2	load timer	
2B	22FB	tin	30	timer input (time future)	D
			4	timer input (time past)	D
4E	24FE	talt	4	timer alt start	
51	25F1	taltwt	15	timer alt wait (time past)	D
			48	timer alt wait (time future)	D
47	24F7	enbt	8	enable timer	
2E	22FE	dist	23	disable timer	

Operation Code	Memory Code	Mnemonic	Processor Cycles	Name	D E
07	F7	in	2 <b>w</b> +19	input message	D
0B	FB	out	2 <b>w</b> +19	output message	D
0F	FF	outword	23	output word	D
0E	FE	outbyte	23	output byte	D
43	24F3	alt	2	alt start	
44	24F4	altwt	5	alt wait (channel ready)	D
			17	alt wait (channel not ready)	D
45	24F5	altend	4	alt end	
49	24F9	enbs	3	enable skip	
30	23F0	diss	4	disable skip	
12	21F2	resetch	3	reset channel	
48	24F8	enbc	7	enable channel (ready)	
			5	enable channel (not ready)	
2F	22FF	disc	8	disable channel	

Table 4.16 IMS T425 input/output operation codes

Table 4.17 IMS T425 control operation codes

Operation Code	Memory Code	Mnemonic	Processor Cycles	Name	D E
20	22F0	ret	5	return	
1B	21FB	Idpi	2	load pointer to instruction	
3C	23FC	gajw	2	general adjust workspace	
06	F6	gcall	4	general call	
21	22F1	lend	10	loop end (loop)	D
			5	loop end (exit)	D

Table 4.18 IMS T425 scheduling operation codes

Code	Mnemonic	Processor Cycles	Name	D
FD	startp	12	start process	D
F3	endp	13	end process	D
23F9	runp	10	run process	
21F5	stopp	11	stop process	
21FE	Idpri	1	load current priority	
	FD F3 23F9 21F5	FD startp F3 endp 23F9 runp 21F5 stopp	Code         Mnemonic         Cycles           FD         startp         12           F3         endp         13           23F9         runp         10           21F5         stopp         11	Code         Mnemonic         Cycles         Name           FD         startp         12         start process           F3         endp         13         end process           23F9         runp         10         run process           21F5         stopp         11         stop process

Operation Code	Memory Code	Mnemonic	Processor Cycles	Name	D E
13	21F3	csub0	2	check subscript from 0	E
4D	24FD	ccnt1	3	check count from 1	E
29	22F9	testerr	2	test error false and clear (no error)	Ì
			3	test error false and clear (error)	
10	21F0	seterr	1	set error	E
55	25F5	stoperr	2	stop on error (no error)	D
57	25F7	cirhaiterr	1	clear halt-on-error	
58	25F8	sethalterr	1	set halt-on-error	
59	25F9	testhalterr	2	test halt-on-error	

Table 4.19 IMS T425 error handling operation codes

Table 4.20 IMS T425 processor initialisation operation codes

Operation Code	Memory Code	Mnemonic	Processor Cycles	Name	D E
2A	22FA	testpranal	2	test processor analysing	
3E	23FE	saveh	4	save high priority queue registers	
3D	23FD	savel	4	save low priority queue registers	
18	21F8	sthf	1	store high priority front pointer	
50	25F0	sthb	1	store high priority back pointer	
1C	21FC	stlf	1	store low priority front pointer	
17	21F7	stlb	1	store low priority back pointer	
54	25F4	sttimer	1	store timer	
17C	2127FC	Iddevid	1	load device identity	
7E	27FE	Idmemstartval	1	load value of memstart address	

Table 4.21 IMS T425 debugger support codes

Operation Code	Memory Code	Mnemonic	Processor Cycles	Name	D E
0	00	jump 0	3	jump 0 (break not enabled)	D
			11	jump 0 (break enabled, high priority)	
			13	jump 0 (break enabled, low priority)	
B1	2BF1	break	9	break (high priority)	
			11	break (low priority)	
B2	2BF2	clrj0break	1	clear jump 0 break enable flag	
B3	2BF3	setj0break	1	set jump 0 break enable flag	
B4	2BF4	testj0break	2	test jump 0 break enable flag set	
7A	27FA	timerdisableh	1	disable high priority timer interrupt	
7B	27FB	timerdisablel	1	disable low priority timer interrupt	
7C	27FC	timerenableh	6	enable high priority timer interrupt	
7D	27FD	timerenablel	6	enable low priority timer interrupt	

# 5 System services

System services include all the necessary logic to initialise and sustain operation of the device. They also include error handling and analysis facilities.

## 5.1 Power

Power is supplied to the device via the VCC and GND pins. Several of each are provided to minimise inductance within the package. All supply pins must be connected. The supply must be decoupled close to the chip by at least one 100 nF low inductance (e.g. ceramic) capacitor between VCC and GND. Four layer boards are recommended; if two layer boards are used, extra care should be taken in decoupling.

Input voltages must not exceed specification with respect to VCC and GND, even during power-up and power-down ramping, otherwise *latchup* can occur. CMOS devices can be permanently damaged by excessive periods of latchup.

# 5.2 CapPlus, CapMinus

The internally derived power supply for internal clocks requires an external low leakage, low inductance  $1\mu$ F capacitor to be connected between **CapPlus** and **CapMinus**. A ceramic capacitor is preferred, with an impedance less than 3 Ohms between 100 KHz and 10 MHz. If a polarised capacitor is used the negative terminal should be connected to **CapMinus**. Total PCB track length should be less than 50 mm. The connections must not touch power supplies or other noise sources.

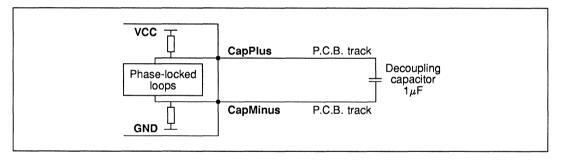


Figure 5.1 Recommended PLL decoupling

## 5.3 Clockin

Transputer family components use a standard clock frequency, supplied by the user on the **Clockin** input. The nominal frequency of this clock for all transputer family components is 5 MHz, regardless of device type, transputer word length or processor cycle time. High frequency internal clocks are derived from **Clockin**, simplifying system design and avoiding problems of distributing high speed clocks externally.

A number of transputer devices may be connected to a common clock, or may have individual clocks providing each one meets the specified stability criteria. In a multi-clock system the relative phasing of **ClockIn** clocks is not important, due to the asynchronous nature of the links. Mark/space ratio is unimportant provided the specified limits of **ClockIn** pulse widths are met.

Oscillator stability is important. **ClockIn** must be derived from a crystal oscillator; RC oscillators are not sufficiently stable. **ClockIn** must not be distributed through a long chain of buffers. Clock edges must be monotonic and remain within the specified voltage and time limits.

System services 281

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTE
TDCLDCH	ClockIn pulse width low	40			ns	1
TDCHDCL	ClockIn pulse width high	40	ļ		ns	1
TDCLDCL	ClockIn period		200		ns	1,2,4
TDCerror	ClockIn timing error			±0.5	ns	1,3
TDC1DC2	Difference in ClockIn for 2 linked devices	(		400	ppm	1,4
TDCr	ClockIn rise time	1		10	ns	1,5
TDCf	ClockIn fall time			8	ns	1,5

Table 5.1 Input clock

## Notes

- 1 These paramters are not tested.
- 2 Measured between corresponding points on consecutive falling edges.
- 3 Variation of individual falling edges from their nominal times.
- 4 This value allows the use of 200ppm crystal oscillators for two devices connected together by a link.
- 5 Clock transitions must be monotonic within the range VIH to VIL (table 10.3).

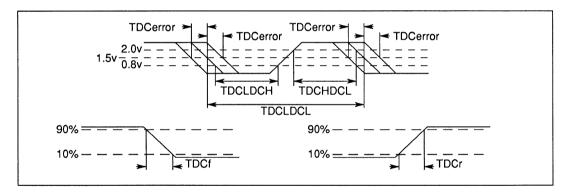


Figure 5.2 ClockIn timing

# 5.4 ProcSpeedSelect0-2

Processor speed of the IMS T425 is variable in discrete steps. The desired speed can be selected, up to the maximum rated for a particular component, by the three speed select lines **ProcSpeedSelect0-2**. The pins are tied high or low, according to the table below, for the various speeds. The pins are arranged so that the IMS T425 can be plugged directly into a board designed for a IMS T800.

Only six of the possible speed select combinations are currently used; the other two are not valid speed selectors. The frequency of **ClockIn** for the speeds given in the table is 5 MHz.

Proc Speed Select2	Proc Speed Select1	Proc Speed Select0	Processor Clock Speed MHz	Processor Cycle Time ns	Notes
0	0	0	20.0	50.0	
0	0	1	22.5	44.4	
0	1	0	25.0	40.0	
0	1	1	30.0	33.3	
1	0	0	35.0	28.6	
1	0	1			Invalid
1 1	1	0	17.5	57.1	
1	1	1			Invalid

Table 5.2 Processor speed selection

Note: Inclusion of a speed selection in this table does not imply immediate availability.

# 5.5 Reset

Reset can go high with VCC, but must at no time exceed the maximum specified voltage for VIH. After VCC is valid ClockIn should be running for a minimum period TDCVRL before the end of Reset. The falling edge of Reset initialises the transputer, triggers the memory configuration sequence and starts the bootstrap routine. Link outputs are forced low during reset; link inputs and EventReq should be held low. Memory request (DMA) must not occur whilst Reset is high but can occur before bootstrap (page 308).

After the end of Reset there will be a delay of 144 periods of ClockIn (figure 5.3). Following this, the MemWrD0, MemRfD1 and MemAD2-31 pins will be scanned to check for the existence of a pre-programmed memory interface configuration (page 297). This lasts for a further 144 periods of ClockIn. Regardless of whether a configuration was found, 36 configuration read cycles will then be performed on external memory using the default memory configuration (page 299), in an attempt to access the external configuration ROM. A delay will then occur, its period depending on the actual configuration. Finally eight complete and consecutive refresh cycles will initialise any dynamic RAM, using the new memory configuration. If the memory configuration does not enable refresh of dynamic RAM the refresh cycles will be replaced by an equivalent delay with no external memory activity.

If **BootFromRom** is high bootstrapping will then take place immediately, using data from external memory; otherwise the transputer will await an input from any link. The processor will be in the low priority state.

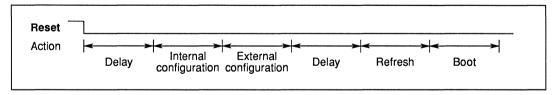


Figure 5.3 IMS T425 post-reset sequence

## 5.6 Bootstrap

The transputer can be bootstrapped either from a link or from external ROM. To facilitate debugging, **Boot-FromRom** may be dynamically changed but must obey the specified timing restrictions. It is sampled once only by the transputer, before the first instruction is executed after **Reset** is taken low.

If **BootFromRom** is connected high (e.g. to **VCC**) the transputer starts to execute code from the top two bytes in external memory, at address #7FFFFFFE. This location should contain a backward jump to a program in ROM. Following this access, **BootFromRom** may be taken low if required. The processor is in the low priority

state, and the W register points to MemStart (page 286).

Table	5.3	Reset	and	Analyse
-------	-----	-------	-----	---------

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTE
TPVRH	Power valid before Reset	10			ms	
TRHRL	Reset pulse width high	8			Clockin	1
TDCVRL	ClockIn running before Reset end	10			ms	2
TAHRH	Analyse setup before Reset	3			ms	
TRLAL	Analyse hold after Reset end	1			ClockIn	1
TBRVRL	BootFromRom setup	0			ms	
TRLBRX	BootFromRom hold after Reset	0			ms	3
TALBRX	BootFromRom hold after Analyse					3

- 1 Full periods of Clockin TDCLDCL required.
- 2 At power-on reset.
- 3 Must be stable until after end of bootstrap period. See Bootstrap section.

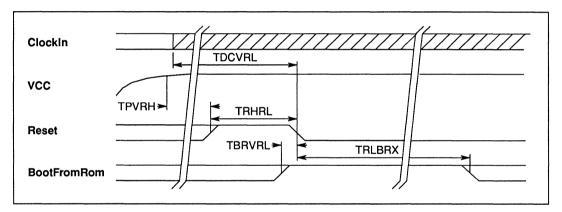


Figure 5.4 Transputer reset timing with Analyse low

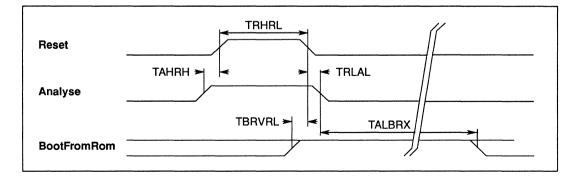


Figure 5.5 Transputer reset and analyse timing

If **BootFromRom** is connected low (e.g. to **GND**) the transputer will wait for the first bootstrap message to arrive on any one of its links. The transputer is ready to receive the first byte on a link within two processor cycles **TPCLPCL** after **Reset** goes low.

If the first byte received (the control byte) is greater than 1 it is taken as the quantity of bytes to be input. The following bytes, to that quantity, are then placed in internal memory starting at location *MemStart*. Following reception of the last byte the transputer will start executing code at *MemStart* as a low priority process. **BootFromRom** may be taken high after reception of the last byte, if required. The memory space immediately above the loaded code is used as work space. Messages arriving on other links after the control byte has been received and on the bootstrapping link after the last bootstrap byte will be retained until a process inputs from them.

# 5.7 Peek and poke

Any location in internal or external memory can be interrogated and altered when the transputer is waiting for a bootstrap from link. If the control byte is 0 then eight more bytes are expected on the same link. The first four byte word is taken as an internal or external memory address at which to poke (write) the second four byte word. If the control byte is 1 the next four bytes are used as the address from which to peek (read) a word of data; the word is sent down the output channel of the same link.

Following such a peek or poke, the transputer returns to its previously held state. Any number of accesses may be made in this way until the control byte is greater than 1, when the transputer will commence reading its bootstrap program. Any link can be used, but addresses and data must be transmitted via the same link as the control byte.

## 5.8 Analyse

If **Analyse** is taken high when the transputer is running, the transputer will halt at the next descheduling point (page 273). From **Analyse** being asserted, the processor will halt within three time slice periods plus the time taken for any high priority process to complete. As much of the transputer status is maintained as is necessary to permit analysis of the halted machine. Processor flags **Error**, **HaltOnError** and **EnableJOBreak** are normally cleared at reset on the IMS T425; however, if **Analyse** is asserted the flags are not altered. Memory refresh continues.

Input links will continue with outstanding transfers. Output links will not make another access to memory for data but will transmit only those bytes already in the link buffer. Providing there is no delay in link acknowledgement, the links should be inactive within a few microseconds of the transputer halting.

Reset should not be asserted before the transputer has halted and link transfers have ceased. When Reset is taken low whilst Analyse is high, neither the memory configuration sequence nor the block of eight refresh cycles will occur; the previous memory configuration will be used for any external memory accesses. If BootFromRom is high the transputer will bootstrap as soon as Analyse is taken low, otherwise it will await a control byte on any link. If Analyse is taken low without Reset going high the transputer state and operation are undefined. After the end of a valid Analyse sequence the registers have the values given in table 5.4.

Table 5.4 Register values after Analyse

- I MemStart if bootstrapping from a link, or the external memory bootstrap address if bootstrapping from ROM.
- W MemStart if bootstrapping from ROM, or the address of the first free word after the bootstrap program if bootstrapping from link.
- A The value of I when the processor halted.
- B The value of W when the processor halted, together with the priority of the process when the transputer was halted (i.e. the W descriptor).
- C The ID of the bootstrapping link if bootstrapping from link.

5 System services

## 5.9 Error, Errorin

The **Error** pin carries the OR'ed output of the internal *Error* flag and the **ErrorIn** input. If **Error** is high it indicates either that **ErrorIn** is high or that an error was detected in one of the processes. An internal error can be caused, for example, by arithmetic overflow, divide by zero, array bounds violation or software setting the flag directly (page 274). Once set, the *Error* flag is only cleared by executing the instruction *testerr*. The error is not cleared by processor reset, in order that analysis can identify any errant transputer (page 284).

A process can be programmed to stop if the *Error* flag is set; it cannot then transmit erroneous data to other processes, but processes which do not require that data can still be scheduled. Eventually all processes which rely, directly or indirectly, on data from the process in error will stop through lack of data. **ErrorIn** does not directly affect the status of a processor in any way.

By setting the *HaltOnError* flag the transputer itself can be programmed to halt if *Error* becomes set. If *Error* becomes set after *HaltOnError* has been set, all processes on that transputer will cease but will not necessarily cause other transputers in a network to halt. Setting *HaltOnError* after *Error* will not cause the transputer to halt; this allows the processor reset and analyse facilities to function with the flags in indeterminate states.

An alternative method of error handling is to have the errant process or transputer cause all transputers to halt. This can be done by 'daisy-chaining' the **ErrorIn** and **Error** pins of a number of processors and applying the final **Error** output signal to the **EventReq** pin of a suitably programmed master transputer. Since the process state is preserved when stopped by an error, the master transputer can then use the analyse function to debug the fault. When using such a circuit, note that the *Error* flag is in an indeterminate state on power up; the circuit and software should be designed with this in mind.

Error checks can be removed completely to optimise the performance of a proven program; any unexpected error then occurring will have an arbitrary undefined effect.

If a high priority process pre-empts a low priority one, status of the *Error* and *HaltOnError* flags is saved for the duration of the high priority process and restored at the conclusion of it. Status of both flags is transmitted to the high priority process. Either flag can be altered in the process without upsetting the error status of any complex operation being carried out by the pre-empted low priority process.

In the event of a transputer halting because of *HaltOnError*, the links will finish outstanding transfers before shutting down. If **Analyse** is asserted then all inputs continue but outputs will not make another access to memory for data. Memory refresh will continue to take place.

After halting due to the *Error* flag changing from 0 to 1 whilst *HaltOnError* is set, register *I* points two bytes past the instruction which set *Error*. After halting due to the **Analyse** pin being taken high, register *I* points one byte past the instruction being executed. In both cases *I* will be copied to register *A*.

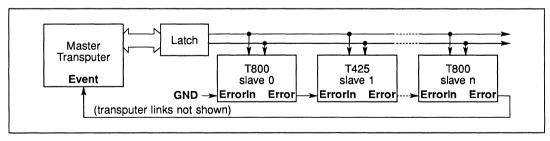


Figure 5.6 Error handling in a multi-transputer system

# 6 Memory

The IMS T425 has 4 Kbytes of fast internal static memory for high rates of data throughput. Each internal memory access takes one processor cycle **ProcClockOut** (page 288). The transputer can also access 4 Gbytes of external memory space. Internal and external memory are part of the same linear address space. Internal RAM can be disabled by holding **DisableIntRAM** high. All internal addresses are then mapped to external RAM. This pin should not be altered after **Reset** has been taken low.

IMS T425 memory is byte addressed, with words aligned on four-byte boundaries. The least significant byte of a word is the lowest addressed byte.

The bits in a byte are numbered 0 to 7, with bit 0 the least significant. The bytes are numbered from 0, with byte 0 the least significant. In general, wherever a value is treated as a number of component values, the components are numbered in order of increasing numerical significance, with the least significant component numbered 0. Where values are stored in memory, the least significant component value is stored at the lowest (most negative) address.

Internal memory starts at the most negative address #80000000 and extends to #80000FFF. User memory begins at #80000070; this location is given the name *MemStart*. An instruction *Idmemstartval* is provided to obtain the value of **MemStart**.

The context of a process in the transputer model involves a workspace descriptor (WPtr) and an instruction pointer (IPtr). WPtr is a word address pointer to a workspace in memory. IPtr points to the next instruction to be executed for the process which is the currently executing process. The context switch performed by the breakpoint instruction swaps the WPtr and IPtr of the currently executing process with the WPtr and IPtr held above MemStart. Two contexts are held above MemStart, one for high priority and one for low priority; this allows processes at both levels to have breakpoints. Note that on bootstrapping from a link, these contexts are overwritten by the loaded code. If this is not acceptable, the values should be peeked from memory before bootstrapping from a link.

The reserved area of internal memory below MemStart is used to implement link and event channels.

Two words of memory are reserved for timer use, *TPtrLoc0* for high priority processes and *TPtrLoc1* for low priority processes. They either indicate the relevant priority timer is not in use or point to the first process on the timer queue at that priority level.

Values of certain processor registers for the current low priority process are saved in the reserved *IntSaveLoc* locations when a high priority process pre-empts a low priority one. Other locations are reserved for extended features such as block moves.

External memory space starts at #80001000 and extends up through #00000000 to #7FFFFFFF. Memory configuration data and ROM bootstrapping code must be in the most positive address space, starting at #7FFFFF6C and #7FFFFFFE respectively. Address space immediately below this is conventionally used for ROM based code.

Figure 6.1 IMS T425 memory map

#### Notes

1 These locations are used as auxiliary processor registers and should not be manipulated by the user. Like processor registers, their contents may be useful for implementing debugging tools (Analyse, page 284). For details see *Transputer Instruction Set - A Compiler Writers' Guide*.

# 7 External memory interface

The External Memory Interface (EMI) allows access to a 32 bit address space, supporting dynamic and static RAM as well as ROM and EPROM. EMI timing can be configured at **Reset** to cater for most memory types and speeds, and a program is supplied with the Transputer Development System to aid in this configuration.

There are 17 internal configurations which can be selected by a single pin connection (page 297). If none are suitable the user can configure the interface to specific requirements, as shown in page 299.

#### 7.1 ProcClockOut

This clock is derived from the internal processor clock, which is in turn derived from ClockIn. Its period is equal to one internal microcode cycle time, and can be derived from the formula

## TPCLPCL = TDCLDCL / PLLx

where TPCLPCL is the ProcClockOut Period, TDCLDCL is the ClockIn Period and PLLx is the phase lock loop factor for the relevant speed part, obtained from the ordering details (Ordering section).

The time value **Tm** is used to define the duration of **Tstates** and, hence, the length of external memory cycles; its value is exactly half the period of one **ProcClockOut** cycle (0.5\***TPCLPCL**), regardless of mark/space ratio of **ProcClockOut**.

Edges of the various external memory strobes coincide with rising or falling edges of **ProcClockOut**. It should be noted, however, that there is a skew associated with each coincidence. The value of skew depends on whether coincidence occurs when the **ProcClockOut** edge and strobe edge are both rising, when both are falling or if either is rising when the other is falling. Timing values given in the strobe tables show the best and worst cases. If a more accurate timing relationship is required, the exact **Tstate** timing and strobe edge to **ProcClockOut** relationships should be calculated and the correct skew factors applied from the edge skew timing table 7.4.

The timing parameters in the following tables are based on 17 MHz and 20 MHz parts. Data for higher speeds is based on tests on a limited number of samples and may change when full characterisation is completed.

### 7.2 Tstates

The external memory cycle is divided into six **Tstates** with the following functions:

- T1 Address setup time before address valid strobe.
- T2 Address hold time after address valid strobe.
- T3 Read cycle tristate or write cycle data setup.
- T4 Extendable data setup time.
- T5 Read or write data.
- T6 Data hold.

Under normal conditions each **Tstate** may be from one to four periods **Tm** long, the duration being set during memory configuration. The default condition on **Reset** is that all **Tstates** are the maximum four periods **Tm** long to allow external initialisation cycles to read slow ROM.

Period **T4** can be extended indefinitely by adding externally generated wait states.

An external memory cycle is always an even number of periods **Tm** in length and the start of **T1** always coincides with a rising edge of **ProcClockOut**. If the total configured quantity of periods **Tm** is an odd number, one extra period **Tm** will be added at the end of **T6** to force the start of the next **T1** to coincide with a rising edge of **ProcClockOut**. This period is designated **E** in configuration diagrams (figure 7.11).

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTE
TPCLPCL	ProcClockOut period	<b>a</b> -2	а	a+2	ns	1,5
TPCHPCL	ProcClockOut pulse width high	<b>b</b> -11.5	b	<b>b</b> +3.5	ns	2,5
TPCLPCH	ProcClockOut pulse width low		С		ns	3,5
Tm	ProcClockOut half cycle	b-1	b	b+1	ns	2,5
TPCstab	ProcClockOut stability			8	%	4,5

Table 7.1 ProcClockOut

#### Notes

- 1 a is TDCLDCL/PLLx.
- 2 b is 0.5\*TPCLPCL (half the processor clock period).
- 3 c is TPCLPCL-TPCHPCL.
- 4 Stability is the variation of cycle periods between two consecutive cycles, measured at corresponding points on the cycles.
- 5 This parameter is sampled and not 100% tested.

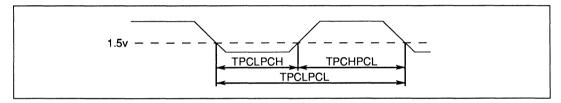


Figure 7.1 IMS T425 ProcClockOut timing

## 7.3 Internal access

During an internal memory access cycle the external memory interface bus **MemAD2-31** reflects the word address used to access internal RAM, **MemnotWrD0** reflects the read/write operation and **MemnotRfD1** is high; all control strobes are inactive. This is true unless and until a memory refresh cycle or DMA (memory request) activity takes place, when the bus will carry the appropriate external address or data.

The bus activity is not adequate to trace the internal operation of the transputer in full, but may be used for hardware debugging in conjuction with peek and poke (page 284).

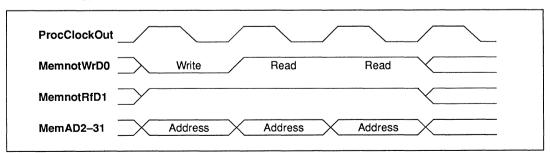


Figure 7.2 IMS T425 bus activity for internal memory cycle

#### 7.4 MemAD2-31

External memory addresses and data are multiplexed on one bus. Only the top 30 bits of address are output on the external memory interface, using pins **MemAD2-31**. They are normally output only during **Tstates T1** and **T2**, and should be latched during this time. Byte addressing is carried out internally by the transputer for read cycles. For write cycles the relevant bytes in memory are addressed by the write strobes **notMemWrB0-3**.

The data bus is 32 bits wide. It uses MemAD2-31 for the top 30 bits and MemnotRfD1 and MemnotWrD0 for the lower two bits. Read cycle data may be set up on the bus at any time after the start of T3, but must be valid when the transputer reads it at the end of T5. Data may be removed any time during T6, but must be off the bus no later than the end of that period.

Write data is placed on the bus at the start of T3 and removed at the end of T6. If T6 is extended to force the next cycle Tmx (page 290) to start on a rising edge of ProcClockOut, data will be valid during this time also.

### 7.5 MemnotWrD0

During T1 and T2 this pin will be low if the cycle is a write cycle, otherwise it will be high. During Tstates T3 to T6 it becomes bit 0 of the data bus. In both cases it follows the general timing of MemAD2-31.

#### 7.6 MemnotRfD1

During T1 and T2, this pin is low if the address on MemAD2-31 is a refresh address, otherwise it is high. During Tstates T3 to T6 it becomes bit 1 of the data bus. In both cases it follows the general timing of MemAD2-31.

#### 7.7 notMemRd

For a read cycle the read strobe **notMemRd** is low during **T4** and **T5**. Data is read by the transputer on the rising edge of this strobe, and may be removed immediately afterward. If the strobe duration is insufficient it may be extended by adding extra periods **Tm** to either or both of the **Tstates T4** and **T5**. Further extension may be obtained by inserting wait states at the end of **T4**.

In the read cycle timing diagrams ProcClockOut is included as a guide only; it is shown with each Tstate configured to one period Tm.

#### 7.8 notMemS0-4

To facilitate control of different types of memory and devices, the EMI is provided with five strobe outputs, four of which can be configured by the user. The strobes are conventionally assigned the functions shown in the read and write cycle diagrams, although there is no compulsion to retain these designations.

**notMemS0** is a fixed format strobe. Its leading edge is always coincident with the start of **T2** and its trailing edge always coincident with the end of **T5**.

The leading edge of **notMemS1** is always coincident with the start of **T2**, but its duration may be configured to be from zero to 31 periods **Tm**. Regardless of the configured duration, the strobe will terminate no later than the end of **T6**. The strobe is sometimes programmed to extend beyond the normal end of **Tmx**. When wait states are inserted into an EMI cycle the end of **Tmx** is delayed, but the potential active duration of the strobe is not altered. Thus the strobe can be configured to terminate relatively early under certain conditions (page 306). If **notMemS1** is configured to be zero it will never go low.

notMemS2, notMemS3 and notMemS4 are identical in operation. They all terminate at the end of T5, but the start of each can be delayed from one to 31 periods Tm beyond the start of T2. If the duration of one of these strobes would take it past the end of T5 it will stay high. This can be used to cause a strobe to become active only when wait states are inserted. If one of these strobes is configured to zero it will never go low. Figure 7.5 shows the effect of Wait on strobes in more detail; each division on the scale is one period Tm.

Τa	h	۵۱	7	2	R	ea	H

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTE
TaZdV	Address tristate to data valid	0			ns	
TdVRdH	Data setup before read	25			ns	
TRdHdX	Data hold after read	0			ns	
TS0LRdL	notMemS0 before start of read	a-4	a	a+4	ns	1
TS0HRdH	End of read from end of notMemS0	-4		4	ns	
TRdLRdH	Read period	<b>b</b> -3		<b>b</b> +5	ns	2

- 1 a is total of T2+T3 where T2, T3 can be from one to four periods Tm each in length.
- 2 b is total of T4+Twait+T5 where T4, T5 can be from one to four periods Tm each in length and Twait may be any number of periods Tm in length.

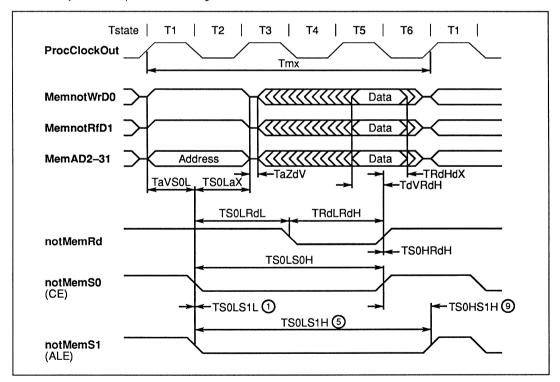


Figure 7.3 IMS T425 external read cycle: static memory

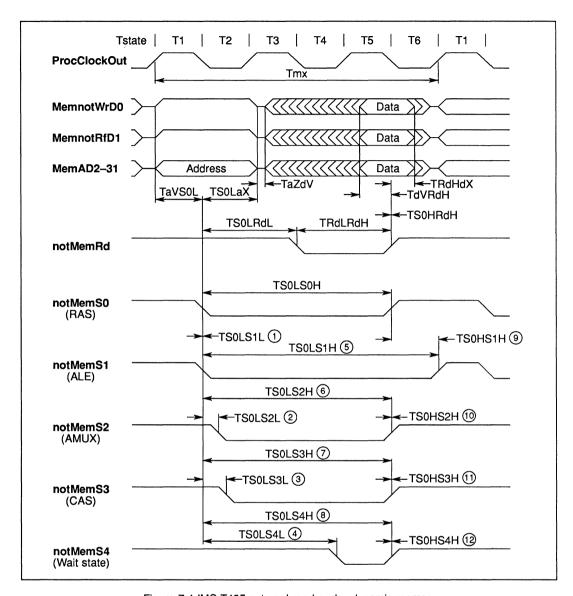


Figure 7.4 IMS T425 external read cycle: dynamic memory

SYMBOL	<b>(</b>	PARAMETER	MIN	NOM	MAX	UNITS	NOTE
TaVS0L		Address setup before notMemS0	<b>a</b> -8			ns	1
TS0LaX		Address hold after notMemS0	<b>b</b> -8	b	<b>b</b> +8	ns	2
TS0LS0H		notMemS0 pulse width low	<b>c</b> -5		<b>c</b> +6	ns	3
TS0LS1L	1	notMemS1 from notMemS0	-4	1	4	ns	1
TS0LS1H	5	notMemS1 end from notMemS0	<b>d</b> -1		<b>d</b> +9	ns	4,6
TS0HS1H	9	notMemS1 end from notMemS0 end	<b>e</b> -8		e+4	ns	5,6
TS0LS2L	2	notMemS2 delayed after notMemS0	f-6	ļ	f+5	ns	7
TS0LS2H	6	notMemS2 end from notMemS0	<b>c</b> -5		c+7	ns	3
TS0HS2H	10	notMemS2 end from notMemS0 end	-4		7	ns	
TS0LS3L	3	notMemS3 delayed after notMemS0	<b>f</b> -6		f+5	ns	7
TS0LS3H	7	notMemS3 end from notMemS0	<b>c</b> -5		<b>c</b> +7	ns	3
TS0HS3H	11	notMemS3 end from notMemS0 end	-4		7	ns	
TS0LS4L	4	notMemS4 delayed after notMemS0	f-6		f+5	ns	7
TS0LS4H	8	notMemS4 end from notMemS0	<b>c</b> -5		<b>c</b> +7	ns	3
TS0HS4H	12	notMemS4 end from notMemS0 end	-4		7	ns	
Tmx		Complete external memory cycle		g			8

Table 7.3 IMS T425 strobe timing

- 1 a is T1 where T1 can be from one to four periods Tm in length.
- 2 b is T2 where T2 can be from one to four periods Tm in length.
- 3 c is total of T2+T3+T4+Twait+T5 where T2, T3, T4, T5 can be from one to four periods Tm each in length and Twait may be any number of periods Tm in length.
- 4 d can be from zero to 31 periods Tm in length.
- 5 e can be from -27 to +4 periods Tm in length.
- 6 If the configuration would cause the strobe to remain active past the end of **T6** it will go high at the end of **T6**. If the strobe is configured to zero periods **Tm** it will remain high throughout the complete cycle **Tmx**.
- 7 f can be from zero to 31 periods Tm in length. If this length would cause the strobe to remain active past the end of T5 it will go high at the end of T5. If the strobe value is zero periods Tm it will remain low throughout the complete cycle Tmx.
- 8 g is one complete external memory cycle comprising the total of T1+T2+T3+T4+Twait+T5+T6 where T1, T2, T3, T4, T5 can be from one to four periods Tm each in length, T6 can be from one to five periods Tm in length and Twait may be zero or any number of periods Tm in length.

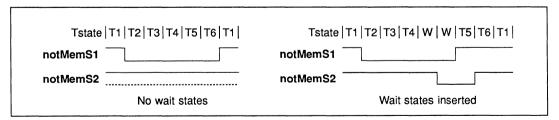


Figure 7.5 IMS T425 effect of wait states on strobes

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTE
TPCHS0H	notMemS0 rising from ProcClockOut rising	-6		4	ns	
TPCLS0H	notMemS0 rising from ProcClockOut falling	-5	į .	10	ns	
TPCHS0L	notMemS0 falling from ProcClockOut rising	-8		3	ns	
TPCLS0L	notMemS0 falling from ProcClockOut falling	-5		7	ns	

Table 7.4 Strobe S0 to ProcClockOut skew

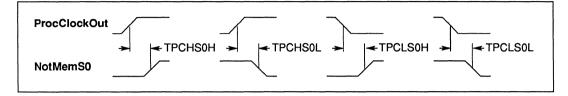


Figure 7.6 IMS T425 skew of notMemS0 to ProcClockOut

# 7.9 notMemWrB0-3

Because the transputer uses word addressing, four write strobes are provided; one to write each byte of the word. If a particular byte is not to be written, then the corresponding data outputs are tristated. **notMemWrB0** addresses the least significant byte.

The transputer has both early and late write cycle modes. For a late write cycle the relevant write strobes notMemWrB0-3 are low during T4 and T5; for an early write they are also low during T3. Data should be latched into memory on the rising edge of the strobes in both cases, although it is valid until the end of T6. If the strobe duration is insufficient, it may be extended at configuration time by adding extra periods Tm to either or both of Tstates T4 and T5 for both early and late modes. For an early cycle they may also be added to T3. Further extension may be obtained by inserting wait states at the end of T4. If the data hold time is insufficient, extra periods Tm may be added to T6 to extend it.

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTE
TdVWrH	Data setup before write	<b>d</b> -7		<b>d</b> +10	ns	1,5
TWrHdX	Data hold after write	<b>a</b> -10		<b>a</b> +5	ns	1,2
TS0LWrL	notMemS0 before start of early write	<b>b</b> -5		<b>b</b> +5	ns	1,3
	notMemS0 before start of late write	<b>c</b> -5		<b>c</b> +5	ns	1,4
TS0HWrH	End of write from end of notMemS0	-5		4	ns	1
TWrLWrH	Early write pulse width	<b>d</b> -4		<b>d</b> +7	ns	1,5
	Late write pulse width	<b>e</b> -4		<b>e</b> +7	ns	1,6

Table 7.5 Write

- 1 Timing is for all write strobes notMemWrB0-3.
- 2 a is T6 where T6 can be from one to five periods Tm in length.
- 3 b is T2 where T2 can be from one to four periods Tm in length.
- 4 c is total of T2+T3 where T2, T3 can be from one to four periods Tm each in length.
- 5 d is total of T3+T4+Twait+T5 where T3, T4, T5 can be from one to four periods Tm each in length and Twait may be zero or any number of periods Tm in length.
- 6 e is total of T4+Twait+T5 where T4, T5 can be from one to four periods Tm each in length and Twait may be zero or any number of periods Tm in length.

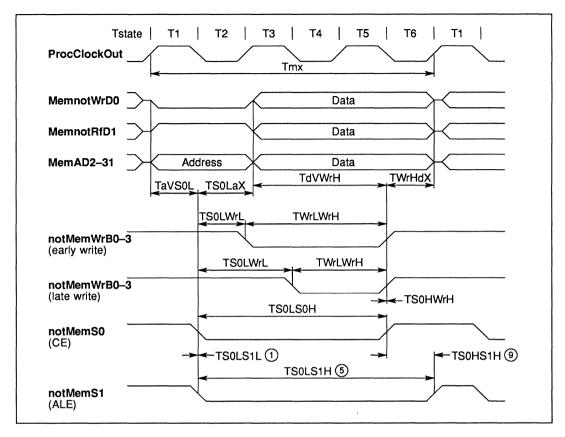


Figure 7.7 IMS T425 external write cycle

In the write cycle timing diagram ProcClockOut is included as a guide only; it is shown with each Tstate configured to one period Tm. The strobe is inactive during internal memory cycles.

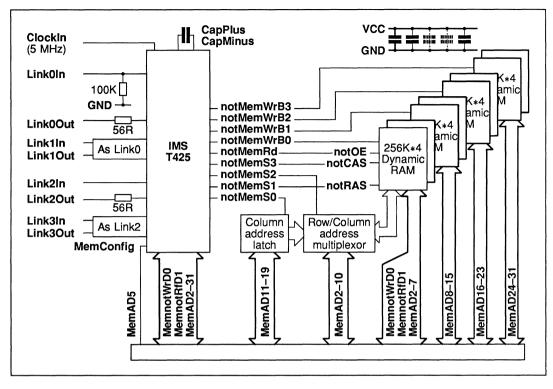


Figure 7.8 IMS T425 dynamic RAM application

# 7.10 MemConfig

**MemConfig** is an input pin used to read configuration data when setting external memory interface (EMI) characteristics. It is read by the processor on two occasions after **Reset** goes low; first to check if one of the preset internal configurations is required, then to determine a possible external configuration.

## 7.10.1 Internal configuration

The internal configuration scan comprises 64 periods TDCLDCL of ClockIn during the internal scan period of 144 ClockIn periods. MemnotWrD0, MemnotRfD1 and MemAD2-32 are all high at the beginning of the scan. Starting with MemnotWrD0, each of these lines goes low successively at intervals of two ClockIn periods and stays low until the end of the scan. If one of these lines is connected to MemConfig the preset internal configuration mode associated with that line will be used as the EMI configuration. The default configuration is that defined in the table for MemAD31; connecting MemConfig to VCC will also produce this default configuration. Note that only 17 of the possible configurations are valid, all others remain at the default configuration.

	Di	uratio p		each ds Tn		ite	(	Stro coeff	obe icien	t	Write cycle	Refresh interval	Cycle time
Pin	T1	T2	тз	T4	T5	Т6	s1	s2	s3	s4	type	Clockin cycles	Proc cycles
MemnotWrD0	1	1	1	1	1	1	30	1	3	5	late	72	3
MemnotRfD1	1	2	1	1	1	2	30	1	2	7	late	72	4
MemAD2	1	2	1	1	2	3	30	1	2	7	late	72	5
MemAD3	2	3	1	1	2	3	30	1	3	8	late	72	6
MemAD4	1	1	1	1	1	1	3	1	2	3	early	72	3
MemAD5	1	1	2	1	2	1	5	1	2	3	early	72	4
MemAD6	2	1	2	1	3	1	6	1	2	3	early	72	5
MemAD7	2	2	2	1	3	2	7	1	3	4	early	72	6
MemAD8	1	1	1	1	1	1	30	1	2	3	early	†	3
MemAD9	1	1	2	1	2	1	30	2	5	9	early	†	4
MemAD10	2	2	2	2	4	2	30	2	3	8	late	72	7
MemAD11	3	3	3	3	3	3	30	2	4	13	late	72	9
MemAD12	1	1	2	1	2	1	4	1	2	3	early	72	4
MemAD13	2	1	2	1	2	2	5	1	2	3	early	72	5
MemAD14	2	2	2	1	3	2	6	1	3	4	early	72	6
MemAD15	2	1	2	3	3	3	8	1	2	3	early	72	7
MemAD31	4	4	4	4	4	4	31	30	30	18	late	72	12

Table 7.6 IMS T425 internal configuration coding

<sup>†</sup> Provided for static RAM only.

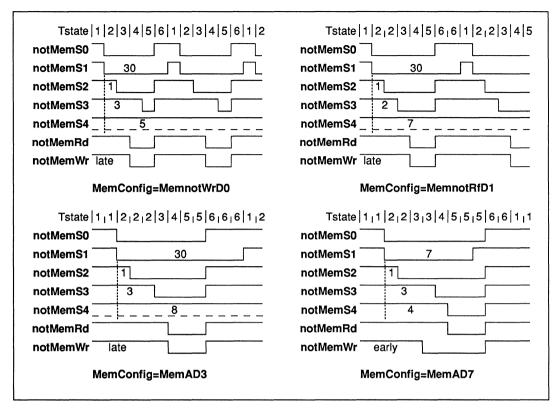


Figure 7.9 IMS T425 internal configuration

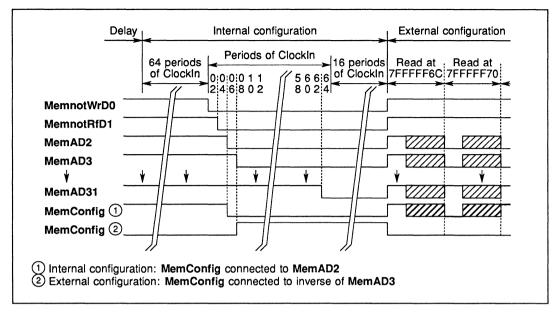


Figure 7.10 IMS T425 internal configuration scan

# 7.10.2 External configuration

If **MemConfig** is held low until **MemnotWrD0** goes low the internal configuration is ignored and an external configuration will be loaded instead. An external configuration scan always follows an internal one, but if an internal configuration occurs any external configuration is ignored.

The external configuration scan comprises 36 successive external read cycles, using the default EMI configuration preset by **MemAD31**. However, instead of data being read on the data bus as for a normal read cycle, only a single bit of data is read on **MemConfig** at each cycle. Addresses put out on the bus for each read cycle are shown in table 7.7, and are designed to address ROM at the top of the memory map. The table shows the data to be held in ROM; data required at the **MemConfig** pin is the inverse of this.

MemConfig is typically connected via an inverter to MemnotWrD0. Data bit zero of the least significant byte of each ROM word then provides the configuration data stream. By switching MemConfig between various data bus lines up to 32 configurations can be stored in ROM, one per bit of the data bus. MemConfig can be permanently connected to a data line or to GND. Connecting MemConfig to GND gives all Tstates configured to four periods; notMemS1 pulse of maximum duration; notMemS2-4 delayed by maximum; refresh interval 72 periods of ClockIn; refresh enabled; late write.

The external memory configuration table 7.7 shows the contribution of each memory address to the 13 configuration fields. The lowest 12 words (#7FFFFF6C to #7FFFF98, fields 1 to 6) define the number of extra periods **Tm** to be added to each **Tstate**. If field 2 is 3 then three extra periods will be added to **T2** to extend it to the maximum of four periods.

The next five addresses (field 7) define the duration of **notMemS1** and the following fifteen (fields 8 to 10) define the delays before strobes **notMemS2-4** become active. The five bits allocated to each strobe allow durations of from 0 to 31 periods **Tm**, as described in strobes page 290.

Addresses #7FFFFFEC to #7FFFFFF4 (fields 11 and 12) define the refresh interval and whether refresh is to be used, whilst the final address (field 13) supplies a high bit to **MemConfig** if a late write cycle is required.

The columns to the right of the coding table show the values of each configuration bit for the four sample

external configuration diagrams. Note the inclusion of period E at the end of T6 in some diagrams. This is inserted to bring the start of the next Tstate T1 to coincide with a rising edge of ProcClockOut (page 288).

Wait states **W** have been added to show the effect of them on strobe timing; they are not part of a configuration. In each case which includes wait states, two wait periods are defined. This shows that if a wait state would cause the start of **T5** to coincide with a falling edge of **ProcClockOut**, another period **Tm** is generated by the EMI to force it to coincide with a rising edge of **ProcClockOut**. This coincidence is only necessary if wait states are added, otherwise coincidence with a falling edge is permitted. Any configuration memory access is only permitted to be extended using wait, up to a total of 14 **ClockIn** periods.

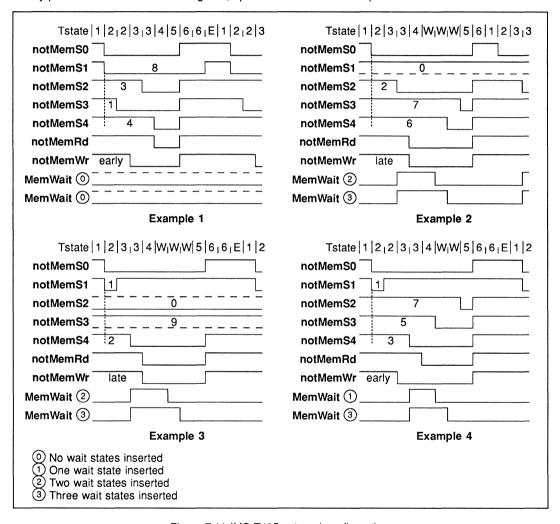


Figure 7.11 IMS T425 external configuration

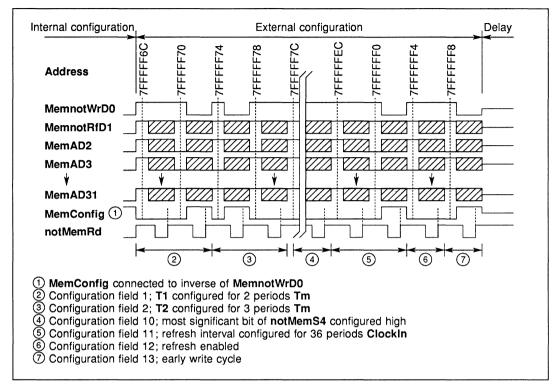


Figure 7.12 IMS T425 external configuration scan

Table 7.7 IMS T425 external configuration coding

Scan	MemAD			Exa	mple	diag	ram
cycle	address	Field	Function	1	2	3	4
1	7FFFFF6C	1	T1 least significant bit	0	0	0	0
2	7FFFFF70	1	T1 most significant bit	0	0	0	0
3	7FFFFF74	2	T2 least significant bit	1	0	0	1 1
4 5 6 7	7FFFFF78	2	T2 most significant bit	0	0	0	0
5	7FFFFF7C	3	T3 least significant bit	1	1	1	1
6	7FFFFF80	3	T3 most significant bit	0	0	0	0
7	7FFFFF84	4	T4 least significant bit	0	0	0	0
8	7FFFFF88	4	T4 most significant bit	0	0	0	0
9	7FFFFF8C	5	T5 least significant bit	0	0	0	0
10	7FFFFF90	5	T5 most significant bit	0	0	0	0
11	7FFFFF94	6	T6 least significant bit	1	0	1	1
12	7FFFFF98	6	T6 most significant bit	0	0	0	0
13	7FFFFF9C	7	notMemS1 least significant bit	0	0	1	1
14	7FFFFFA0	7	-	0	0	0	0
15	7FFFFFA4	7		0	0	0	0
16	7FFFFFA8	7	·	1	0	0	0
17	7FFFFFAC	7	notMemS1 most significant bit	0	0	0	0
18	7FFFFB0	8	notMemS2 least significant bit	1	0	0	1
19	7FFFFB4	8		1	1	0	1
20	7FFFFB8	8		0	0	0	1
21	7FFFFBC	8		0	0	0	0
22	7FFFFFC0	8	notMemS2 most significant bit	0	0	0	0
23	7FFFFFC4	9	notMemS3 least significant bit	1	1	1	1
24	7FFFFFC8	9	_	0	1	0	0
25	7FFFFFCC	9	↓ ↓	0	1	0	1
26	7FFFFFD0	9		0	0	1	0
27	7FFFFD4	9	notMemS3 most significant bit	0	0	0	0
28	7FFFFFD8	10	notMemS4 least significant bit	0	0	0	1
29	7FFFFDC	10		0	1	1	1
30	7FFFFE0	10		1	1	0	0
31	7FFFFE4	10	_	0	0	0	0
32	7FFFFE8	10	notMemS4 most significant bit	0	0	0	0
33	7FFFFEC	11	Refresh Interval least significant bit	-	-	-	-
34	7FFFFFF0	11	Refresh Interval most significant bit	-	-	-	-
35	7FFFFFF4	12	Refresh Enable	-	-	-	-
36	7FFFFFF8	13	Late Write	0	1	1	0

Refresh interval	Interval in μs	Field 11 encoding	Complete cycle (mS)
18	3.6	00	0.922
36	7.2	01	1.843
54	10.8	10	2.765
72	14.4	11	3.686

Table 7.8 IMS T425 memory refresh configuration coding

Refresh intervals are in periods of ClockIn and ClockIn frequency is 5 MHz:

Interval = 18 \* 200 = 3600 ns

Refresh interval is between successive incremental refresh addresses. Complete cycles are shown for 256 row DRAMS.

Table 7.9 Memory configuration

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTE
TMCVRdH	Memory configuration data setup	25			ns	
TRdHMCX	Memory configuration data hold	0			ns	
TS0LRdH	notMemS0 to configuration data read	<b>a</b> -12		a+12	ns	1

#### Notes

1 a is 16 periods Tm.

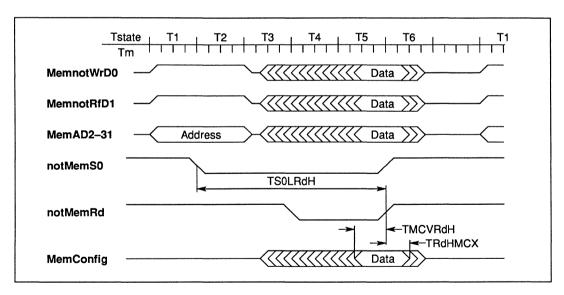


Figure 7.13 IMS T425 external configuration read cycle timing

#### 7.11 RefreshPending

When high, this pin signals that a refresh cycle is pending. It remains high until the refresh cycle is started by the transputer. The minimum time for the **RefreshPending** pin to be high is for one cycle of **ProcClockOut** (two periods **Tm**), when the EMI was not about to perform a memory read or write. If the EMI was held in the tristate condition with **MemGranted** asserted, then **RefreshPending** will be asserted when the refresh controller in the EMI is ready to perform a refresh. **MemReq** may be re-asserted any time after the commencement of the refresh cycle. **RefreshPending** changes state near the rising edge of **ProcClockOut** and can therefore be sampled by the falling edge of **ProcClockOut**.

If no DMA is active then refresh will be performed following the end of the current internal or external memory cycle. If DMA is active the transputer will wait for DMA to terminate before commencing the refresh cycle. Unlike **MemnotRfD1**, **RefreshPending** is never tristated and can thus be interrogated by the DMA device; the DMA cycle can then be suspended, at the discretion of the DMA device, to allow refresh to take place.

The simple circuit of Figure 7.14 will suspend DMA requests from the external logic when **RefreshPending** is asserted, so that a memory refresh cycle can be performed. DMA is restored on completion of the refresh cycle. The transputer will not perform an external memory cycle other than a refresh cycle, using this method, until the requesting device removes its DMA request.

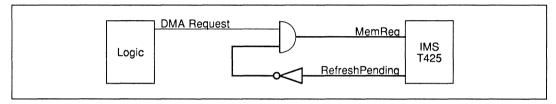


Figure 7.14 IMS T425 refresh with DMA

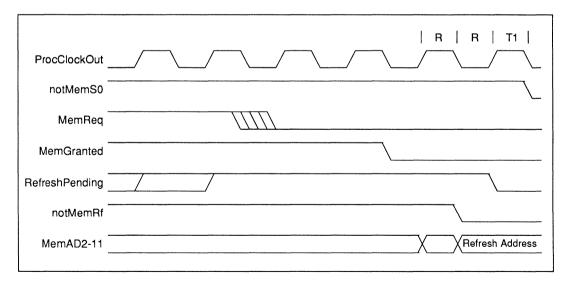


Figure 7.15 IMS T425 RefreshPending timing

#### 7.12 notMemRf

The IMS T425 can be operated with memory refresh enabled or disabled. The selection is made during memory configuration, when the refresh interval is also determined. Refresh cycles do not interrupt internal memory accesses, although the internal addresses cannot be reflected on the external bus during refresh.

When refresh is disabled no refresh cycles occur. During the post-**Reset** period eight dummy refresh cycles will occur with the appropriate timing but with no bus or strobe activity.

A refresh cycle uses the same basic external memory timing as a normal external memory cycle, except that it starts two periods **Tm** before the start of **T1**. If a refresh cycle is due during an external memory access, it will be delayed until the end of that external cycle. Two extra periods **Tm** (periods **R** in the diagram) will then be inserted between the end of **T6** of the external memory cycle and the start of **T1** of the refresh cycle itself. The refresh address and various external strobes become active approximately one period **Tm** before **T1**. Bus signals are active until the end of **T2**, whilst **notMemRf** remains active until the end of **T6**.

For a refresh cycle, MemnotRfD1 goes low before notMemRf goes low and MemnotWrD0 goes high with the same timing as MemnotRfD1. All the address lines share the same timing, but only MemAD2-11 give the refresh address. MemAD12-30 stay high during the address period, whilst MemAD31 remains low. Refresh cycles generate strobes notMemS0-4 with timing as for a normal external cycle, but notMemRd and notMemWrB0-3 remain high. MemWait operates normally during refresh cycles.

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTE
TRfLRfH	Refresh pulse width low	<b>a</b> -2		<b>a</b> +9	ns	1
TRaVS0L	Refresh address setup before notMemS0	<b>b</b> -12			ns	
TRfLS0L	Refresh indicator setup before notMemS0	<b>b</b> -4	b	<b>b</b> +6	ns	2

Table 7.10 Memory refresh

#### Notes

- 1 a is total Tmx+Tm.
- 2 b is total T1+Tm where T1 can be from one to four periods Tm in length.

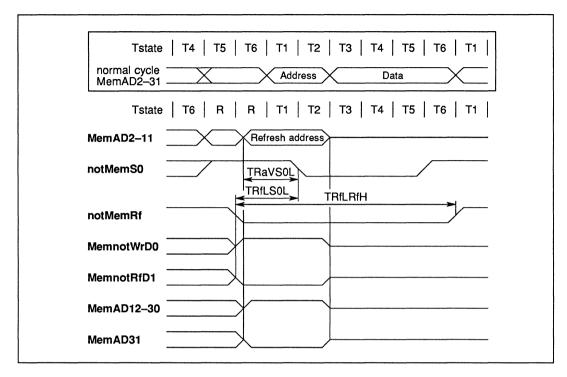


Figure 7.16 IMS T425 refresh cycle timing

#### 7.13 MemWait

Taking MemWait high with the timing shown will extend the duration of T4. MemWait is sampled relative to the falling edge of ProcClockOut during a T3 period, and should not change state in this region. By convention, notMemS4 is used to synchronize wait state insertion. If this or another strobe is used, its delay should be such as to take the strobe low an even number of periods Tm after the start of T1, to coincide with a rising edge of ProcClockOut.

**MemWait** may be kept high indefinitely, although if dynamic memory refresh is used it should not be kept high long enough to interfere with refresh timing. **MemWait** operates normally during all cycles, including refresh and configuration cycles. It does not affect internal memory access in any way.

If the start of **T5** would coincide with a falling edge of **ProcClockOut** an extra wait period **Tm** (**EW**) is generated by the EMI to force coincidence with a rising edge. Rising edge coincidence is only forced if wait states are added, otherwise coincidence with a falling edge is permitted.

Table 7.11 Memory wait

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTE
TPCLWtH	Wait setup	-(0.5 <b>Tm</b> +9)			ns	1,2
TPCLWtL	Wait hold	0.5 <b>Tm</b> +10			ns	1,2
TWtLWtH	Delay before re-assertion of Wait	2 <b>Tm</b>				

## Notes

- 1 ProcClockOut load should not exceed 50pf.
- 2 If wait period exceeds refresh interval, refresh cycles will be lost.

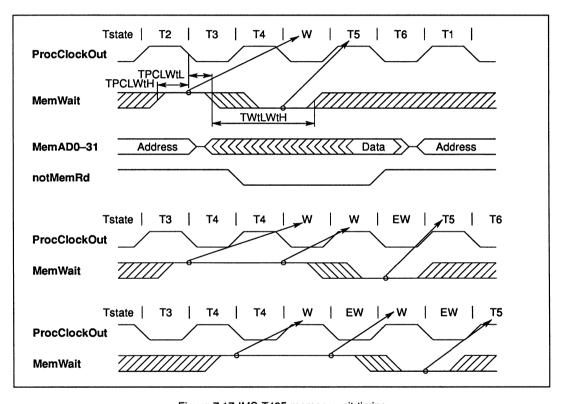


Figure 7.17 IMS T425 memory wait timing

## 7.14 MemReg, MemGranted

Direct memory access (DMA) can be requested at any time by taking the asynchronous **MemReq** input high. The transputer samples **MemReq** during the final period **Tm** of **T6** of both refresh and external memory cycles. To guarantee taking over the bus immediately following either, **MemReq** must be set up at least two periods **Tm** before the end of **T6**. In the absence of an external memory cycle, **MemReq** is sampled during every low period of **ProcClockOut**. The address bus is tristated two periods **Tm** after the **ProcClockOut** rising edge which follows the sample. **MemGranted** is asserted one period **Tm** after that.

Removal of **MemReq** is sampled during each low period of **ProcClockOut** and **MemGranted** is removed synchronously with the next falling edge of **ProcClockOut**. If accurate timing of DMA is required, **MemReq** should be set low coincident with a falling edge of **ProcClockOut**. Further external bus activity, either refresh, external cycles or reflection of internal cycles, will commence at the next rising edge of **ProcClockOut**.

Strobes are left in their inactive states during DMA. DMA cannot interrupt a refresh or external memory cycle, and outstanding refresh cycles will occur before the bus is released to DMA. DMA does not interfere with internal memory cycles in any way, although a program running in internal memory would have to wait for the end of DMA before accessing external memory. DMA cannot access internal memory. If DMA extends longer than one refresh interval (Memory Refresh Configuration Coding, table 7.8), the DMA user becomes responsible for refresh. DMA may also inhibit an internally running program from accessing external memory.

DMA allows a bootstrap program to be loaded into external RAM ready for execution after reset. If **MemReq** is held high throughout reset, **MemGranted** will be asserted before the bootstrap sequence begins. **MemReq** must be high at least one period **TDCLDCL** of **ClockIn** before **Reset**. The circuit should be designed to ensure correct operation if **Reset** could interrupt a normal DMA cycle.

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTE
TMRHMGH	Memory request response time	4 <b>Tm</b> -2ns		7 <b>Tm</b> +7ns		1
TMRLMGL	Memory request end response time	2 <b>Tm</b> -2ns		5 <b>Tm</b> +22ns		
TADZMGH	Bus tristate before memory granted	Tm-2ns		Tm+22ns		
TMGLADV	Bus active after end of memory granted	-10ns		Tm+2ns		

Table 7.12 Memory request

#### Notes

1 These values assume no external memory cycle is in progress. If an external cycle is active, maximum time could be (1 EMI cycle Tmx)+(1 refresh cycle TRfLRfH)+(6 periods Tm).

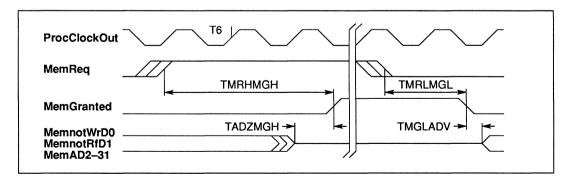


Figure 7.18 IMS T425 memory request timing

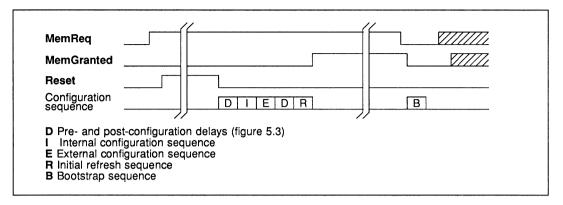


Figure 7.19 IMS T425 DMA sequence at reset

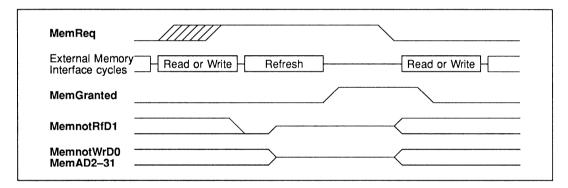


Figure 7.20 IMS T425 operation of MemReg, MemGranted with external, refresh memory cycles

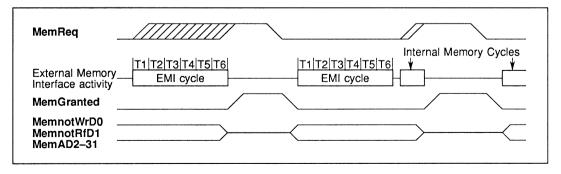


Figure 7.21 IMS T425 operation of MemReq, MemGranted with external, internal memory cycles

## 8 Events

**EventReq** and **EventAck** provide an asynchronous handshake interface between an external event and an internal process. When an external event takes **EventReq** high the external event channel (additional to the external link channels) is made ready to communicate with a process. When both the event channel and the process are ready the processor takes **EventAck** high and the process, if waiting, is scheduled. **EventAck** is removed after **EventReq** goes low.

**EventWaiting** is asserted high by the transputer when a process executes an input on the event channel; typically with the occam **EVENT** ? **ANY** instruction. It remains high whilst the transputer is waiting for or servicing **EventReq** and is returned low when **EventAck** goes high. The **EventWaiting** pin changes near the falling edge of **ProcClockOut** and can therefore be sampled by the rising edge of **ProcClockOut**.

The **EventWaiting** pin can only be asserted by executing an *in* instruction on the event channel. The **EventWaiting** pin is not asserted high when an enable channel (*enbc*) instruction is executed on the Event channel (during an ALT construct in occam, for example). The **EventWaiting** pin can be asserted by executing the occam input on the event channel (such as **Event? ANY**), provided that this does not occur as a guard in an alternative process. The **EventWaiting** pin can not be used to signify that an alternative process (ALT) is waiting on an input from the event channel.

**EventWaiting** allows a process to control external logic; for example, to clock a number of inputs into a memory mapped data latch so that the event request type can be determined. This function is not available on the IMS T414 and IMS T800.

Only one process may use the event channel at any given time. If no process requires an event to occur **EventAck** will never be taken high. Although **EventReq** triggers the channel on a transition from low to high, it must not be removed before **EventAck** is high. **EventReq** should be low during **Reset**; if not it will be ignored until it has gone low and returned high. **EventAck** is taken low when **Reset** occurs.

If the process is a high priority one and no other high priority process is running, the latency is as described on page 269. Setting a high priority task to wait for an event input allows the user to interrupt a transputer program running at low priority. The time taken from asserting **EventReq** to the execution of the microcode interrupt handler in the CPU is four cycles. The following functions take place during the four cycles:

- Cycle 1 Sample EventReq at pad on the rising edge of ProcClockOut and synchronise.
- Cycle 2 Edge detect the synchronised EventReq and form the interrupt request.
- Cycle 3 Sample interrupt vector for microcode ROM in the CPU.
- **Cycle 4** Execute the interrupt routine for Event rather than the next instruction.

Table 8.1 Event

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTE
TVHKH	Event request response	0			ns	
TKHVL	Event request hold	0			ns	
TVLKL	Delay before removal of event acknowledge	0		6 <b>Tm</b> +7ns		
TKLVH	Delay before re-assertion of event request	0			ns	
TKHEWL	Event acknowledge to end of event waiting	0			ns	
TKLEWH	End of event acknowledge to event waiting	0		Į.	ns	

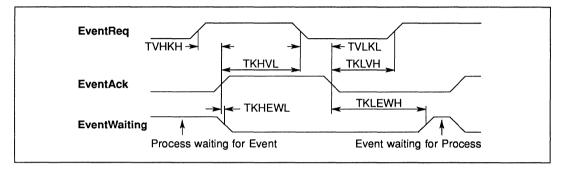


Figure 8.1 IMS T425 event timing

## 9 Links

Four identical INMOS bi-directional serial links provide synchronized communication between processors and with the outside world. Each link comprises an input channel and output channel. A link between two transputers is implemented by connecting a link interface on one transputer to a link interface on the other transputer. Every byte of data sent on a link is acknowledged on the input of the same link, thus each signal line carries both data and control information.

The quiescent state of a link output is low. Each data byte is transmitted as a high start bit followed by a one bit followed by eight data bits followed by a low stop bit. The least significant bit of data is transmitted first. After transmitting a data byte the sender waits for the acknowledge, which consists of a high start bit followed by a zero bit. The acknowledge signifies both that a process was able to receive the acknowledged data byte and that the receiving link is able to receive another byte. The sending link reschedules the sending process only after the acknowledge for the final byte of the message has been received.

The IMS T425 links allow an acknowledge packet to be sent before the data packet has been fully received. This overlapped acknowledge technique is fully compatible with all other INMOS transputer links.

The IMS T425 links support the standard INMOS communication speed of 10 Mbits/sec. In addition they can be used at 5 or 20 Mbits/sec. Links are not synchronised with **ClockIn** or **ProcClockOut** and are insensitive to their phases. Thus links from independently clocked systems may communicate, providing only that the clocks are nominally identical and within specification.

Links are TTL compatible and intended to be used in electrically quiet environments, between devices on a single printed circuit board or between two boards via a backplane. Direct connection may be made between devices separated by a distance of less than 300 millimetres. For longer distances a matched 100 ohm transmission line should be used with series matching resistors **RM**. When this is done the line delay should be less than 0.4 bit time to ensure that the reflection returns before the next data bit is sent.

Buffers may be used for very long transmissions. If so, their overall propagation delay should be stable within the skew tolerance of the link, although the absolute value of the delay is immaterial.

Link speeds can be set by LinkSpecial, Link0Special and Link123Special. The link 0 speed can be set independently. Table 9.1 shows uni-directional and bi-directional data rates in Kbytes/sec for each link speed; LinknSpecial is to be read as Link0Special when selecting link 0 speed and as Link123Special for the others. Data rates are quoted for a transputer using internal memory, and will be affected by a factor depending on the number of external memory accesses and the length of the external memory cycle.

Link	Linkn		Kbytes/sec	
Special	Special	Mbits/sec	Uni	Bi
0	0	10	910	1250
0	1	5	450	670
1	0	10	910	1250
1	1	20	1740	2350

Table 9.1 Speed Settings for Transputer Links

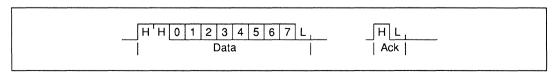


Figure 9.1 IMS T425 link data and acknowledge packets

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTE
TJQr	LinkOut rise time			20	ns	1
TJQf	LinkOut fall time	1	ŀ	10	ns	1
TJDr	LinkIn rise time		1	20	ns	1 1
TJDf	LinkIn fall time	ì	1	20	ns	1
TJQJD	Buffered edge delay	0			ns	
TJBskew	Variation in TJQJD 20 Mbits/	3	ļ	3	ns	2
	10 Mbits/	s		10	ns	2
į	5 Mbits/	s		30	ns	2
CLIZ	LinkIn capacitance @ f=1MH	z	1	7	pF	1
CLL	LinkOut load capacitance		ŀ	50	pF	
RM	Series resistor for 100Ω transmission line		56		ohms	

Table 9.2 Link

#### Notes

- 1 These paramters are sampled, but are not 100% tested.
- 2 This is the variation in the total delay through buffers, transmission lines, differential receivers etc., caused by such things as short term variation in supply voltages and differences in delays for rising and falling edges.

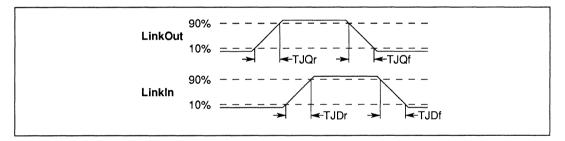


Figure 9.2 IMS T425 link timing

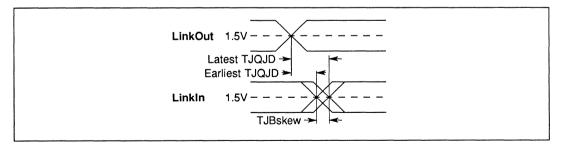


Figure 9.3 IMS T425 buffered link timing

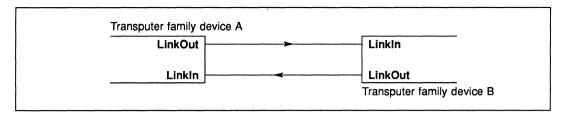


Figure 9.4 IMS T425 Links directly connected

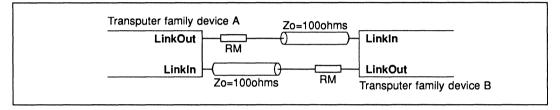


Figure 9.5 IMS T425 Links connected by transmission line

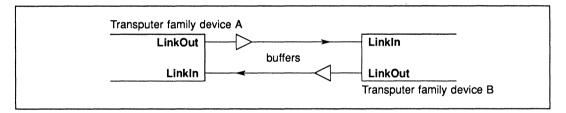


Figure 9.6 IMS T425 Links connected by buffers

# 10 Electrical specifications

#### 10.1 DC electrical characteristics

Table 10.1 Absolute maximum ratings

SYMBOL	PARAMETER	MIN	MAX	UNITS	NOTE
VCC	DC supply voltage	0	7.0	٧	1,2,3
VI, VO	Voltage on input and output pins	-0.5	VCC+0.5	V	1,2,3
11	Input current	1	±25	mA	4
OSCT	Output short circuit time (one pin)	1	1	s	2
TS	Storage temperature	-65	150	°C	2
TA	Ambient temperature under bias	-55	125	°C	2
PDmax	Maximum allowable dissipation	İ	2	W	

#### Notes

- 1 All voltages are with respect to GND.
- 2 This is a stress rating only and functional operation of the device at these or any other conditions beyond those indicated in the operating sections of this specification is not implied. Stresses greater than those listed may cause permanent damage to the device. Exposure to absolute maximum rating conditions for extended periods may affect reliability.
- 3 This device contains circuitry to protect the inputs against damage caused by high static voltages or electrical fields. However, it is advised that normal precautions be taken to avoid application of any voltage higher than the absolute maximum rated voltages to this high impedance circuit. Unused inputs should be tied to an appropriate logic level such as VCC or GND.
- 4 The input current applies to any input or output pin and applies when the voltage on the pin is between GND and VCC.

PARAMETER MAX UNITS NOTE MIN 4.75 5.25 DC supply voltage VCC V 1,2 Input or output voltage 0 Load capacitance on any pin 60 pF èС Operating temperature range IMS T425-S 0 70 3

-55

°C

125

3

Table 10.2 Operating conditions

# TA Notes

SYMBOL

VCC

CL

TA

VI, VO

- 1 All voltages are with respect to GND.
- 2 Excursions beyond the supplies are permitted but not recommended; see DC characteristics.
- 3 Air flow rate 400 linear ft/min transverse air flow.

Operating temperature range IMS T425-M

SYMBOL	PARAME	TER	MIN	MAX	UNITS	NOTE
VIH	High level input voltage		2.0	VCC+0.5	٧	1,2
VIL	Low level input voltage		-0.5	0.8	V	1,2
II	Input current	@ GND <vi<vcc< td=""><td></td><td>±10</td><td>μA</td><td>1,2</td></vi<vcc<>		±10	μA	1,2
VOH	Output high voltage	@ IOH=2mA	VCC-1		V	1,2
VOL	Output low voltage	@ IOL=4mA		0.4	V	1,2
IOS	Output short circuit current	: @ GND <vo<vcc< td=""><td>36</td><td>65</td><td>mA</td><td>1,2,3,6</td></vo<vcc<>	36	65	mA	1,2,3,6
			65	100	mA	1,2,4,6
IOZ	Tristate output current	@ GND <vo<vcc< td=""><td></td><td>±10</td><td><math>\mu</math>A</td><td>1,2</td></vo<vcc<>		±10	$\mu$ A	1,2
PD	Power dissipation			1.0	W	2,5
CIN	Input capacitance	@ f=1MHz	ļ	7	pF	6
COZ	Output capacitance	@ f=1MHz	Ì	10	pF	6

Table 10.3 DC characteristics

#### Notes

- 1 All voltages are with respect to GND.
- 2 Parameters for IMS T425-S measured at 4.75V<VCC<5.25V and 0°C<TA<70°C. Input clock frequency = 5 MHz.
- 3 Current sourced from non-link outputs.
- 4 Current sourced from link outputs.
- 5 Power dissipation varies with output loading and program execution. Power dissipation for processor operating at 20 MHz.
- 6 This parameter is sampled and not 100% tested.

## 10.2 Equivalent circuits

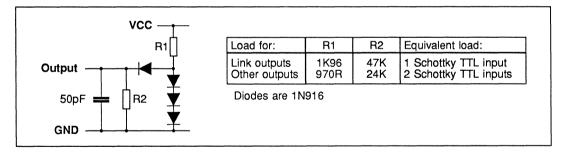


Figure 10.1 Load circuit for AC measurements

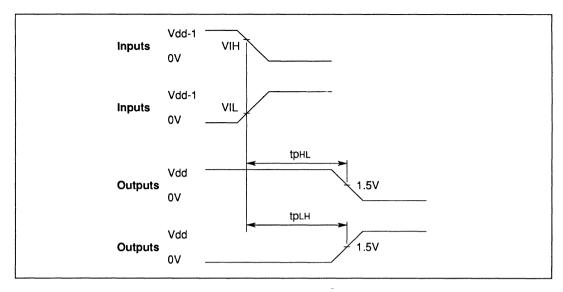


Figure 10.2 AC measurements timing waveforms

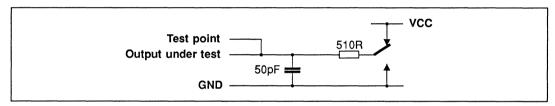


Figure 10.3 Tristate load circuit for AC measurements

## 10.3 AC timing characteristics

	Table	10.4	Input.	output	edaes
--	-------	------	--------	--------	-------

SYMBOL	PARAMETER	MIN	MAX	UNITS	NOTE
TDr	Input rising edges	2	20	ns	1,2
TDf	Input falling edges	2	20	ns	1,2
TQr	Output rising edges		25	ns	1
TQf	Output falling edges		15	ns	1
TS0LaHZ	Address high to tristate	а	<b>a</b> +6	ns	3
TS0LaLZ	Address low to tristate	а	<b>a</b> +6	ns	3

#### Notes

- 1 Non-link pins; see section on links.
- 2 All inputs except Clockin; see section on Clockin.
- 3 a is T2 where T2 can be from one to four periods Tm in length.
  Address lines include MemnotWrD0, MemnotRfD1, MemAD2-31.

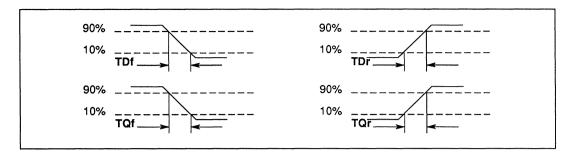


Figure 10.4 IMS T425 input and output edge timing

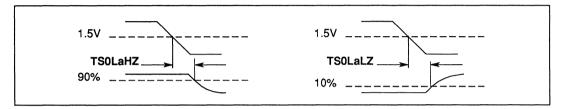
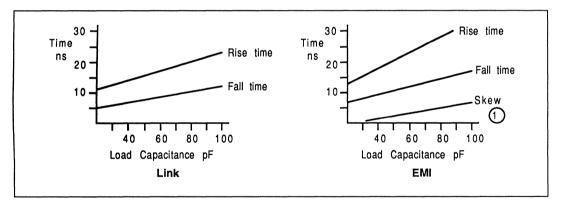


Figure 10.5 IMS T425 tristate timing relative to notMemS0



### Notes

Figure 10.6 Typical rise/fall times

1 Skew is measured between **notMemS0** with a standard load (2 Schottky TTL inputs and 30pF) and **notMemS0** with a load of 2 Schottky TTL inputs and varying capacitance.

## 10.4 Power rating

Internal power dissipation  $P_{INT}$  of transputer and peripheral chips depends on VCC, as shown in figure 10.7.  $P_{INT}$  is substantially independent of temperature.

Total power dissipation  $P_D$  of the chip is

$$P_D = P_{INT} + P_{IO}$$

where  $P_{IO}$  is the power dissipation in the input and output pins; this is application dependent.

Internal working temperature  $T_J$  of the chip is

$$T_J = T_A + \theta J_A * P_D$$

where  $T_A$  is the external ambient temperature in °C and  $\theta J_A$  is the junction-to-ambient thermal resistance in °C/W.  $\theta J_A$  for each package is given in the Packaging Specifications section.

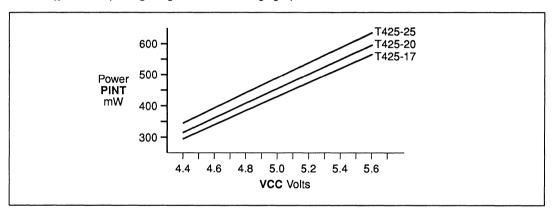


Figure 10.7 IMS T425 internal power dissipation vs VCC

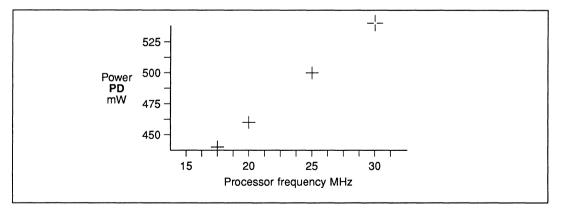


Figure 10.8 IMS T425 typical power dissipation with processor speed

## 11 Performance

The performance of the transputer is measured in terms of the number of bytes required for the program, and the number of (internal) processor cycles required to execute the program. The figures here relate to OCCam programs. For the same function, other languages should achieve approximately the same performance as occam.

#### 11.1 Performance overview

These figures are averages obtained from detailed simulation, and should be used only as an initial guide; they assume operands are of type **INT**. The abbreviations in table 11.1 are used to represent the quantities indicated. In the replicator section of the table, figures in braces {} are not necessary if the number of replications is a compile time constant. To estimate performance, add together the time for the variable references and the time for the operation.

Table 11.1 Key to performance table

np	number of component processes
ne	number of processes earlier in queue
r	1 if INT parameter or array parameter, 0 if not
ts	number of table entries (table size)
w	width of constant in nibbles
р	number of places to shift
Eg	expression used in a guard
Et	timer expression used in a guard
Tb	most significant bit set of multiplier ((-1) if the multiplier is 0)
Tbp	most significant bit set in a positive multiplier when counting from zero ((-1) if the multiplier is 0)
Tbc	most significant bit set in the two's complement of a negative multiplier
nsp	Number of scalar parameters in a procedure
nap	Number of array parameters in a procedure

Table 11.2 Performance

	Size (bytes)	Time (cycles)
Names		
variables		
in expression	1.1+r	2.1+2(r)
assigned to or input to	1.1+r	1.1+(r)
in PROC or FUNCTION call,		` ,
corresponding to an INT parameter	1.1+r	1.1+( <b>r</b> )
channels	1.1	2.1
Array Variables (for single dimension arrays)		
constant subscript	0	0
variable subscript	5.3	7.3
expression subscript	5.3	7.3
·	0.0	7.0
Declarations		
CHAN OF protocol	3.1	3.1
[size]CHAN OF protocol	9.4	2.2 + 20.2* <b>size</b>
PROC	body+2	0
Primitives		
assignment	0	0
input	4	26.5
output	1	26
STOP	2	25
SKIP	0	0
Arithmetic operators		
+ -	1	1
*	2	39
/	2	40
REM	2	38
» «	2	3+ <b>p</b>
Modulo Arithmetic operators		•
PLUS	2	2
MINUS	1 1 .	1
TIMES (fast multiply, positive operand)		4+Tbp
TIMES (fast multiply, positive operand)		5+Tbc
	'	J+150
Boolean operators		
OR NOT	4	8
AND NOT	1	2
Comparison operators		
= constant	0	1
= variable	2	3
<> constant	1	3
<> variable	3	5
> <	1	2
>= <=	2	4
Bit operators		
/\ \/ × ~	2	2
Expressions		
constant in expression	w	w
check if error	4	6 W
CHECK II CHUI	1	L

	Size (bytes)	Time (cycles)
Timers		
timer input	2	3
timer AFTER		!
if past time	2	4
with empty timer queue	2 2 2	31
non-empty timer queue	2	38+ <b>ne</b> +9
ALT (timer)		
with empty timer queue	6	52
non-empty timer queue	6	59+ <b>ne</b> ∗9
timer alt guard	8+2 <b>Eg</b> +2 <b>Et</b>	34+2Eg+2Et
Constructs		
SEQ	0	0
IF	1.3	1.4
if guard	3	4.3
ALT (non timer)	6	26
alt channel guard	10.2+2 <b>Eg</b>	20+2 <b>Eg</b>
skip alt guard	8+2 <b>Eg</b>	10+2 <b>Eg</b>
PAR	11.5+( <b>np</b> -1)*7.5	19.5+( <b>np</b> -1)*30.5
WHILE	4	12
Procedure or function call		
	3.5+(nsp-2)*1.1	16.5+(nsp-2)*1.1
	+nap*2.3	+nap∗2.3
Replicators		
replicated SEQ	7.3{+5.1}	(-3.8)+15.1*count{+7.1}
replicated IF	12.3{+5.1}	(-2.6)+19.4*count{+7.1}
replicated <b>ALT</b>	24.8{+10.2}	25.4+33.4*count{+14.2}
replicated timer ALT	24.8{+10.2}	62.4+33.4*count{+14.2}
replicated PAR	39.1{+5.1}	(-6.4)+70.9*count{+7.1}

Table 11.3 Performance

#### 11.2 Fast multiply, TIMES

The IMS T425 has a fast integer multiplication instruction *product*. For a positive multiplier its execution time is 4+**Tbp** cycles, and for a negative multiplier 5+**Tbc** cycles (table 11.1). The time taken for a multiplication by zero is 3 cycles.

Implementations of high level languages on the transputer may take advantage of this instruction. For example, the OCCAM modulo arithmetic operator **TIMES** is implemented by the instruction and the right-hand operand is treated as the multiplier. The fast multiplication instruction is also used in high level language implementations for the multiplication implicit in multi-dimensional array access.

#### 11.3 Arithmetic

A set of functions are provided within the development system to support the efficient implementation of multiple length integer arithmetic and floating point arithmetic. In table 11.4 n gives the number of places shifted and all arguments and results are assumed to be local. Full details of these functions are provided in the OCCAM reference manual, supplied as part of the development system and available as a separate publication.

When calculating the execution time of the predefined maths functions, no time needs to be added for calling overhead. These functions are compiled directly into special purpose instructions which are designed to support the efficient implementation of multiple length integer arithmetic and floating point arithmetic.

11 Performance 323

Function		Cycles	+ cycles for parameter access †
LONGADD		2	7
LONGSUM		3	8
LONGSUB		2	7
LONGDIFF		3	8
LONGPROD		34	8
LONGDIV		36	8
SHIFTRIGHT	(n<32)	4+n	8
	( <b>n</b> >=32)	n-27	
SHIFTLEFT	(n<32)	4+n	8
	( <b>n</b> >=32)	n-27	
NORMALISE	(n<32)	<b>n</b> +6	7
	( <b>n</b> >=32)	n-25	
	(n=64)	4	
ASHIFTRIGHT		SHIFTRIGHT+2	5
ASHIFTLEFT		SHIFTLEFT+4	5
ROTATERIGHT		SHIFTRIGHT	7
ROTATELEFT		SHIFTLEFT	7
FRACMUL		LONGPROD+4	5

Table 11.4 Arithmetic performance

# 11.4 Floating point operations

Floating point operations for the IMS T425 are provided by a run-time package. This requires approximately 400 bytes of memory for the single length arithmetic operations, and 2500 bytes for the double length arithmetic operations. Table 11.5 summarizes the estimated performance of the package.

							Processo	or cycles		
							IMS T425			
							Typical	Worst		
REAL32	+	_					230	300		
	*						200	240		
	/						245	280		
	<	>	=	>=	<=	<b>&lt;&gt;</b>	60	60		
REAL 64	+	_					565	700		
	*						760	940		
	/						1115	1420		
	<	>	=	>=	<=	<b>&lt;&gt;</b>	60	60		

Table 11.5 IMS T425 floating point operations performance

<sup>†</sup> Assuming local variables.

## 11.4.1 Special purpose functions and procedures

The functions and procedures given in tables 11.7 and 11.8 are provided by the development system to give access to the special instructions available on the IMS T425. Table 11.6 shows the key to the table.

Table 11.6 Key to special performance table

Tb	most significant bit set in the word counting from zero
n	number of words per row (consecutive memory locations)
r	number of rows in the two dimensional move
nr	number of bits to reverse

Table 11.7 Special purpose functions performance

Function	Cycles	+ cycles for parameter access †
BITCOUNT	2+ <b>Tb</b>	2
CRCBYTE	11	8
CRCWORD	35	8
BITREVNBIT	5+nr	4
BITREVWORD	36	2

† Assuming local variables.

Table 11.8 Special purpose procedures performance

Procedure	Cycles	+ cycles for parameter access †
MOVE 2D	8+(2n+23)*r	8
DRAW2D	8+(2n+23)*r	8
CLIP2D	8+(2 <b>n</b> +23)* <b>r</b>	8

† Assuming local variables.

## 11.5 Effect of external memory

Extra processor cycles may be needed when program and/or data are held in external memory, depending both on the operation being performed, and on the speed of the external memory. After a processor cycle which initiates a write to memory, the processor continues execution at full speed until at least the next memory access.

Whilst a reasonable estimate may be made of the effect of external memory, the actual performance will depend upon the exact nature of the given sequence of operations.

External memory is characterized by the number of extra processor cycles per external memory cycle, denoted as **e**. For the IMS T425, with the fastest external memory the value of **e** is 2; a typical value for a large external memory is 5.

If program is stored in external memory, and e has the value 2 or 3, then no extra cycles need be estimated for linear code sequences. For larger values of e, the number of extra cycles required for linear code sequences may be estimated at (e-3)/4. A transfer of control may be estimated as requiring e+3 cycles.

These estimates may be refined for various constructs. In table 11.9 n denotes the number of components in a construct. In the case of **IF**, the n'th conditional is the first to evaluate to **TRUE**, and the costs include the costs of the conditionals tested. The number of bytes in an array assignment or communication is denoted by **b**.

	IMS T	425
	Program off chip	Data off chip
Boolean expressions	<b>e</b> -2	0
IF	3 <b>en</b> -8	en
Replicated IF	(6 <b>e</b> -4) <b>n</b> +7	(5 <b>e</b> -2) <b>n</b> +8
Replicated SEQ	(3e-3)n+2	(4 <b>e</b> -2)n
PAR	(3 <b>e</b> -1) <b>n</b> +8	3en+4
Replicated PAR	(10 <b>e</b> -8) <b>n</b> +8	16 <b>en</b> -12
ALT	(2 <b>e</b> -4) <b>n</b> +6 <b>e</b>	(2e-2)n+10e-8
Array assignment and communication in one transputer	Ó	max (2e, e(b/2)

Table 11.9 External memory performance

The following simulation results illustrate the effect of storing program and/or data in external memory. The results are normalized to 1 for both program and data on chip. The first program (Sieve of Erastosthenes) is an extreme case as it is dominated by small, data access intensive loops; it contains no concurrency, communication, or even multiplication or division. The second program is the pipeline algorithm for Newton Raphson square root computation.

	Program	e=2	e=3	e=4	e=5	On chip
Program off chip	1	1.3	1.5	1.7	1.9	1
	2	1.1	1.2	1.2	1.3	1
Data off chip	1	1.5	1.8	2.1	2.3	1
•	2	1.2	1.4	1.6	1.7	1
Program and data off chip	1 1	1.8	2.2	2.7	3.2	1
	2	1.3	1.6	1.8	2.0	1

Table 11.10 IMS T425 external memory performance

## 11.6 Interrupt latency

If the process is a high priority one and no other high priority process is running, the latency is as described in table 11.11. The timings given are in full processor cycles **TPCLPCL**; the number of **Tm** states is also given where relevant. Maximum latency assumes all memory accesses are internal ones.

СУ
)

	Тур	ical	Maximum		
	TPCLPCL Tm		TPCLPCL	Tm	
IMS T425	19	38	53	116	

# 12 Package specifications

## 12.1 84 pin grid array package

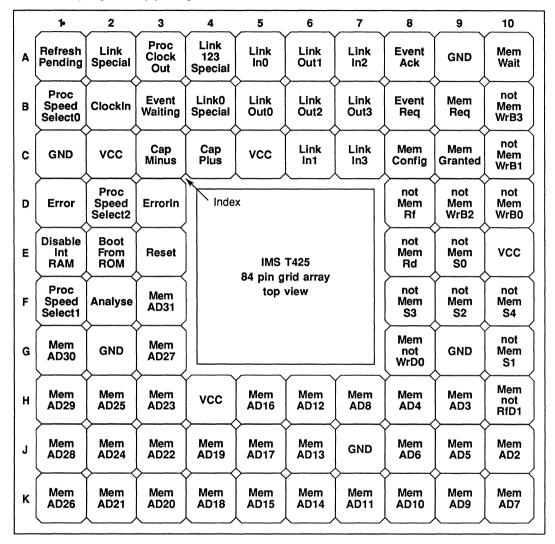


Figure 12.1 IMS T425 84 pin grid array package pinout

Figure 12.2 84 pin grid array package dimensions

Table 12.1	84 pin	grid array	package	dimensions
------------	--------	------------	---------	------------

	Millimetres		Inches		
DIM	NOM	TOL	NOM	TOL	Notes
Α	26.924	±0.254	1.060	±0.010	
В	17.019	±0.127	0.670	±0.005	
С	2.456	±0.278	0.097	±0.011	
D	4.572	±0.127	0.180	±0.005	
E	3.302	±0.127	0.130	$\pm 0.005$	
F	0.457	±0.025	0.018	±0.002	Pin diameter
G	1.143	±0.127	0.045	±0.005	Flange diameter
ĸ	22.860	±0.127	0.900	±0.005	
L	2.540	±0.127	0.100	$\pm 0.005$	
М	0.508		0.020		Chamfer

Package weight is approximately 7.2 grams

Table 12.2 84 pin grid array package junction to ambient thermal resistance

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTE
θJA	At 400 linear ft/min transverse air flow			35	°C/W	

## 12.2 84 pin PLCC J-bend package

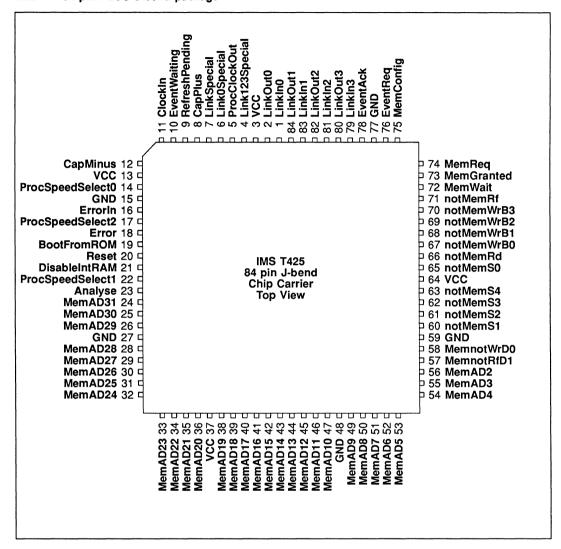


Figure 12.3 IMS T425 84 pin PLCC J-bend package pinout

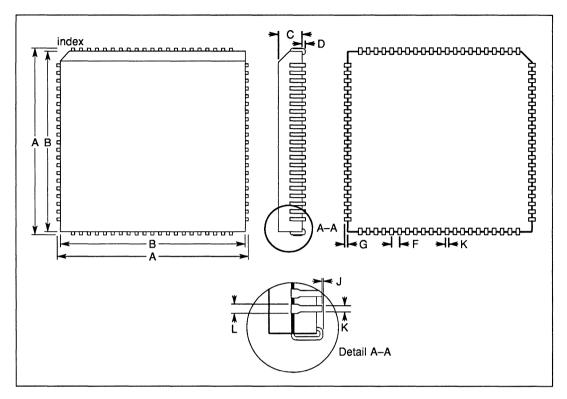


Figure 12.4 84 pin PLCC J-bend package dimensions

Table 12.3 84 pin PLCC J-bend package dimensions

	Millimetres		Inches		
DIM	NOM	TOL	МОМ	TOL	Notes
Α	30.226	±0.127	1.190	±0.005	
В	29.312	±0.127	1.154	±0.005	
С	3.810	±0.127	0.150	±0.005	
D	0.508	±0.127	0.020	±0.005	
F	1.270	±0.127	0.050	±0.005	
G	0.457	±0.127	0.018	±0.005	
J	0.000	±0.051	0.000	±0.002	
K	0.457	±0.127	0.018	±0.005	
L	0.762	±0.127	0.030	±0.005	ļ

Package weight is approximately 7.0 grams

Table 12.4 84 pin PLCC J-bend package junction to ambient thermal resistance

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTE
$\theta$ JA	At 400 linear ft/min transverse air flow		35		°C/W	

#### 12.3 84 lead quad cerpack package

The leads are unformed to allow the user to form them to specific requirements.

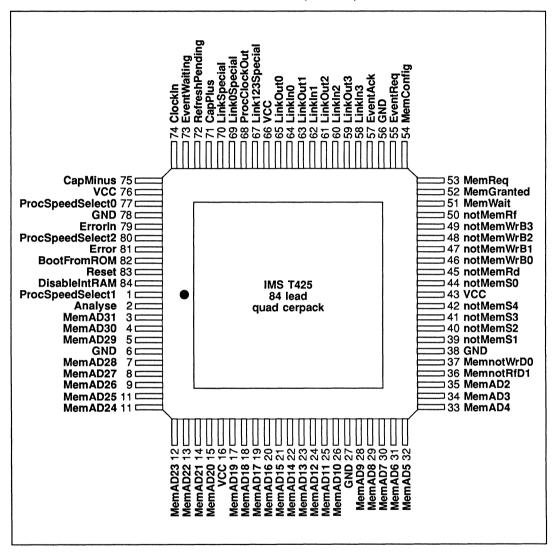


Figure 12.5 IMS T425 84 lead quad cerpack package pinout

Figure 12.6 84 lead quad cerpack package dimensions

	Millimetres		ind	ches	
DIM	NOM	TOL	NOM	TOL	Notes
Α	38.100	±0.508	1.500	±0.020	
В	26.924	±0.305	1.060	±0.012	
C	20.574	±0.203	0.810	±0.008	
D	19.558	±0.254	0.770	±0.010	
E	0.508		0.020		
F	1.270	±0.051	0.050	±0.002	
G	2.489	±0.305	0.098	±0.012	
Н	0.635	±0.076	0.025	±0.003	
J	1.143	±0.102	0.045	±0.004	
K	3.099		0.122		Max.
L	27.940		1.100		Max.
М	0.178	±0.025	0.007	±0.001	

Table 12.5 84 lead quad cerpack package dimensions

# 13 Ordering

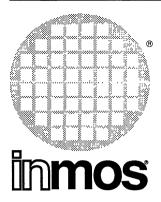
This section indicates the designation of speed and package selections for the various devices. Speed of **ClockIn** is 5 MHz for all parts. Transputer processor cycle time is nominal; it can be calculated more exactly using the phase lock loop factor **PLLx**, as detailed in the external memory section.

For availability contact local INMOS sales office or authorised distributor.

Table 13.1 IMS T425 ordering details

INMOS designation	Processor clock speed	Processor cycle time	PLLx	Package
IMS T425-G17S	17.5 MHz	57 ns	3.5	Ceramic Pin Grid
IMS T425-G20S	20.0 MHz	50 ns	4.0	Ceramic Pin Grid
IMS T425-G25S	25.0 MHz	40 ns	5.0	Ceramic Pin Grid
IMS T425-G30S	30.0 MHz	33 ns	6.0	Ceramic Pin Grid
IMS T425-J17S IMS T425-J20S	17.5 MHz 20.0 MHz	57 ns 50 ns	3.5 4.0	Plastic PLCC J-Bend Plastic PLCC J-Bend
IMS T425-G17M	17.5 MHz	57 ns	3.5	Ceramic Pin Grid MIL Spec
IMS T425-G20M	20.0 MHz	50 ns	4.0	Ceramic Pin Grid MIL Spec
IMS T425-Q17M IMS T425-Q20M	17.5 MHz 20.0 MHz	57 ns 50 ns	3.5 4.0	Quad Cerpack MIL Spec Quad Cerpack MIL Spec

The timing parameters in this datasheet are based on 17 MHz and 20 MHz parts. Data for higher speeds is based on tests on a limited number of samples and may change when full characterisation is completed.



# **IMS T414** transputer

The IMS T425 is recommended for new designs

**Engineering Data** 

## **FEATURES**

32 bit architecture

50 ns internal cycle time

20 MIPS (peak) instruction rate

IMS T414-20 is pin compatible with the IMS T805-20, IMS T800-20 and IMS T425-20

2 Kbytes on-chip static RAM

80 Mbytes/sec sustained data rate to internal memory

4 Gbytes directly addressable external memory

26 Mbytes/sec sustained data rate to external memory

950 ns response to interrupts

Four INMOS serial links 5/10/20 Mbits/sec Bi-directional data rate of 1.6 Mbytes/sec per link

Internal timers of 1  $\mu$ s and 64  $\mu$ s

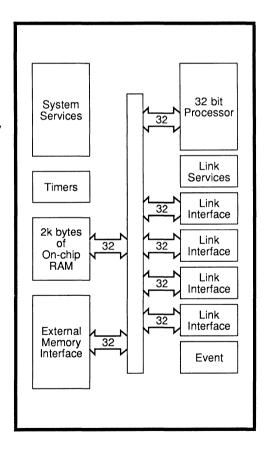
Boot from ROM or communication links

Single 5 MHz clock input

Single +5V ±5% power supply

## **APPLICATIONS**

High speed multi processor systems Real time processing Microprocessor applications Workstations and workstation clusters Image processing Graphics processing Accelerator processors Distributed databases Supercomputers System simulation Digital signal processing Telecommunications Robotics Fault tolerant systems Medical instrumentation Pattern recognition Artificial intelligence



42 1403 05 May 1989

## 1 Introduction

The IMS T414 transputer is a 32 bit CMOS microcomputer with 2 Kbytes on-chip RAM for high speed processing, a configurable memory interface and four standard INMOS communication links. The instruction set achieves efficient implementation of high level languages and provides direct support for the OCCAM model of concurrency when using either a single transputer or a network. Procedure calls, process switching and typical interrupt latency are sub-microsecond. The IMS T414 provides high performance arithmetic and microcode support for floating point operations. A device running at 20 MHz achieves an instruction throughput of 10 MIPS.

For convenience of description, the IMS T414 operation is split into the basic blocks shown in figure 1.1.

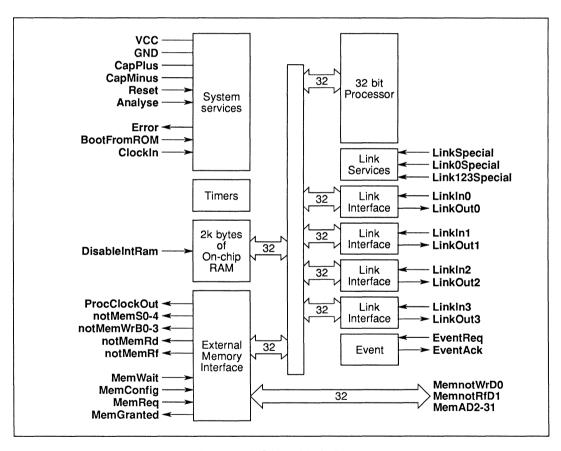


Figure 1.1 IMS T414 block diagram

1 Introduction 335

The IMS T414 can directly access a linear address space of 4 Gbytes. The 32 bit wide memory interface uses multiplexed data and address lines and provides a data rate of up to 4 bytes every 150 nanoseconds (26.6 Mbytes/sec) for a 20 MHz device. A configurable memory controller provides all timing, control and DRAM refresh signals for a wide variety of mixed memory systems.

System Services include processor reset and bootstrap control, together with facilities for error analysis.

The INMOS communication links allow networks of transputer family products to be constructed by direct point to point connections with no external logic. The IMS T414 links support the standard operating speed of 10 Mbits/sec, but also operate at 5 or 20 Mbits/sec.

The IMS T414 is designed to implement the OCCam language, detailed in the OCCam Reference Manual, but also efficiently supports other languages such as C, Pascal and Fortran. Access to the transputer at machine level is seldom required, but if necessary refer to the *Transputer Instruction Set - A Compiler Writers' Guide*.

This data sheet supplies hardware implementation and characterisation details for the IMS T414. It is intended to be read in conjunction with the Transputer Architecture chapter, which details the architecture of the transputer and gives an overview of OCCAM.

# 2 Pin designations

Table 2.1 IMS T414 system services

Pin	In/Out	Function
VCC, GND		Power supply and return
CapPlus, CapMinus		External capacitor for internal clock power supply
Clockin	in	Input clock
Reset	in	System reset
Error	out	Error indicator
Analyse	in	Error analysis
BootFromRom	in	Bootstrap from external ROM or from link
DisableIntRAM	in	Disable internal RAM
HoldToGND		Must be connected to GND
DoNotWire		Must not be wired

Table 2.2 IMS T414 external memory interface

Pin	In/Out	Function
ProcClockOut	out	Processor clock
MemnotWrD0	in/out	Multiplexed data bit 0 and write cycle warning
MemnotRfD1	in/out	Multiplexed data bit 1 and refresh warning
MemAD2-31	in/out	Multiplexed data and address bus
notMemRd	out	Read strobe
notMemWrB0-3	out	Four byte-addressing write strobes
notMemS0-4	out	Five general purpose strobes
notMemRf	out	Dynamic memory refresh indicator
MemWait	in	Memory cycle extender
MemReq	in	Direct memory access request
MemGranted	out	Direct memory access granted
MemConfig	in	Memory configuration data input

Table 2.3 IMS T414 event

Pin	In/Out	Function	
EventReq	in	Event request	
EventAck	out	Event request acknowledge	

Table 2.4 IMS T414 link

Pin	In/Out	Function
Linkln0-3	in	Four serial data input channels
LinkOut0-3	out	Four serial data output channels
LinkSpecial	in	Select non-standard speed as 5 or 20 Mbits/sec
Link0Special	in	Select special speed for Link 0
Link123Special	in	Select special speed for Links 1,2,3

Signal names are prefixed by **not** if they are active low, otherwise they are active high. Pinout details for various packages are given on page 393.

# 3 Processor

The 32 bit processor contains instruction processing logic, instruction and work pointers, and an operand register. It directly accesses the high speed 2 Kbyte on-chip memory, which can store data or program. Where larger amounts of memory or programs in ROM are required, the processor has access to 4 Gbytes of memory via the External Memory Interface (EMI).

## 3.1 Registers

The design of the transputer processor exploits the availability of fast on-chip memory by having only a small number of registers; six registers are used in the execution of a sequential process. The small number of registers, together with the simplicity of the instruction set, enables the processor to have relatively simple (and fast) data-paths and control logic. The six registers are:

The workspace pointer which points to an area of store where local variables are kept.

The instruction pointer which points to the next instruction to be executed.

The operand register which is used in the formation of instruction operands.

The A, B and C registers which form an evaluation stack.

A, B and C are sources and destinations for most arithmetic and logical operations. Loading a value into the stack pushes B into C, and A into B, before loading A. Storing a value from A, pops B into A and C into B.

Expressions are evaluated on the evaluation stack, and instructions refer to the stack implicitly. For example, the *add* instruction adds the top two values in the stack and places the result on the top of the stack. The use of a stack removes the need for instructions to respecify the location of their operands. Statistics gathered from a large number of programs show that three registers provide an effective balance between code compactness and implementation complexity.

No hardware mechanism is provided to detect that more than three values have been loaded onto the stack. It is easy for the compiler to ensure that this never happens.

Any location in memory can be accessed relative to the workpointer register, enabling the workspace to be of any size.

Further register details are given in Transputer Instruction Set - A Compiler Writers' Guide.

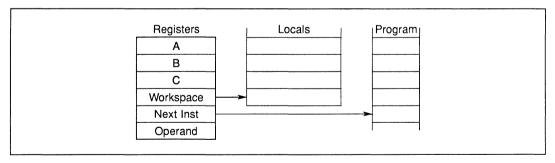


Figure 3.1 Registers

#### 3.2 Instructions

The instruction set has been designed for simple and efficient compilation of high-level languages. All instructions have the same format, designed to give a compact representation of the operations occurring most frequently in programs.

Each instruction consists of a single byte divided into two 4-bit parts. The four most significant bits of the byte are a function code and the four least significant bits are a data value.

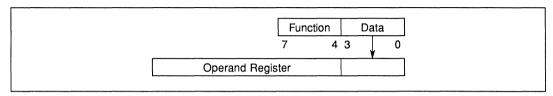


Figure 3.2 Instruction format

## 3.2.1 Direct functions

The representation provides for sixteen functions, each with a data value ranging from 0 to 15. Ten of these, shown in table 3.1, are used to encode the most important functions.

load constant add constant
load local store local load local pointer
load non-local store non-local
jump conditional jump call

Table 3.1 Direct functions

The most common operations in a program are the loading of small literal values and the loading and storing of one of a small number of variables. The *load constant* instruction enables values between 0 and 15 to be loaded with a single byte instruction. The *load local* and *store local* instructions access locations in memory relative to the workspace pointer. The first 16 locations can be accessed using a single byte instruction.

The *load non-local* and *store non-local* instructions behave similarly, except that they access locations in memory relative to the *A* register. Compact sequences of these instructions allow efficient access to data structures, and provide for simple implementations of the static links or displays used in the implementation of high level programming languages such as OCCAM, C, Fortran, Pascal or ADA.

## 3.2.2 Prefix functions

Two more function codes allow the operand of any instruction to be extended in length; *prefix* and *negative* prefix.

All instructions are executed by loading the four data bits into the least significant four bits of the operand register, which is then used as the instruction's operand. All instructions except the prefix instructions end by clearing the operand register, ready for the next instruction.

The *prefix* instruction loads its four data bits into the operand register and then shifts the operand register up four places. The *negative prefix* instruction is similar, except that it complements the operand register before shifting it up. Consequently operands can be extended to any length up to the length of the operand register by a sequence of prefix instructions. In particular, operands in the range -256 to 255 can be represented using one prefix instruction.

3 Processor 339

The use of prefix instructions has certain beneficial consequences. Firstly, they are decoded and executed in the same way as every other instruction, which simplifies and speeds instruction decoding. Secondly, they simplify language compilation by providing a completely uniform way of allowing any instruction to take an operand of any size. Thirdly, they allow operands to be represented in a form independent of the processor wordlength.

#### 3.2.3 Indirect functions

The remaining function code, *operate*, causes its operand to be interpreted as an operation on the values held in the evaluation stack. This allows up to 16 such operations to be encoded in a single byte instruction. However, the prefix instructions can be used to extend the operand of an *operate* instruction just like any other. The instruction representation therefore provides for an indefinite number of operations.

Encoding of the indirect functions is chosen so that the most frequently occurring operations are represented without the use of a prefix instruction. These include arithmetic, logical and comparison operations such as add, exclusive or and greater than. Less frequently occurring operations have encodings which require a single prefix operation.

## 3.2.4 Expression evaluation

Evaluation of expressions sometimes requires use of temporary variables in the workspace, but the number of these can be minimised by careful choice of the evaluation order.

Program	Mnem	onic
x := 0	ldc	0
	stl	X
x := #24	pfix ldc	2
1	ldc	2 4
	stl	X
x := y + z	ldl	y z
	ldl	Z
	add	
	stl	X

Table 3.2 Expression evaluation

## 3.2.5 Efficiency of encoding

Measurements show that about 70% of executed instructions are encoded in a single byte; that is, without the use of prefix instructions. Many of these instructions, such as *load constant* and *add* require just one processor cycle.

The instruction representation gives a more compact representation of high level language programs than more conventional instruction sets. Since a program requires less store to represent it, less of the memory bandwidth is taken up with fetching instructions. Furthermore, as memory is word accessed the processor will receive four instructions for every fetch.

Short instructions also improve the effectiveness of instruction pre-fetch, which in turn improves processor performance. There is an extra word of pre-fetch buffer, so the processor rarely has to wait for an instruction fetch before proceeding. Since the buffer is short, there is little time penalty when a jump instruction causes the buffer contents to be discarded.

# 3.3 Processes and concurrency

A process starts, performs a number of actions, and then either stops without completing or terminates complete. Typically, a process is a sequence of instructions. A transputer can run several processes in parallel (concurrently). Processes may be assigned either high or low priority, and there may be any number of each (page 341).

The processor has a microcoded scheduler which enables any number of concurrent processes to be executed together, sharing the processor time. This removes the need for a software kernel.

At any time, a concurrent process may be

Active

Being executed.

On a list waiting to be executed.

Inactive

Ready to input.

Ready to output.

Waiting until a specified time.

The scheduler operates in such a way that inactive processes do not consume any processor time. It allocates a portion of the processor's time to each process in turn. Active processes waiting to be executed are held in two linked lists of process workspaces, one of high priority processes and one of low priority processes (page 341). Each list is implemented using two registers, one of which points to the first process in the list, the other to the last. In the Linked Process List figure 3.3, process S is executing and P, Q and R are active, awaiting execution. Only the low priority process queue registers are shown; the high priority process ones perform in a similar manner.

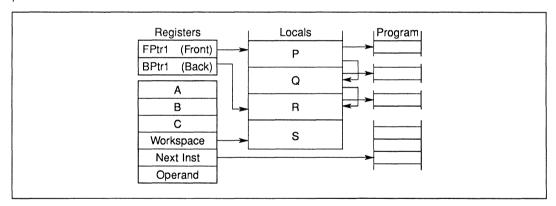


Figure 3.3 Linked process list

Table 3.3 Priority queue control registers

Function	High Priority	Low Priority
Pointer to front of active process list	Fptr0	Fptr1
Pointer to back of active process list	Bptr0	Bptr1

Each process runs until it has completed its action, but is descheduled whilst waiting for communication from another process or transputer, or for a time delay to complete. In order for several processes to operate in parallel, a low priority process is only permitted to run for a maximum of two time slices before it is forcibly descheduled at the next descheduling point (page 344). The time slice period is 5120 cycles of the external 5 MHz clock, giving ticks approximately 1 ms apart.

A process can only be descheduled on certain instructions, known as descheduling points (page 344). As a result, an expression evaluation can be guaranteed to execute without the process being timesliced part way through.

Whenever a process is unable to proceed, its instruction pointer is saved in the process workspace and the next process taken from the list. Process scheduling pointers are updated by instructions which cause scheduling operations, and should not be altered directly. Actual process switch times are less than 1  $\mu$ s, as little state needs to be saved and it is not necessary to save the evaluation stack on rescheduling.

The processor provides a number of special operations to support the process model, including *start process* and *end process*. When a main process executes a parallel construct, *start process* instructions are used to create the necessary additional concurrent processes. A *start process* instruction creates a new process by adding a new workspace to the end of the scheduling list, enabling the new concurrent process to be executed together with the ones already being executed. When a process is made active it is always added to the end of the list, and thus cannot pre-empt processes already on the same list.

The correct termination of a parallel construct is assured by use of the *end process* instruction. This uses a workspace location as a counter of the parallel construct components which have still to terminate. The counter is initialised to the number of components before the processes are *started*. Each component ends with an *end process* instruction which decrements and tests the counter. For all but the last component, the counter is non zero and the component is descheduled. For the last component, the counter is zero and the main process continues.

## 3.4 Priority

The IMS T414 supports two levels of priority. Priority 1 (low priority) processes are executed whenever there are no active priority 0 (high priority) processes.

High priority processes are expected to execute for a short time. If one or more high priority processes are able to proceed, then one is selected and runs until it has to wait for a communication, a timer input, or until it completes processing.

If no process at high priority is able to proceed, but one or more processes at low priority are able to proceed, then one is selected.

Low priority processes are periodically timesliced to provide an even distribution of processor time between computationally intensive tasks.

If there are **n** low priority processes, then the maximum latency from the time at which a low priority process becomes active to the time when it starts processing is 2**n**-2 timeslice periods. It is then able to execute for between one and two timeslice periods, less any time taken by high priority processes. This assumes that no process monopolises the transputer's time; i.e. it has a distribution of descheduling points (page 344).

Each timeslice period lasts for 5120 cycles of the external 5 MHz input clock (approximately 1 ms at the standard frequency of 5 MHz).

If a high priority process is waiting for an external channel to become ready, and if no other high priority process is active, then the interrupt latency (from when the channel becomes ready to when the process starts executing) is typically 19 processor cycles, a maximum of 58 cycles (assuming use of on-chip RAM).

#### 3.5 Communications

Communication between processes is achieved by means of channels. Process communication is point-topoint, synchronised and unbuffered. As a result, a channel needs no process queue, no message queue and no message buffer.

A channel between two processes executing on the same transputer is implemented by a single word in memory; a channel between processes executing on different transputers is implemented by point-to-point

links. The processor provides a number of operations to support message passing, the most important being input message and output message.

The *input message* and *output message* instructions use the address of the channel to determine whether the channel is internal or external. Thus the same instruction sequence can be used for both, allowing a process to be written and compiled without knowledge of where its channels are connected.

The process which first becomes ready must wait until the second one is also ready. A process performs an input or output by loading the evaluation stack with a pointer to a message, the address of a channel, and a count of the number of bytes to be transferred, and then executing an *input message* or *output message* instruction. Data is transferred if the other process is ready. If the channel is not ready or is an external one the process will deschedule.

## 3.6 Timers

The transputer has two 32 bit timer clocks which 'tick' periodically. The timers provide accurate process timing, allowing processes to deschedule themselves until a specific time.

One timer is accessible only to high priority processes and is incremented every microsecond, cycling completely in approximately 4295 seconds. The other is accessible only to low priority processes and is incremented every 64 microseconds, giving exactly 15625 ticks in one second. It has a full period of approximately 76 hours.

Table 3.4 Timer registers

Clock0	Current value of high priority (level 0) process clock
Clock1	Current value of low priority (level 1) process clock
TNextReg0	Indicates time of earliest event on high priority (level 0) timer queue
TNextReg1	Indicates time of earliest event on low priority (level 1) timer queue

The current value of the processor clock can be read by executing a *load timer* instruction. A process can arrange to perform a *timer input*, in which case it will become ready to execute after a specified time has been reached. The *timer input* instruction requires a time to be specified. If this time is in the 'past' then the instruction has no effect. If the time is in the 'future' then the process is descheduled. When the specified time is reached the process is scheduled again.

Figure 3.4 shows two processes waiting on the timer queue, one waiting for time 21, the other for time 31.

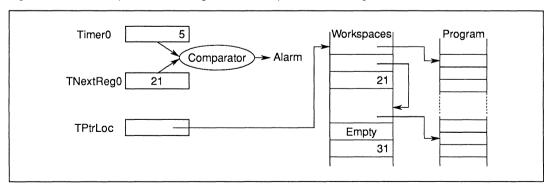


Figure 3.4 Timer registers

# 4 Instruction set summary

The Function Codes table 4.7. gives the basic function code set (page 338). Where the operand is less than 16, a single byte encodes the complete instruction. If the operand is greater than 15, one prefix instruction (pfix) is required for each additional four bits of the operand. If the operand is negative the first prefix instruction will be nfix.

**Function** Memory Mnemonic code code ldc #3 #4 #43 *Idc* #35 is coded as #3 pfix #2 #23 ldc #5 #4 #45 ldc #987 is coded as pfix #9 #2 #29 pfix #8 #2 #28 ldc #7 #4 #47 ldc -31 (ldc #FFFFFE1) is coded as nfix #1 #6 #61 #1 #4 ldc #41

Table 4.1 prefix coding

Tables 4.8 to 4.18 give details of the operation codes. Where an operation code is less than 16 (e.g. add: operation code 05), the operation can be stored as a single byte comprising the operate function code F and the operand (5 in the example). Where an operation code is greater than 15 (e.g. ladd: operation code 16), the prefix function code 2 is used to extend the instruction.

Mnem	nonic	Function code	Memory code
add is coded as	(op. code #5)		#F5
opr	add	#F	#F5
ladd is coded as	(op. code #16)		#21F6
pfix	#1	#2	#21
opr	#6	#F	#F6

Table 4.2 operate coding

The Processor Cycles column refers to the number of periods **TPCLPCL** taken by an instruction executing in internal memory. The number of cycles is given for the basic operation only; where the memory code for an instruction is two bytes, the time for the *prefix* function (one cycle) should be added. For a 20 MHz transputer one cycle is 50 ns. Some instruction times vary. Where a letter is included in the cycles column it is interpreted from table 4.3.

Table 4.3	Instruction	set interpretation	1
-----------	-------------	--------------------	---

Ident	Interpretation				
b	Bit number of the highest bit set in register A. Bit 0 is the least significant bit.				
n	Number of places shifted.				
w	Number of words in the message. Part words are counted as full words. If the message is not word aligned the number of words is increased to include the part words at either end of the message.				

The **DE** column of the tables indicates the descheduling/error features of an instruction as described in table 4.4.

Table 4.4 Instruction features

ldent	Feature	See page:
D	The instruction is a descheduling point	344
E	The instruction will affect the Error flag	344, 354

# 4.1 Descheduling points

The instructions in table 4.5 are the only ones at which a process may be descheduled (page 340). They are also the ones at which the processor will halt if the **Analyse** pin is asserted (page 353).

Table 4.5 Descheduling point instructions

input message	output message	output byte	output word
timer alt wait	timer input	stop on error	alt wait
jump	loop end	end process	stop process

# 4.2 Error instructions

The instructions in table 4.6 are the only ones which can affect the Error flag (page 354) directly.

Table 4.6 Error setting instructions

add	add constant	subtract	remainder
multiply	fractional multiply	divide	
long add set error check word	long subtract testerr check subscript from 0	long divide check single	cflerr check count from 1

Function Code	Memory Code	Mnemonic	Processor Cycles	Name	D E
0	0X	j	3	jump	D
1	1X	ldlp	1	load local pointer	
2	2X	pfix	1	prefix	1 1
3	3X	ldnl	2	load non-local	
4	4X	ldc	1	load constant	
5	5X	idnip	1	load non-local pointer	
6	6X	nfix	1	negative prefix	
7	7X	ldl	2	load local	
8	8X	adc	1	add constant	E
9	9X	call	7	call	
Α	AX	cj	2	conditional jump (not taken)	1 1
			4	conditional jump (taken)	
В	BX	ajw	1	adjust workspace	
С	CX	eqc	2	equals constant	
D	DX	stl	1	store local	
E	EX	stnl	2	store non-local	1
F	FX	opr	-	operate	

Table 4.7 IMS T414 function codes

Table 4.8 IMS T414 arithmetic/logical operation codes

Operation Code	Memory Code	Mnemonic	Processor Cycles	Name	D E
46	24F6	and	1	and	
4B	24FB	or	1	or	
33	23F3	xor	1	exclusive or	
32	23F2	not	1	bitwise not	
41	24F1	shl	n+2	shift left	
40	24F0	shr	n+2	shift right	] ]
05	F5	add	1	add	E
0C	FC	sub	1	subtract	E
53	25F3	mul	38	multiply	E
72	27F2	fmul	35	fractional multiply (no rounding)	E
			40	fractional multiply (rounding)	E
2C	22FC	div	39	divide	E
1F	21FF	rem	37	remainder	E
09	F9	gt	2	greater than	
04	F4	diff	1	difference	
52	25F2	sum	1	sum	
08	F8	prod	b+4	product	

Operation Code	Memory Code	Mnemonic	Processor Cycles	Name	D E
16	21F6	ladd	2	long add	E
38	23F8	Isub	2	long subtract	E
37	23F7	Isum	3	long sum	
4F	24FF	ldiff	3	long diff	
31	23F1	lmul	33	long multiply	
1A	21FA	ldiv	35	long divide	E
36	23F6	Ishl	n+3	long shift left (n<32)	
			n-28	long shift left(n≥32)	
35	23F5	Ishr	n+3	long shift right (n<32)	
			n-28	long shift right (n≥32)	
19	21F9	norm	<b>n</b> +5	normalise (n<32)	
			n-26	normalise (n≥32)	
			3	normalise (n=64)	

Table 4.9 IMS T414 long arithmetic operation codes

Table 4.10 IMS T414 floating point support operation codes

Operation Code	Memory Code	Mnemonic	Processor Cycles	Name	D E
73	27F3	cflerr	3	check floating point error	E
63	26F3	unpacksn	15	unpack single length fp number	
6D	26FD	roundsn	12/15	round single length fp number	
6C	26FC	postnormsn	5/30	post-normalise correction of	
				single length fp number	
71	27F1	ldinf	1	load single length infinity	

Processor cycles are shown as Typical/Maximum cycles.

Table 4.11 IMS T414 general operation codes

Operation Code	Memory Code	Mnemonic	Processor Cycles	Name	D E
00	F0	rev	1	reverse	
3A	23FA	xword	4	extend to word	
56	25F6	cword	5	check word	E
1D	21FD	xdble	2	extend to double	
4C	24FC	csngl	3	check single	E
42	24F2	mint	1	minimum integer	

Operation Code	Memory Code	Mnemonic	Processor Cycles	Name	D E
02	F2	bsub	1	byte subscript	
0A	FA	wsub	2	word subscript	
34	23F4	bcnt	2	byte count	Ì
3F	23FF	wcnt	5	word count	
01	F1	lb	5	load byte	
3B	23FB	sb	4	store byte	
4A	24FA	move	2w+8	move message	

Table 4.12 IMS T414 indexing/array operation codes

Table 4.13 IMS T414 timer handling operation codes

Operation Code	Memory Code	Mnemonic	Processor Cycles	Name	D E
22	22F2	ldtimer	2	load timer	
2B	22FB	tin	30	timer input (time future)	D
			4	timer input (time past)	D
4E	24FE	talt	4	timer alt start	
51	25F1	taltwt	15	timer alt wait (time past)	ם
			48	timer alt wait (time future)	D
47	24F7	enbt	8	enable timer	
2E	22FE	dist	23	disable timer	

Table 4.14 IMS T414 input/output operation codes

Operation Code	Memory Code	Mnemonic	Processor Cycles	Name	D E
07	F7	in	2w+19	input message	D
0B	FB	out	2w+19	output message	D
0F	FF	outword	23	output word	D
0E	FE	outbyte	23	output byte	D
43	24F3	alt	2	alt start	
44	24F4	altwt	5	alt wait (channel ready)	D
			17	alt wait (channel not ready)	D
45	24F5	altend	4	alt end	
49	24F9	enbs	3	enable skip	
30	23F0	diss	4	disable skip	
12	21F2	resetch	3	reset channel	
48	24F8	enbc	7	enable channel (ready)	
			5	enable channel (not ready)	
2F	22FF	disc	8	disable channel	

Operation Code	Memory Code	Mnemonic	Processor Cycles	Name	D E
20	22F0	ret	5	return	
1B	21FB	ldpi	2	load pointer to instruction	
3C	23FC	gajw	2	general adjust workspace	
06	F6	gcall	4	general call	
21	22F1	lend	10	loop end (loop)	D
			5	loop end (exit)	D

Table 4.15 IMS T414 control operation codes

Table 4.16 IMS T414 scheduling operation codes

Operation Code	Memory Code	Mnemonic	Processor Cycles	Name	D E
0D	FD	startp	12	start process	D
03	F3	endp	13	end process	D
39	23F9	runp	10	run process	1 1
15	21F5	stopp	11	stop process	
1E	21FE	ldpri	1	load current priority	

Table 4.17 IMS T414 error handling operation codes

Operation Code	Memory Code	Mnemonic	Processor Cycles	Name	D E
13	21F3	csub0	2	check subscript from 0	Ε
4D	24FD	ccnt1	3	check count from 1	E
29	22F9	testerr	2	test error false and clear (no error)	
			3	test error false and clear (error)	
10	21F0	seterr	1	set error	E
55	25F5	stoperr	2	stop on error (no error)	D
57	25F7	cirhaiterr	1	clear halt-on-error	
58	25F8	sethalterr	1	set halt-on-error	
59	25F9	testhalterr	2	test halt-on-error	

Table 4.18 IMS T414 processor initialisation operation codes

Operation Code	Memory Code	Mnemonic	Processor Cycles	Name	D E
2A	22FA	testpranal	2	test processor analysing	
3E	23FE	saveh	4	save high priority queue registers	
3D	23FD	savel	4	save low priority queue registers	1
18	21F8	sthf	1	store high priority front pointer	1
50	25F0	sthb	1	store high priority back pointer	
1C	21FC	stlf	1	store low priority front pointer	
17	21F7	stlb	1	store low priority back pointer	
54	25F4	sttimer	1	store timer	

# 5 System services

System services include all the necessary logic to initialise and sustain operation of the device. They also include error handling and analysis facilities.

#### 5.1 Power

Power is supplied to the device via the VCC and GND pins. Several of each are provided to minimise inductance within the package. All supply pins must be connected. The supply must be decoupled close to the chip by at least one 100 nF low inductance (e.g. ceramic) capacitor between VCC and GND. Four layer boards are recommended; if two layer boards are used, extra care should be taken in decoupling.

Input voltages must not exceed specification with respect to VCC and GND, even during power-up and power-down ramping, otherwise *latchup* can occur. CMOS devices can be permanently damaged by excessive periods of latchup.

## 5.2 CapPlus, CapMinus

The internally derived power supply for internal clocks requires an external low leakage, low inductance  $1\mu F$  capacitor to be connected between **CapPlus** and **CapMinus**. A ceramic capacitor is preferred, with an impedance less than 3 Ohms between 100 KHz and 20 MHz. If a polarised capacitor is used the negative terminal should be connected to **CapMinus**. Total PCB track length should be less than 50 mm. The connections must not touch power supplies or other noise sources.

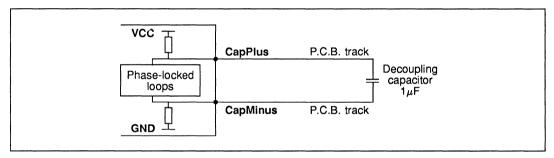


Figure 5.1 Recommended PLL decoupling

#### 5.3 Clockin

Transputer family components use a standard clock frequency, supplied by the user on the **ClockIn** input. The nominal frequency of this clock for all transputer family components is 5 MHz, regardless of device type, transputer word length or processor cycle time. High frequency internal clocks are derived from **ClockIn**, simplifying system design and avoiding problems of distributing high speed clocks externally.

A number of transputer devices may be connected to a common clock, or may have individual clocks providing each one meets the specified stability criteria. In a multi-clock system the relative phasing of **ClockIn** clocks is not important, due to the asynchronous nature of the links. Mark/space ratio is unimportant provided the specified limits of **ClockIn** pulse widths are met.

Oscillator stability is important. ClockIn must be derived from a crystal oscillator; RC oscillators are not sufficiently stable. ClockIn must not be distributed through a long chain of buffers. Clock edges must be monotonic and remain within the specified voltage and time limits.

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTE
TDCLDCH	ClockIn pulse width low	40			ns	1
TDCHDCL		40	Ì		ns	1
TDCLDCL	ClockIn period		200		ns	1,2,4
TDCerror	ClockIn timing error	1	l	±0.5	ns	1,3
TDC1DC2	Difference in ClockIn for 2 linked devices			400	ppm	1,4
TDCr	ClockIn rise time	1	1	10	ns	1,5
TDCf	ClockIn fall time			8	ns	1,5

Table 5.1 Input clock

- 1 These paramters are not tested.
- 2 Measured between corresponding points on consecutive falling edges.
- 3 Variation of individual falling edges from their nominal times.
- 4 This value allows the use of 200ppm crystal oscillators for two devices connected together by a link.
- 5 Clock transitions must be monotonic within the range VIH to VIL (table 10.3).

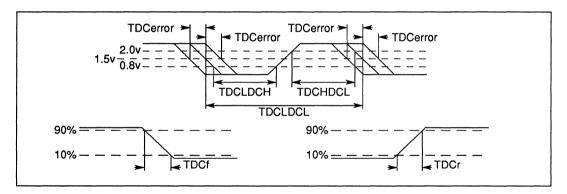


Figure 5.2 ClockIn timing

#### 5.4 Reset

Reset can go high with VCC, but must at no time exceed the maximum specified voltage for VIH. After VCC is valid ClockIn should be running for a minimum period TDCVRL before the end of Reset. The falling edge of Reset initialises the transputer, triggers the memory configuration sequence and starts the bootstrap routine. Link outputs are forced low during reset; link inputs and EventReq should be held low. Memory request (DMA) must not occur whilst Reset is high but can occur before bootstrap (page 376).

After the end of Reset there will be a delay of 144 periods of ClockIn (figure 5.3). Following this, the MemWrD0, MemRfD1 and MemAD2-31 pins will be scanned to check for the existence of a pre-programmed memory interface configuration (page 366). This lasts for a further 144 periods of ClockIn. Regardless of whether a configuration was found, 36 configuration read cycles will then be performed on external memory using the default memory configuration (page 368), in an attempt to access the external configuration ROM. A delay will then occur, its period depending on the actual configuration. Finally eight complete and consecutive refresh cycles will initialise any dynamic RAM, using the new memory configuration. If the memory configuration does not enable refresh of dynamic RAM the refresh cycles will be replaced by an equivalent delay with no external memory activity.

If **BootFromRom** is high bootstrapping will then take place immediately, using data from external memory; otherwise the transputer will await an input from any link. The processor will be in the low priority state.

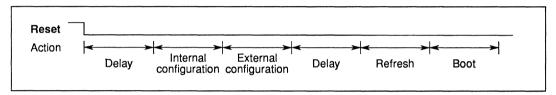


Figure 5.3 IMS T414 post-reset sequence

# 5.5 Bootstrap

The transputer can be bootstrapped either from a link or from external ROM. To facilitate debugging, **Boot-FromRom** may be dynamically changed but must obey the specified timing restrictions. It is sampled once only by the transputer, before the first instruction is executed after **Reset** is taken low.

If **BootFromRom** is connected high (e.g. to **VCC**) the transputer starts to execute code from the top two bytes in external memory, at address #7FFFFFFE. This location should contain a backward jump to a program in ROM. Following this access, **BootFromRom** may be taken low if required. The processor is in the low priority state, and the *W* register points to *MemStart* (page 355).

If **BootFromRom** is connected low (e.g. to **GND**) the transputer will wait for the first bootstrap message to arrive on any one of its links. The transputer is ready to receive the first byte on a link within two processor cycles **TPCLPCL** after **Reset** goes low.

If the first byte received (the control byte) is greater than 1 it is taken as the quantity of bytes to be input. The following bytes, to that quantity, are then placed in internal memory starting at location *MemStart*. Following reception of the last byte the transputer will start executing code at *MemStart* as a low priority process. **BootFromRom** may be taken high after reception of the last byte, if required. The memory space immediately above the loaded code is used as work space. Messages arriving on other links after the control byte has been received and on the bootstrapping link after the last bootstrap byte will be retained until a process inputs from them.

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTE
TPVRH	Power valid before Reset	10			ms	
TRHRL	Reset pulse width high	8			ClockIn	1
TDCVRL	ClockIn running before Reset end	10			ms	2
TAHRH	Analyse setup before Reset	3	l		ms	
TRLAL	Analyse hold after Reset end	1			ClockIn	1
TBRVRL	BootFromRom setup	0			ms	
TRLBRX	BootFromRom hold after Reset	0	Ì		ms	3
TALBRX	BootFromRom hold after Analyse					3

Table 5.2 Reset and Analyse

- 1 Full periods of Clockin TDCLDCL required.
- 2 At power-on reset.
- 3 Must be stable until after end of bootstrap period. See Bootstrap section.

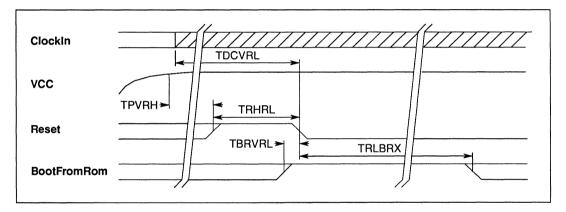


Figure 5.4 Transputer reset timing with Analyse low

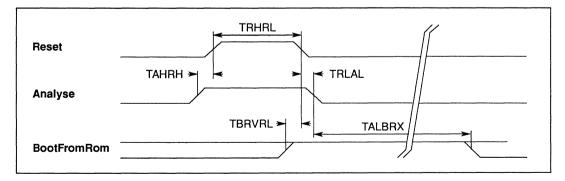


Figure 5.5 Transputer reset and analyse timing

5 System services 353

#### 5.6 Peek and poke

Any location in internal or external memory can be interrogated and altered when the transputer is waiting for a bootstrap from link. If the control byte is 0 then eight more bytes are expected on the same link. The first four byte word is taken as an internal or external memory address at which to poke (write) the second four byte word. If the control byte is 1 the next four bytes are used as the address from which to peek (read) a word of data; the word is sent down the output channel of the same link.

Following such a peek or poke, the transputer returns to its previously held state. Any number of accesses may be made in this way until the control byte is greater than 1, when the transputer will commence reading its bootstrap program. Any link can be used, but addresses and data must be transmitted via the same link as the control byte.

## 5.7 Analyse

If **Analyse** is taken high when the transputer is running, the transputer will halt at the next descheduling point (page 344). From **Analyse** being asserted, the processor will halt within three time slice periods plus the time taken for any high priority process to complete. As much of the transputer status is maintained as is necessary to permit analysis of the halted machine. Processor flags **Error** and **HaltOnError** are not altered at reset, whether **Analyse** is asserted or not. Memory refresh continues.

Input links will continue with outstanding transfers. Output links will not make another access to memory for data but will transmit only those bytes already in the link buffer. Providing there is no delay in link acknowledgement, the links should be inactive within a few microseconds of the transputer halting.

Reset should not be asserted before the transputer has halted and link transfers have ceased. When Reset is taken low whilst Analyse is high, neither the memory configuration sequence nor the block of eight refresh cycles will occur; the previous memory configuration will be used for any external memory accesses. If BootFromRom is high the transputer will bootstrap as soon as Analyse is taken low, otherwise it will await a control byte on any link. If Analyse is taken low without Reset going high the transputer state and operation are undefined. After the end of a valid Analyse sequence the registers have the values given in table 5.3.

Table 5.3 Register values after Analyse

- I MemStart if bootstrapping from a link, or the external memory bootstrap address if bootstrapping from ROM.
- W MemStart if bootstrapping from ROM, or the address of the first free word after the bootstrap program if bootstrapping from link.
- A The value of I when the processor halted.
- B The value of W when the processor halted, together with the priority of the process when the transputer was halted (i.e. the W descriptor).
- C The ID of the bootstrapping link if bootstrapping from link.

## 5.8 Error

The **Error** pin is connected directly to the internal *Error* flag and follows the state of that flag. If **Error** is high it indicates an error in one of the processes caused, for example, by arithmetic overflow, divide by zero, array bounds violation or software setting the flag directly (page 344). Once set, the *Error* flag is only cleared by executing the instruction *testerr*. The error is not cleared by processor reset, in order that analysis can identify any errant transputer (page 353).

A process can be programmed to stop if the *Error* flag is set; it cannot then transmit erroneous data to other processes, but processes which do not require that data can still be scheduled. Eventually all processes which rely, directly or indirectly, on data from the process in error will stop through lack of data.

By setting the *HaltOnError* flag the transputer itself can be programmed to halt if *Error* becomes set. If *Error* becomes set after *HaltOnError* has been set, all processes on that transputer will cease but will not necessarily cause other transputers in a network to halt. Setting *HaltOnError* after *Error* will not cause the transputer to halt; this allows the processor reset and analyse facilities to function with the flags in indeterminate states.

An alternative method of error handling is to have the errant process or transputer cause all transputers to halt. This can be done by applying the **Error** output signal of the errant transputer to the **EventReq** pin of a suitably programmed master transputer. Since the process state is preserved when stopped by an error, the master transputer can then use the analyse function to debug the fault. When using such a circuit, note that the *Error* flag is in an indeterminate state on power up; the circuit and software should be designed with this in mind.

Error checks can be removed completely to optimise the performance of a proven program; any unexpected error then occurring will have an arbitrary undefined effect.

If a high priority process pre-empts a low priority one, status of the *Error* and *HaltOnError* flags is saved for the duration of the high priority process and restored at the conclusion of it. Status of the *Error* flag is transmitted to the high priority process but the *HaltOnError* flag is cleared before the process starts. Either flag can be altered in the process without upsetting the error status of any complex operation being carried out by the pre-empted low priority process.

In the event of a transputer halting because of *HaltOnError*, the links will finish outstanding transfers before shutting down. If **Analyse** is asserted then all inputs continue but outputs will not make another access to memory for data. Memory refresh will continue to take place.

After halting due to the *Error* flag changing from 0 to 1 whilst *HaltOnError* is set, register *I* points two bytes past the instruction which set *Error*. After halting due to the **Analyse** pin being taken high, register *I* points one byte past the instruction being executed. In both cases *I* will be copied to register *A*.

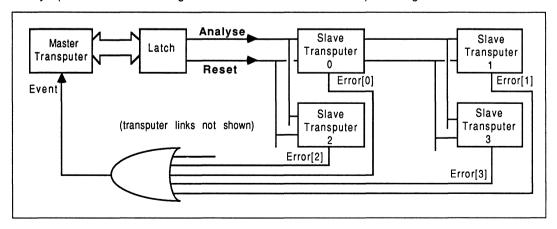


Figure 5.6 Error handling in a multi-transputer system

# 6 Memory

The IMS T414 has 2 Kbytes of fast internal static memory for high rates of data throughput. Each internal memory access takes one processor cycle **ProcClockOut** (page 357). The transputer can also access 4 Gbytes of external memory space. Internal and external memory are part of the same linear address space. Internal RAM can be disabled by holding **DisableIntRAM** high. All internal addresses are then mapped to external RAM. This pin should not be altered after **Reset** has been taken low.

IMS T414 memory is byte addressed, with words aligned on four-byte boundaries. The least significant byte of a word is the lowest addressed byte.

The bits in a byte are numbered 0 to 7, with bit 0 the least significant. The bytes are numbered from 0, with byte 0 the least significant. In general, wherever a value is treated as a number of component values, the components are numbered in order of increasing numerical significance, with the least significant component numbered 0. Where values are stored in memory, the least significant component value is stored at the lowest (most negative) address.

Internal memory starts at the most negative address #80000000 and extends to #800007FF. User memory begins at #80000048; this location is given the name *MemStart*.

The reserved area of internal memory below MemStart is used to implement link and event channels.

Two words of memory are reserved for timer use, *TPtrLoc0* for high priority processes and *TPtrLoc1* for low priority processes. They either indicate the relevant priority timer is not in use or point to the first process on the timer gueue at that priority level.

Values of certain processor registers for the current low priority process are saved in the reserved *IntSaveLoc* locations when a high priority process pre-empts a low priority one.

External memory space starts at #80000800 and extends up through #00000000 to #7FFFFFFF. Memory configuration data and ROM bootstrapping code must be in the most positive address space, starting at #7FFFFF6C and #7FFFFFFE respectively. Address space immediately below this is conventionally used for ROM based code.

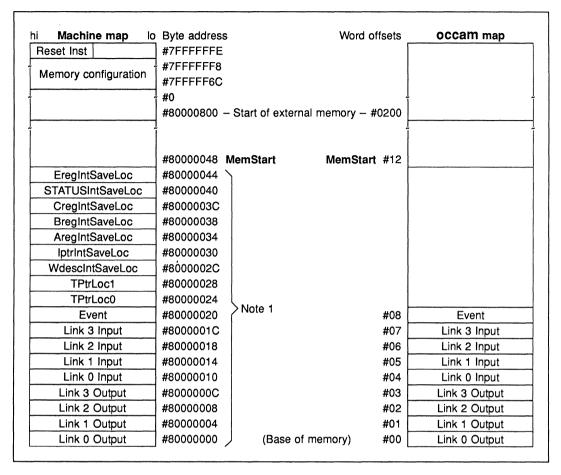


Figure 6.1 IMS T414 memory map

#### Notes

1 These locations are used as auxiliary processor registers and should not be manipulated by the user. Like processor registers, their contents may be useful for implementing debugging tools (Analyse, page 353). For details see Transputer Instruction Set - A Compiler Writers' Guide.

# 7 External memory interface

The External Memory Interface (EMI) allows access to a 32 bit address space, supporting dynamic and static RAM as well as ROM and EPROM. EMI timing can be configured at **Reset** to cater for most memory types and speeds, and a program is supplied with the Transputer Development System to aid in this configuration.

There are 13 internal configurations which can be selected by a single pin connection (page 366). If none are suitable the user can configure the interface to specific requirements, as shown in page 368.

#### 7.1 ProcClockOut

This clock is derived from the internal processor clock, which is in turn derived from **Clockin**. Its period is equal to one internal microcode cycle time, and can be derived from the formula

#### TPCLPCL = TDCLDCL / PLLx

where TPCLPCL is the ProcClockOut Period, TDCLDCL is the ClockIn Period and PLLx is the phase lock loop factor for the relevant speed part, obtained from the ordering details (Ordering section).

The time value **Tm** is used to define the duration of **Tstates** and, hence, the length of external memory cycles; its value is exactly half the period of one **ProcClockOut** cycle (0.5\***TPCLPCL**), regardless of mark/space ratio of **ProcClockOut**.

Edges of the various external memory strobes coincide with rising or falling edges of **ProcClockOut**. It should be noted, however, that there is a skew associated with each coincidence. The value of skew depends on whether coincidence occurs when the **ProcClockOut** edge and strobe edge are both rising, when both are falling or if either is rising when the other is falling. Timing values given in the strobe tables show the best and worst cases. If a more accurate timing relationship is required, the exact **Tstate** timing and strobe edge to **ProcClockOut** relationships should be calculated and the correct skew factors applied from the edge skew timing table 7.4.

#### 7.2 Tstates

The external memory cycle is divided into six **Tstates** with the following functions:

- T1 Address setup time before address valid strobe.
- T2 Address hold time after address valid strobe.
- T3 Read cycle tristate or write cycle data setup.
- T4 Extendable data setup time.
- T5 Read or write data.
- T6 Data hold.

Under normal conditions each **Tstate** may be from one to four periods **Tm** long, the duration being set during memory configuration. The default condition on **Reset** is that all **Tstates** are the maximum four periods **Tm** long to allow external initialisation cycles to read slow ROM.

Period **T4** can be extended indefinitely by adding externally generated wait states.

An external memory cycle is always an even number of periods **Tm** in length and the start of **T1** always coincides with a rising edge of **ProcClockOut**. If the total configured quantity of periods **Tm** is an odd number, one extra period **Tm** will be added at the end of **T6** to force the start of the next **T1** to coincide with a rising edge of **ProcClockOut**. This period is designated **E** in configuration diagrams (figure 7.11).

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTE
TPCLPCL	ProcClockOut period	a-1	а	a+1	ns	1
TPCHPCL	ProcClockOut pulse width high	<b>b</b> -2.5	b	<b>b</b> +2.5	ns	2
TPCLPCH	ProcClockOut pulse width low	i	С		ns	3
Tm	ProcClockOut half cycle	<b>b</b> -0.5	b	<b>b</b> +0.5	ns	2
TPCstab	ProcClockOut stability			4	%	4

Table 7.1 ProcClockOut

#### Notes

- 1 a is TDCLDCL/PLLx.
- 2 b is 0.5\*TPCLPCL (half the processor clock period).
- 3 c is TPCLPCL-TPCHPCL.
- 4 Stability is the variation of cycle periods between two consecutive cycles, measured at corresponding points on the cycles.

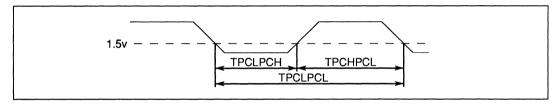


Figure 7.1 IMS T414 ProcClockOut timing

#### 7.3 Internal access

During an internal memory access cycle the external memory interface bus **MemAD2-31** reflects the word address used to access internal RAM, **MemnotWrD0** reflects the read/write operation and **MemnotRfD1** is high; all control strobes are inactive. This is true unless and until a memory refresh cycle or DMA (memory request) activity takes place, when the bus will carry the appropriate external address or data.

The bus activity is not adequate to trace the internal operation of the transputer in full, but may be used for hardware debugging in conjuction with peek and poke (page 353).

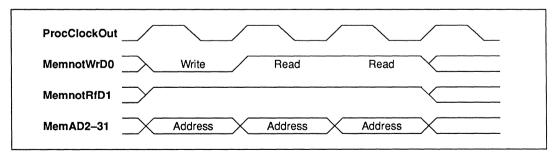


Figure 7.2 IMS T414 bus activity for internal memory cycle

#### 7.4 MemAD2-31

External memory addresses and data are multiplexed on one bus. Only the top 30 bits of address are output on the external memory interface, using pins **MemAD2-31**. They are normally output only during **Tstates T1** and **T2**, and should be latched during this time. Byte addressing is carried out internally by the transputer for read cycles. For write cycles the relevant bytes in memory are addressed by the write strobes **notMemWrB0-3**.

The data bus is 32 bits wide. It uses MemAD2-31 for the top 30 bits and MemnotRfD1 and MemnotWrD0 for the lower two bits. Read cycle data may be set up on the bus at any time after the start of T3, but must be valid when the transputer reads it at the end of T5. Data may be removed any time during T6, but must be off the bus no later than the end of that period.

Write data is placed on the bus at the start of T3 and removed at the end of T6. If T6 is extended to force the next cycle Tmx (page 359) to start on a rising edge of ProcClockOut, data will be valid during this time also.

## 7.5 MemnotWrD0

During T1 and T2 this pin will be low if the cycle is a write cycle, otherwise it will be high. During Tstates T3 to T6 it becomes bit 0 of the data bus. In both cases it follows the general timing of MemAD2-31.

#### 7.6 MemnotRfD1

During T1 and T2, this pin is low if the address on MemAD2-31 is a refresh address, otherwise it is high. During Tstates T3 to T6 it becomes bit 1 of the data bus. In both cases it follows the general timing of MemAD2-31.

## 7.7 notMemRd

For a read cycle the read strobe **notMemRd** is low during **T4** and **T5**. Data is read by the transputer on the rising edge of this strobe, and may be removed immediately afterward. If the strobe duration is insufficient it may be extended by adding extra periods **Tm** to either or both of the **Tstates T4** and **T5**. Further extension may be obtained by inserting wait states at the end of **T4**.

In the read cycle timing diagrams **ProcClockOut** is included as a guide only; it is shown with each **Tstate** configured to one period **Tm**.

#### 7.8 notMemS0-4

To facilitate control of different types of memory and devices, the EMI is provided with five strobe outputs, four of which can be configured by the user. The strobes are conventionally assigned the functions shown in the read and write cycle diagrams, although there is no compulsion to retain these designations.

notMemS0 is a fixed format strobe. Its leading edge is always coincident with the start of T2 and its trailing edge always coincident with the end of T5.

The leading edge of **notMemS1** is always coincident with the start of **T2**, but its duration may be configured to be from zero to 31 periods **Tm**. Regardless of the configured duration, the strobe will terminate no later than the end of **T6**. The strobe is sometimes programmed to extend beyond the normal end of **Tmx**. When wait states are inserted into an EMI cycle the end of **Tmx** is delayed, but the potential active duration of the strobe is not altered. Thus the strobe can be configured to terminate relatively early under certain conditions (page 374). If **notMemS1** is configured to be zero it will never go low.

**notMemS2**, **notMemS3** and **notMemS4** are identical in operation. They all terminate at the end of **T5**, but the start of each can be delayed from one to 31 periods **Tm** beyond the start of **T2**. If the duration of one of these strobes would take it past the end of **T5** it will stay high. This can be used to cause a strobe to become active only when wait states are inserted. If one of these strobes is configured to zero it will never go low. Figure 7.5 shows the effect of **Wait** on strobes in more detail; each division on the scale is one period **Tm**.

Table	7.2 F	Read
-------	-------	------

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTE
TaZdV	Address tristate to data valid	0			ns	
TdVRdH	Data setup before read	20	Ì		ns	
TRdHdX	Data hold after read	0			ns	
TS0LRdL	notMemS0 before start of read	<b>a</b> -2	а	<b>a</b> +2	ns	1 1
TS0HRdH	End of read from end of notMemS0	-1		1	ns	
TRdLRdH	Read period	b		<b>b</b> +6	ns	2

- 1 a is total of T2+T3 where T2, T3 can be from one to four periods Tm each in length.
- 2 b is total of T4+Twait+T5 where T4, T5 can be from one to four periods Tm each in length and Twait may be any number of periods Tm in length.

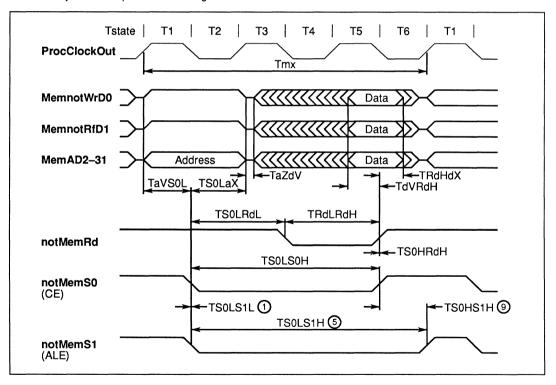


Figure 7.3 IMS T414 external read cycle: static memory

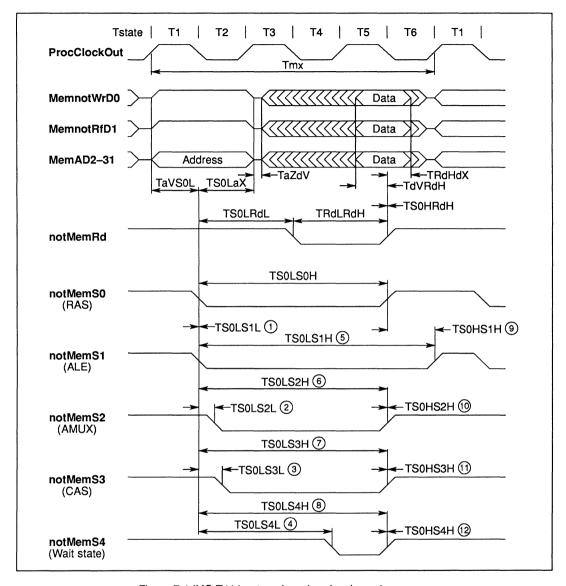


Figure 7.4 IMS T414 external read cycle: dynamic memory

SYMBOL	(h)	PARAMETER	MIN	NOM	MAX	UNITS	NOTE	
TaVS0L		Address setup before notMemS0		а		ns	1	
TS0LaX		Address hold after notMemS0	1	b		ns	2	1
TS0LS0H		notMemS0 pulse width low	С		<b>c</b> +6	ns	3	
TS0LS1L	1	notMemS1 from notMemS0	0		2	ns		
TS0LS1H	5	notMemS1 end from notMemS0	d		<b>d</b> +6	ns	4,6	
TS0HS1H	9	notMemS1 end from notMemS0 end	e-1		e+4	ns	5,6	į
TS0LS2L	2	notMemS2 delayed after notMemS0	f-1		f+4	ns	7	
TS0LS2H	6	notMemS2 end from notMemS0	C+4		<b>c</b> +8	ns	3	
TS0HS2H	10	notMemS2 end from notMemS0 end	0		2	ns		
TS0LS3L	3	notMemS3 delayed after notMemS0	f-1		<b>f</b> +3	ns	7	
TS0LS3H	7	notMemS3 end from notMemS0	C+4		<b>c</b> +8	ns	3	l
TS0HS3H	11	notMemS3 end from notMemS0 end	0		2	ns		
TS0LS4L	4	notMemS4 delayed after notMemS0	f-1		f+2	ns	7	
TS0LS4H	8	notMemS4 end from notMemS0	C+4		<b>c</b> +8	ns	3	ĺ
TS0HS4H	12	notMemS4 end from notMemS0 end	0		2	ns		l
Tmx		Complete external memory cycle		g			8	ĺ

Table 7.3 IMS T414 strobe timing

- 1 a is T1 where T1 can be from one to four periods Tm in length.
- 2 b is T2 where T2 can be from one to four periods Tm in length.
- 3 c is total of T2+T3+T4+Twait+T5 where T2, T3, T4, T5 can be from one to four periods Tm each in length and Twait may be any number of periods Tm in length.
- 4 d can be from zero to 31 periods Tm in length.
- 5 e can be from -27 to +4 periods Tm in length.
- 6 If the configuration would cause the strobe to remain active past the end of **T6** it will go high at the end of **T6**. If the strobe is configured to zero periods **Tm** it will remain high throughout the complete cycle **Tmx**.
- 7 f can be from zero to 31 periods Tm in length. If this length would cause the strobe to remain active past the end of T5 it will go high at the end of T5. If the strobe value is zero periods Tm it will remain low throughout the complete cycle Tmx.
- 8 g is one complete external memory cycle comprising the total of T1+T2+T3+T4+Twait+T5+T6 where T1, T2, T3, T4, T5 can be from one to four periods Tm each in length, T6 can be from one to five periods Tm in length and Twait may be zero or any number of periods Tm in length.



Figure 7.5 IMS T414 effect of wait states on strobes

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTE
TPCHS0H	Strobe rising from ProcClockOut rising	0		3	ns	
TPCLS0H	Strobe rising from ProcClockOut falling	1		4	ns	
TPCHS0L	Strobe falling from ProcClockOut rising	-3		0	ns	
TPCLS0L	Strobe falling from ProcClockOut falling	-1		2	ns	

Table 7.4 Strobe S0 to ProcClockOut skew

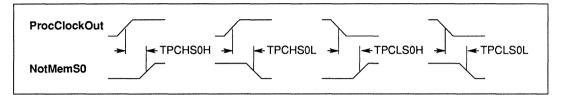


Figure 7.6 IMS T414 skew of notMemS0 to ProcClockOut

#### 7.9 notMemWrB0-3

Because the transputer uses word addressing, four write strobes are provided; one to write each byte of the word. If a particular byte is not to be written, then the corresponding data outputs are tristated. **notMemWrB0** addresses the least significant byte.

The transputer has both early and late write cycle modes. For a late write cycle the relevant write strobes notMemWrB0-3 are low during T4 and T5; for an early write they are also low during T3. Data should be latched into memory on the rising edge of the strobes in both cases, although it is valid until the end of T6. If the strobe duration is insufficient, it may be extended at configuration time by adding extra periods Tm to either or both of Tstates T4 and T5 for both early and late modes. For an early cycle they may also be added to T3. Further extension may be obtained by inserting wait states at the end of T4. If the data hold time is insufficient, extra periods Tm may be added to T6 to extend it.

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTE
TdVWrH	Data setup before write	d			ns	1,5
TWrHdX	Data hold after write	a			ns	1,2
TS0LWrL	notMemS0 before start of early write	<b>b</b> -3		<b>b</b> +2	ns	1,3
	notMemS0 before start of late write	<b>c</b> -3		<b>c</b> +2	ns	1,4
TS0HWrH	End of write from end of notMemS0	-2		2	ns	1
TWrLWrH	Early write pulse width	d		<b>d</b> +6	ns	1,5
	Late write pulse width	e		<b>e</b> +6	ns	1,6

Table 7.5 Write

- 1 Timing is for all write strobes notMemWrB0-3.
- 2 a is T6 where T6 can be from one to five periods Tm in length.
- 3 b is T2 where T2 can be from one to four periods Tm in length.
- 4 c is total of T2+T3 where T2, T3 can be from one to four periods Tm each in length.
- 5 d is total of T3+T4+Twait+T5 where T3, T4, T5 can be from one to four periods Tm each in length and Twait may be zero or any number of periods Tm in length.
- 6 e is total of T4+Twait+T5 where T4, T5 can be from one to four periods Tm each in length and Twait may be zero or any number of periods Tm in length.

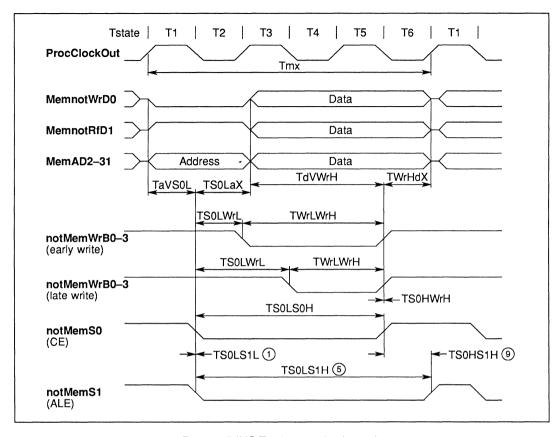


Figure 7.7 IMS T414 external write cycle

In the write cycle timing diagram ProcClockOut is included as a guide only; it is shown with each Tstate configured to one period Tm. The strobe is inactive during internal memory cycles.

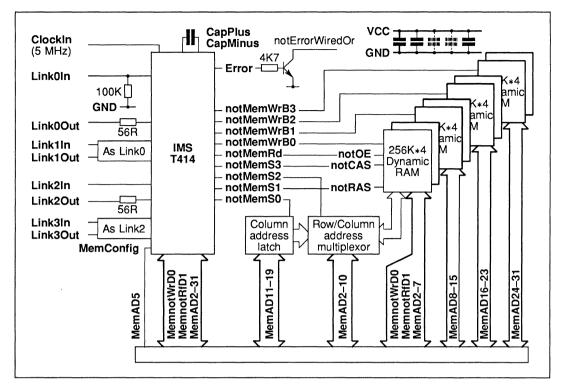


Figure 7.8 IMS T414 dynamic RAM application

## 7.10 MemConfig

**MemConfig** is an input pin used to read configuration data when setting external memory interface (EMI) characteristics. It is read by the processor on two occasions after **Reset** goes low; first to check if one of the preset internal configurations is required, then to determine a possible external configuration.

## 7.10.1 Internal configuration

The internal configuration scan comprises 64 periods TDCLDCL of ClockIn during the internal scan period of 144 ClockIn periods. MemnotWrD0, MemnotRfD1 and MemAD2-32 are all high at the beginning of the scan. Starting with MemnotWrD0, each of these lines goes low successively at intervals of two ClockIn periods and stays low until the end of the scan. If one of these lines is connected to MemConfig the preset internal configuration mode associated with that line will be used as the EMI configuration. The default configuration is that defined in the table for MemAD31; connecting MemConfig to VCC will also produce this default configuration. Note that only 13 of the possible configurations are valid, all others remain at the default configuration.

	Di		on of period			ite	Strobe coefficient							Refresh interval	Cycle time
Pin	T1	T2	тз	T4	<b>T</b> 5	Т6	s1	s2	s3	s4	type	Clockin cycles	Proc cycles		
MemnotWrD0	1	1	1	1	1	1	30	1	3	5	late	72	3		
MemnotRfD1	1	2	1	1	1	2	30	1	2	7	late	72	4		
MemAD2	1	2	1	1	2	3	30	1	2	7	late	72	5		
MemAD3	2	3	1	1	2	3	30	1	3	8	late	72	6		
MemAD4	1	1	1	1	1	1	3	1	2	3	early	72	3		
MemAD5	1	1	2	1	2	1	5	1	2	3	early	72	4		
MemAD6	2	1	2	1	3	1	6	1	2	3	early	72	5		
MemAD7	2	2	2	1	3	2	7	1	3	4	early	72	6		
MemAD8	1	1	1	1	1	1	30	1	2	3	early	†	3		
MemAD9	1	1	2	1	2	1	30	2	5	9	early	†	4		
MemAD10	2	2	2	2	4	2	30	2	3	8	late	72	7		
MemAD11	3	3	3	3	3	3	30	2	4	13	late	72	9		
MemAD31	4	4	4	4	4	4	31	30	30	18	late	72	12		

Table 7.6 IMS T414 internal configuration coding

<sup>†</sup> Provided for static RAM only.

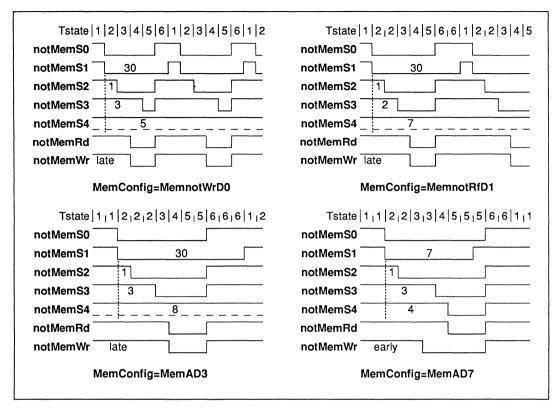


Figure 7.9 IMS T414 internal configuration

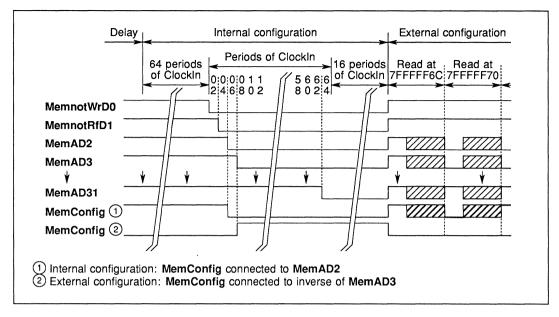


Figure 7.10 IMS T414 internal configuration scan

# 7.10.2 External configuration

If **MemConfig** is held low until **MemnotWrD0** goes low the internal configuration is ignored and an external configuration will be loaded instead. An external configuration scan always follows an internal one, but if an internal configuration occurs any external configuration is ignored.

The external configuration scan comprises 36 successive external read cycles, using the default EMI configuration preset by **MemAD31**. However, instead of data being read on the data bus as for a normal read cycle, only a single bit of data is read on **MemConfig** at each cycle. Addresses put out on the bus for each read cycle are shown in table 7.7, and are designed to address ROM at the top of the memory map. The table shows the data to be held in ROM; data required at the **MemConfig** pin is the inverse of this.

**MemConfig** is typically connected via an inverter to **MemnotWrD0**. Data bit zero of the least significant byte of each ROM word then provides the configuration data stream. By switching **MemConfig** between various data bus lines up to 32 configurations can be stored in ROM, one per bit of the data bus. **MemConfig** can be permanently connected to a data line or to **GND**. Connecting **MemConfig** to **GND** gives all **Tstates** configured to four periods; **notMemS1** pulse of maximum duration; **notMemS2-4** delayed by maximum; refresh interval 72 periods of **ClockIn**; refresh enabled; late write.

The external memory configuration table 7.7 shows the contribution of each memory address to the 13 configuration fields. The lowest 12 words (#7FFFFF6C to #7FFFF98, fields 1 to 6) define the number of extra periods **Tm** to be added to each **Tstate**. If field 2 is 3 then three extra periods will be added to **T2** to extend it to the maximum of four periods.

The next five addresses (field 7) define the duration of **notMemS1** and the following fifteen (fields 8 to 10) define the delays before strobes **notMemS2-4** become active. The five bits allocated to each strobe allow durations of from 0 to 31 periods **Tm**, as described in strobes page 359.

Addresses #7FFFFEC to #7FFFFF4 (fields 11 and 12) define the refresh interval and whether refresh is to be used, whilst the final address (field 13) supplies a high bit to **MemConfig** if a late write cycle is required.

The columns to the right of the coding table show the values of each configuration bit for the four sample

external configuration diagrams. Note the inclusion of period **E** at the end of **T6** in some diagrams. This is inserted to bring the start of the next **Tstate T1** to coincide with a rising edge of **ProcClockOut** (page 357).

Wait states **W** have been added to show the effect of them on strobe timing; they are not part of a configuration. In each case which includes wait states, two wait periods are defined. This shows that if a wait state would cause the start of **T5** to coincide with a falling edge of **ProcClockOut**, another period **Tm** is generated by the EMI to force it to coincide with a rising edge of **ProcClockOut**. This coincidence is only necessary if wait states are added, otherwise coincidence with a falling edge is permitted. Any configuration memory access is only permitted to be extended using wait, up to a total of 14 **ClockIn** periods.

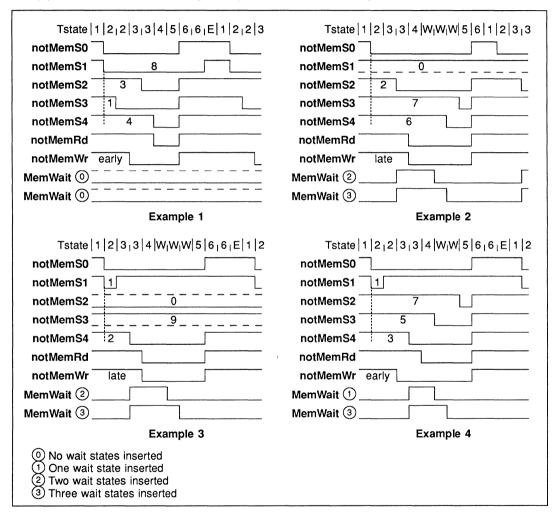


Figure 7.11 IMS T414 external configuration

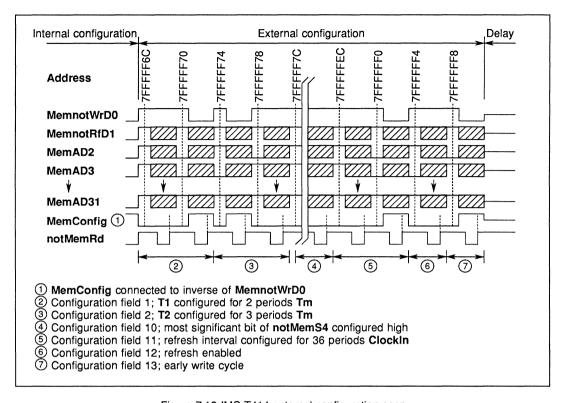


Figure 7.12 IMS T414 external configuration scan

Table 7.7 IMS T414 external configuration coding

Scan	MemAD			Example diagram			ram
cycle	address	Field	Function	1	2	3	4
1	7FFFFF6C	1.	T1 least significant bit	0	0	0	0
2	7FFFFF70	1	T1 most significant bit	0	0	0	0
3	7FFFFF74	2	T2 least significant bit	1	0	0	1
4	7FFFFF78	2	T2 most significant bit	0	0	0	0
5 6 7	7FFFFF7C	3	T3 least significant bit	1	1	1	1
6	7FFFFF80	3	T3 most significant bit	0	0	0	0
7	7FFFFF84	4	T4 least significant bit	0	0	0	0
8	7FFFFF88	4	T4 most significant bit	0	0	0	0
9	7FFFFF8C	5	T5 least significant bit	0	0	0	0
10	7FFFFF90	5	T5 most significant bit	0	0	0	0
11	7FFFFF94	6	T6 least significant bit	1	0	1	1
12	7FFFFF98	6	T6 most significant bit	0	0	0	0
13	7FFFFF9C	7	notMemS1 least significant bit	0	0	1	1
14	7FFFFFA0	7		0	0	0	0
15	7FFFFFA4	7	₩ ₩	0	0	0	0
16	7FFFFFA8	7		1	0	0	0
17	7FFFFFAC	7	notMemS1 most significant bit	0	0	0	0
18	7FFFFB0	8	notMemS2 least significant bit	1	0	0	1
19	7FFFFB4	8		1	1	0	1
20	7FFFFFB8	8	₩ ₩	0	0	0	1
21	7FFFFBC	8	_	0	0	0	0
22	7FFFFC0	8	notMemS2 most significant bit	0	0	0	0
23	7FFFFFC4	9	notMemS3 least significant bit	1	1	1	1
24	7FFFFFC8	9		0	1	0	0
25	7FFFFFCC	9		0	1	0	1
26	7FFFFFD0	9		0	0	1	0
27	7FFFFFD4	9	notMemS3 most significant bit	0	0	0	0
28	7FFFFFD8	10	notMemS4 least significant bit	0	0	0	1
29	7FFFFDC	10		0	1	1	1
30	7FFFFE0	10	₩ ₩	1	1	0	0
31	7FFFFE4	10		0	0	0	0
32	7FFFFE8	10	notMemS4 most significant bit	0	0	0	0
33	7FFFFEC	11	Refresh Interval least significant bit	-	-	-	-
34	7FFFFFF0	11	Refresh Interval most significant bit	-	-	-	-
35	7FFFFFF4	12	Refresh Enable	-	-	-	-
36	7FFFFFF8	13	Late Write	0	1	1	0

Refresh interval	Interval in μs	Field 11 encoding	Complete cycle (mS)		
18	3.6	00	0.922		
36	7.2	01	1.843		
54	10.8	10	2.765		
72	14.4	11	3.686		

Table 7.8 IMS T414 memory refresh configuration coding

Refresh intervals are in periods of ClockIn and ClockIn frequency is 5 MHz:

Interval = 18 \* 200 = 3600 ns

Refresh interval is between successive incremental refresh addresses. Complete cycles are shown for 256 row DRAMS.

Table 7.9 Memory configuration

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTE
TMCVRdH	Memory configuration data setup	30			ns	
TRdHMCX	Memory configuration data hold	0			ns	
TS0LRdH	notMemS0 to configuration data read	а		<b>a</b> +6	ns	1

## Notes

1 a is 16 periods Tm.

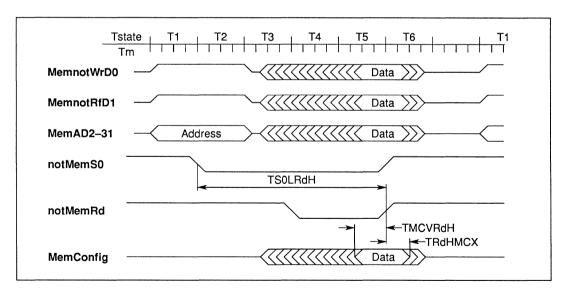


Figure 7.13 IMS T414 external configuration read cycle timing

## 7.11 notMemRf

The IMS T414 can be operated with memory refresh enabled or disabled. The selection is made during memory configuration, when the refresh interval is also determined. Refresh cycles do not interrupt internal memory accesses, although the internal addresses cannot be reflected on the external bus during refresh.

When refresh is disabled no refresh cycles occur. During the post-Reset period eight dummy refresh cycles will occur with the appropriate timing but with no bus or strobe activity.

A refresh cycle uses the same basic external memory timing as a normal external memory cycle, except that it starts two periods **Tm** before the start of **T1**. If a refresh cycle is due during an external memory access, it will be delayed until the end of that external cycle. Two extra periods **Tm** (periods **R** in the diagram) will then be inserted between the end of **T6** of the external memory cycle and the start of **T1** of the refresh cycle itself. The refresh address and various external strobes become active approximately one period **Tm** before **T1**. Bus signals are active until the end of **T2**, whilst **notMemRf** remains active until the end of **T6**.

For a refresh cycle, MemnotRfD1 goes low before notMemRf goes low and MemnotWrD0 goes high with the same timing as MemnotRfD1. All the address lines share the same timing, but only MemAD2-11 give the refresh address. MemAD12-30 stay high during the address period, whilst MemAD31 remains low. Refresh cycles generate strobes notMemS0-4 with timing as for a normal external cycle, but notMemRd and notMemWrB0-3 remain high. MemWait operates normally during refresh cycles.

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTE
TRfLRfH	Refresh pulse width low	а		<b>a</b> +6	ns	1
TRaVS0L	Refresh address setup before notMemS0	T1-5			ns	
TRfLS0L	Refresh indicator setup before notMemS0	<b>b</b> -5		<b>b</b> +5	ns	2

Table 7.10 Memory refresh

#### Notes

- 1 a is total Tmx+Tm.
- 2 b is total T1+Tm where T1 can be from one to four periods Tm in length.

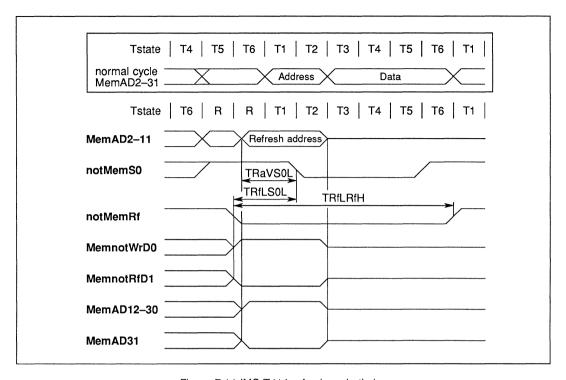


Figure 7.14 IMS T414 refresh cycle timing

# 7.12 MemWait

Taking MemWait high with the timing shown will extend the duration of T4. MemWait is sampled relative to the falling edge of ProcClockOut during a T3 period, and should not change state in this region. By convention, notMemS4 is used to synchronize wait state insertion. If this or another strobe is used, its delay should be such as to take the strobe low an even number of periods Tm after the start of T1, to coincide with a rising edge of ProcClockOut.

MemWait may be kept high indefinitely, although if dynamic memory refresh is used it should not be kept high long enough to interfere with refresh timing. MemWait operates normally during all cycles, including refresh and configuration cycles. It does not affect internal memory access in any way.

If the start of **T5** would coincide with a falling edge of **ProcClockOut** an extra wait period **Tm** (**EW**) is generated by the EMI to force coincidence with a rising edge. Rising edge coincidence is only forced if wait states are added, otherwise coincidence with a falling edge is permitted.

Table 7.11 Memory wait

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTE
TPCHWtH	Wait setup	0.5 <b>Tm</b> +3			ns	1,2
TPCHWtL	Wait hold	0.5 <b>Tm</b> +3			ns	1,2
TWtLWtH	Delay before re-assertion of Wait	2 <b>Tm</b>			ns	ĺ

## Notes

- 1 ProcClockOut load should not exceed 50pf.
- 2 If wait period exceeds refresh interval, refresh cycles will be lost.

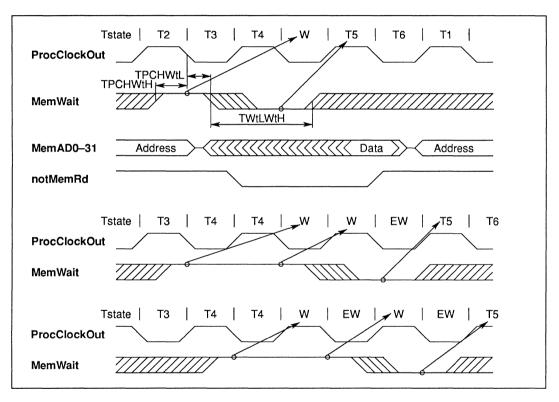


Figure 7.15 IMS T414 memory wait timing

# 7.13 MemReg, MemGranted

Direct memory access (DMA) can be requested at any time by taking the asynchronous **MemReq** input high. The transputer samples **MemReq** during the final period **Tm** of **T6** of both refresh and external memory cycles. To guarantee taking over the bus immediately following either, **MemReq** must be set up at least two periods **Tm** before the end of **T6**. In the absence of an external memory cycle, **MemReq** is sampled during every low period of **ProcClockOut**. The address bus is tristated two periods **Tm** after the **ProcClockOut** rising edge which follows the sample. **MemGranted** is asserted one period **Tm** after that.

Removal of **MemReq** is sampled during each low period of **ProcClockOut** and **MemGranted** is removed synchronously with the next falling edge of **ProcClockOut**. If accurate timing of DMA is required, **MemReq** should be set low coincident with a falling edge of **ProcClockOut**. Further external bus activity, either refresh, external cycles or reflection of internal cycles, will commence at the next rising edge of **ProcClockOut**.

Strobes are left in their inactive states during DMA. DMA cannot interrupt a refresh or external memory cycle, and outstanding refresh cycles will occur before the bus is released to DMA. DMA does not interfere with internal memory cycles in any way, although a program running in internal memory would have to wait for the end of DMA before accessing external memory. DMA cannot access internal memory. If DMA extends longer than one refresh interval (Memory Refresh Configuration Coding, table 7.8), the DMA user becomes responsible for refresh. DMA may also inhibit an internally running program from accessing external memory.

DMA allows a bootstrap program to be loaded into external RAM ready for execution after reset. If **MemReq** is held high throughout reset, **MemGranted** will be asserted before the bootstrap sequence begins. **MemReq** must be high at least one period **TDCLDCL** of **ClockIn** before **Reset**. The circuit should be designed to ensure correct operation if **Reset** could interrupt a normal DMA cycle.

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTE
TMRHMGH	Memory request response time	4		7	Tm	1
TMRLMGL	Memory request end response time	2		5	Tm	
TADZMGH	Bus tristate before memory granted		1		Tm	
TMGLADV	Bus active after end of memory granted		1		Tm	

Table 7.12 Memory request

## Notes

1 These values assume no external memory cycle is in progress. If an external cycle is active, maximum time could be (1 EMI cycle Tmx)+(1 refresh cycle TRfLRfH)+(6 periods Tm).

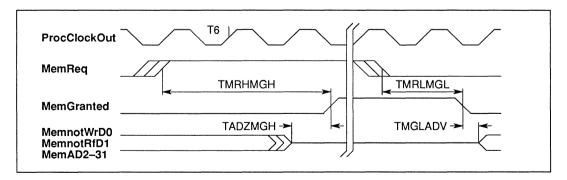


Figure 7.16 IMS T414 memory request timing

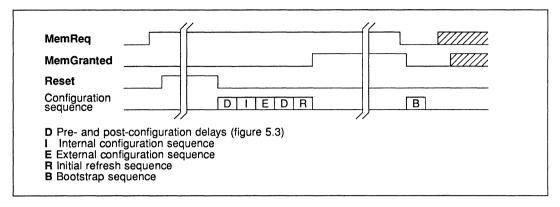


Figure 7.17 IMS T414 DMA sequence at reset

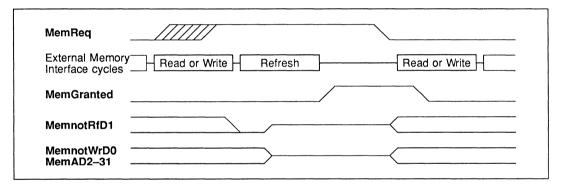


Figure 7.18 IMS T414 operation of MemReq, MemGranted with external, refresh memory cycles

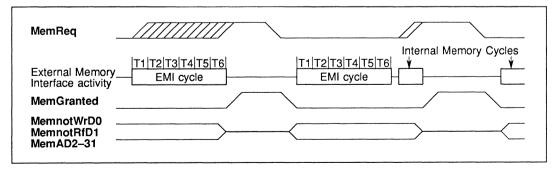


Figure 7.19 IMS T414 operation of MemReg, MemGranted with external, internal memory cycles

# 8 Events

**EventReq** and **EventAck** provide an asynchronous handshake interface between an external event and an internal process. When an external event takes **EventReq** high the external event channel (additional to the external link channels) is made ready to communicate with a process. When both the event channel and the process are ready the processor takes **EventAck** high and the process, if waiting, is scheduled. **EventAck** is removed after **EventReq** goes low.

Only one process may use the event channel at any given time. If no process requires an event to occur **EventAck** will never be taken high. Although **EventReq** triggers the channel on a transition from low to high, it must not be removed before **EventAck** is high. **EventReq** should be low during **Reset**; if not it will be ignored until it has gone low and returned high. **EventAck** is taken low when **Reset** occurs.

If the process is a high priority one and no other high priority process is running, the latency is as described on page 341. Setting a high priority task to wait for an event input allows the user to interrupt a transputer program running at low priority. The time taken from asserting **EventReq** to the execution of the microcode interrupt handler in the CPU is four cycles. The following functions take place during the four cycles:

- Cycle 1 Sample EventReq at pad on the rising edge of ProcClockOut and synchronise.
- Cycle 2 Edge detect the synchronised EventReq and form the interrupt request.
- Cycle 3 Sample interrupt vector for microcode ROM in the CPU.
- **Cycle 4** Execute the interrupt routine for Event rather than the next instruction.

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTE
TVHKH	Event request response	0			ns	
TKHVL	Event request hold	0	Ì		ns	
TVLKL	Delay before removal of event acknowledge	0		а	ns	1
TKLVH	Delay before re-assertion of event request	0	<b>\</b>		ns	

Table 8.1 Event

## Notes

1 a is 3 processor cycles TPCLPCL.

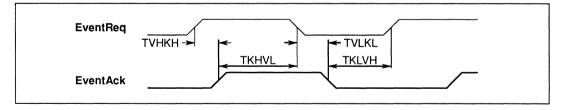


Figure 8.1 IMS T414 event timing

## 9 Links

Four identical INMOS bi-directional serial links provide synchronized communication between processors and with the outside world. Each link comprises an input channel and output channel. A link between two transputers is implemented by connecting a link interface on one transputer to a link interface on the other transputer. Every byte of data sent on a link is acknowledged on the input of the same link, thus each signal line carries both data and control information.

The quiescent state of a link output is low. Each data byte is transmitted as a high start bit followed by a one bit followed by eight data bits followed by a low stop bit. The least significant bit of data is transmitted first. After transmitting a data byte the sender waits for the acknowledge, which consists of a high start bit followed by a zero bit. The acknowledge signifies both that a process was able to receive the acknowledged data byte and that the receiving link is able to receive another byte. The sending link reschedules the sending process only after the acknowledge for the final byte of the message has been received.

The IMS T414 links support the standard INMOS communication speed of 10 Mbits/sec. In addition they can be used at 5 or 20 Mbits/sec. Links are not synchronised with **ClockIn** or **ProcClockOut** and are insensitive to their phases. Thus links from independently clocked systems may communicate, providing only that the clocks are nominally identical and within specification.

Links are TTL compatible and intended to be used in electrically quiet environments, between devices on a single printed circuit board or between two boards via a backplane. Direct connection may be made between devices separated by a distance of less than 300 millimetres. For longer distances a matched 100 ohm transmission line should be used with series matching resistors **RM**. When this is done the line delay should be less than 0.4 bit time to ensure that the reflection returns before the next data bit is sent.

Buffers may be used for very long transmissions. If so, their overall propagation delay should be stable within the skew tolerance of the link, although the absolute value of the delay is immaterial.

Link speeds can be set by LinkSpecial, Link0Special and Link123Special. The link 0 speed can be set independently. Table 9.1 shows uni-directional and bi-directional data rates in Kbytes/sec for each link speed; LinknSpecial is to be read as Link0Special when selecting link 0 speed and as Link123Special for the others. Data rates are quoted for a transputer using internal memory, and will be affected by a factor depending on the number of external memory accesses and the length of the external memory cycle.

Link	Linkn		Kbyte	s/sec
Special	Special	Mbits/sec	Uni	Bi
0	0	10	400	800
0	1	5	200	400
1	0	10	400	800
1	1	20	800	1600

Table 9.1 Speed Settings for Transputer Links

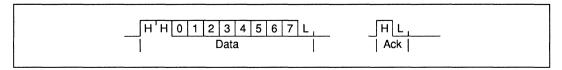


Figure 9.1 IMS T414 link data and acknowledge packets

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTE
TJQr	LinkOut rise time			20	ns	1
TJQf	LinkOut fall time		1	10	ns	1
TJDr	LinkIn rise time			20	ns .	1
TJDf	LinkIn fall time			20	ns	1
TJQJD	Buffered edge delay	0	ĺ		ns	
TJBskew	Variation in TJQJD 20 Mb	its/s		3	ns	2
	10 Mb	its/s		10	ns	2
	5 Mb	its/s		30	ns	2
CLIZ	LinkIn capacitance @ f=1	MHz	l	7	pF	1
CLL	LinkOut load capacitance			50	pF	
RM	Series resistor for 100Ω transmission lin	ne	56		ohms	

Table 9.2 Link

## Notes

- 1 These paramters are sampled, but are not 100% tested.
- 2 This is the variation in the total delay through buffers, transmission lines, differential receivers etc., caused by such things as short term variation in supply voltages and differences in delays for rising and falling edges.

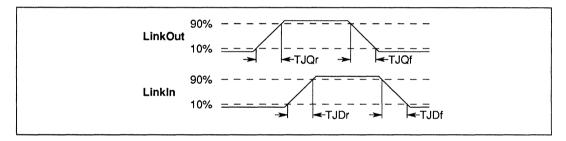


Figure 9.2 IMS T414 link timing

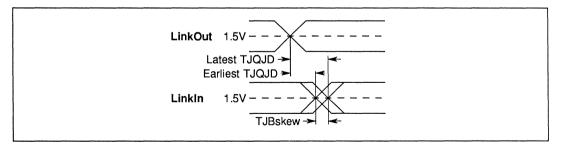


Figure 9.3 IMS T414 buffered link timing

Links 381

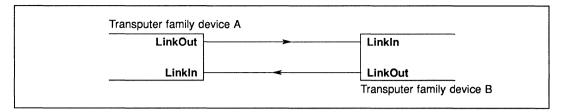


Figure 9.4 IMS T414 Links directly connected

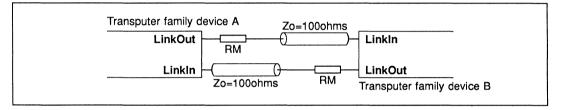


Figure 9.5 IMS T414 Links connected by transmission line

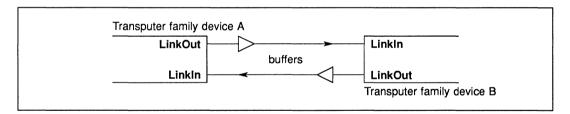


Figure 9.6 IMS T414 Links connected by buffers

# 10 Electrical specifications

## 10.1 DC electrical characteristics

Table 10.1 Absolute maximum ratings

SYMBOL	FARAMETER	MIN	MAX	UNITS	NOTE
VCC	DC supply voltage	0	7.0	٧	1,2,3
VI, VO	Voltage on input and output pins	-0.5	VCC+0.5	٧	1,2,3
11	Input current		±25	mA	4
OSCT	Output short circuit time (one pin)		1	S	2
TS	Storage temperature	-65	150	°C	2
TA	Ambient temperature under bias	-55	125	°C	2
PDmax	Maximum allowable dissipation		2	W	

#### Notes

- 1 All voltages are with respect to GND.
- 2 This is a stress rating only and functional operation of the device at these or any other conditions beyond those indicated in the operating sections of this specification is not implied. Stresses greater than those listed may cause permanent damage to the device. Exposure to absolute maximum rating conditions for extended periods may affect reliability.
- 3 This device contains circuitry to protect the inputs against damage caused by high static voltages or electrical fields. However, it is advised that normal precautions be taken to avoid application of any voltage higher than the absolute maximum rated voltages to this high impedance circuit. Unused inputs should be tied to an appropriate logic level such as VCC or GND.
- 4 The input current applies to any input or output pin and applies when the voltage on the pin is between GND and VCC.

SYMBOL PARAMETER MIN MAX UNITS NOTE VCC 4.75 5.25 DC supply voltage ٧ 1 VI, VO Input or output voltage VCC ٧ 1,2 0 рF CL Load capacitance on any pin 60 TΑ Operating temperature range 0 70 ĊС 3

Table 10.2 Operating conditions

## Notes

- 1 All voltages are with respect to GND.
- 2 Excursions beyond the supplies are permitted but not recommended; see DC characteristics.
- 3 Air flow rate 400 linear ft/min transverse air flow.

SYMBOL	PARAME	TER	MIN	MAX	UNITS	NOTE
VIH	High level input voltage		2.0	VCC+0.5	V	1,2
VIL	Low level input voltage		-0.5	0.8	V	1,2
11	Input current	@ GND <vi<vcc< td=""><td></td><td>±10</td><td>μA</td><td>1,2,7</td></vi<vcc<>		±10	μA	1,2,7
	·	•		±50	$\mu$ A	1,2,8
VOH	Output high voltage	@ IOH=2mA	VCC-1		ĺ۷	1,2
VOL	Output low voltage	@ IOL=4mA		0.4	V	1,2
IOS	Output short circuit current	@ GND <vo<vcc< td=""><td>36</td><td>65</td><td>mA</td><td>1,2,3,6</td></vo<vcc<>	36	65	mA	1,2,3,6
			65	100	mA	1,2,4,6
IOZ	Tristate output current	@ GND <vo<vcc< td=""><td></td><td>±10</td><td>μΑ</td><td>1,2</td></vo<vcc<>		±10	μΑ	1,2
PD	Power dissipation			900	mW	2,5
CIN	Input capacitance	@ f=1MHz		7	pF	6
COZ	Output capacitance	@ f=1MHz		10	pF	6

Table 10.3 DC characteristics

## Notes

- 1 All voltages are with respect to GND.
- 2 Parameters for IMS T414-S measured at 4.75V<VCC<5.25V and 0°C<TA<70°C. Input clock frequency = 5 MHz.</p>
- 3 Current sourced from non-link outputs.
- 4 Current sourced from link outputs.
- 5 Power dissipation varies with output loading and program execution.
- 6 This parameter is sampled and not 100% tested.
- 7 For inputs other than those in Note 8.
- 8 For MemReq, MemWait, MemConfig, Analyse, Reset, Clockin, EventReq, LinkIn0-3, LinkSpecial, LinkOSpecial, Link123Special, BootFromRom, HoldToGND.

# 10.2 Equivalent circuits

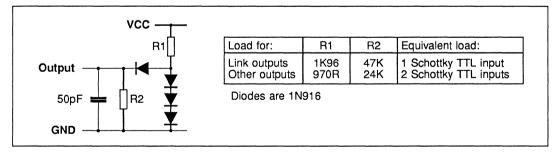


Figure 10.1 Load circuit for AC measurements

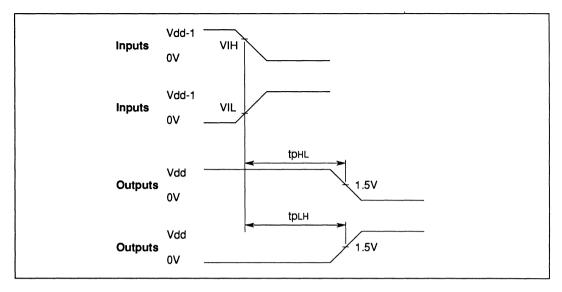


Figure 10.2 AC measurements timing waveforms

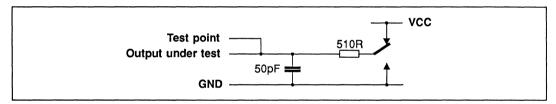


Figure 10.3 Tristate load circuit for AC measurements

# 10.3 AC timing characteristics

Table 10.4 Input, output edges

SYMBOL	PARAMETER	MIN	MAX	UNITS	NOTE
TDr	Input rising edges	2	20	ns	1,2
TDf	Input falling edges	2	20	ns	1,2
TQr	Output rising edges		25	ns	1
TQf	Output falling edges		15	ns	1
TS0LaHZ	Address high to tristate	а	<b>a</b> +6	ns	3
TS0LaLZ	Address low to tristate	а	<b>a</b> +6	ns	3

## Notes

- 1 Non-link pins; see section on links.
- 2 All inputs except Clockin; see section on Clockin.
- 3 a is T2 where T2 can be from one to four periods Tm in length.
  Address lines include MemnotWrD0, MemnotRfD1, MemAD2-31.

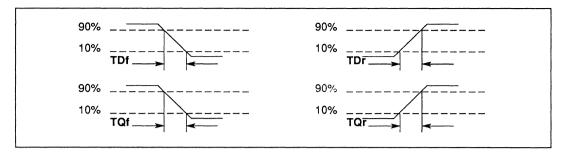


Figure 10.4 IMS T414 input and output edge timing

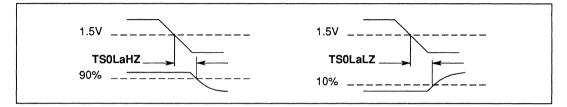
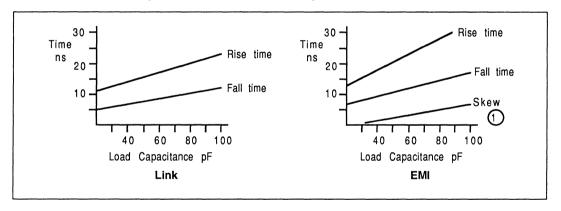


Figure 10.5 IMS T414 tristate timing relative to notMemS0



Notes

Figure 10.6 Typical rise/fall times

1 Skew is measured between **notMemS0** with a standard load (2 Schottky TTL inputs and 30pF) and **notMemS0** with a load of 2 Schottky TTL inputs and varying capacitance.

# 10.4 Power rating

Internal power dissipation  $P_{INT}$  of transputer and peripheral chips depends on VCC, as shown in figure 10.7.  $P_{INT}$  is substantially independent of temperature.

Total power dissipation  $P_D$  of the chip is

$$P_D = P_{INT} + P_{IO}$$

where  $P_{IO}$  is the power dissipation in the input and output pins; this is application dependent.

Internal working temperature  $T_J$  of the chip is

$$T_J = T_A + \theta J_A * P_D$$

where  $T_A$  is the external ambient temperature in °C and  $\theta J_A$  is the junction-to-ambient thermal resistance in °C/W.  $\theta J_A$  for each package is given in the Packaging Specifications section.

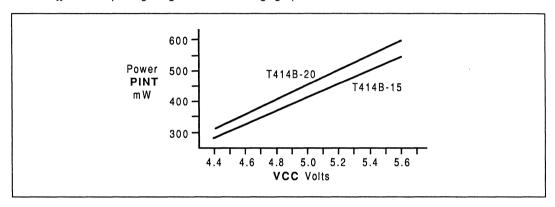


Figure 10.7 IMS T414 internal power dissipation vs VCC

# 11 Performance

The performance of the transputer is measured in terms of the number of bytes required for the program, and the number of (internal) processor cycles required to execute the program. The figures here relate to OCCAM programs. For the same function, other languages should achieve approximately the same performance as OCCAM.

## 11.1 Performance overview

These figures are averages obtained from detailed simulation, and should be used only as an initial guide; they assume operands are of type **INT**. The abbreviations in table 11.1 are used to represent the quantities indicated. In the replicator section of the table, figures in braces {} are not necessary if the number of replications is a compile time constant. To estimate performance, add together the time for the variable references and the time for the operation.

Table 11.1 Key to performance table

np	number of component processes
ne	number of processes earlier in queue
r	1 if INT parameter or array parameter, 0 if not
ts	number of table entries (table size)
w	width of constant in nibbles
р	number of places to shift
Eg	expression used in a guard
Et	timer expression used in a guard
Tb	most significant bit set of multiplier ((-1) if the multiplier is 0)
Tbp	most significant bit set in a positive multiplier when counting from zero ((-1) if the multiplier is 0)
Tbc	most significant bit set in the two's complement of a negative multiplier
nsp	Number of scalar parameters in a procedure
nap	Number of array parameters in a procedure

Table 11.2 Performance

	Size (bytes)	Time (cycles)
Names		
variables		
in expression	1.1+r	2.1+2(r)
assigned to or input to	1.1+r	1.1+( <b>r</b> )
in PROC or FUNCTION call,		al al 4-3
corresponding to an INT parameter channels	1.1+r 1.1	1.1+( <b>r</b> ) 2.1
	1.1	2.1
Array Variables (for single dimension arrays)		
constant subscript	0	0
variable subscript expression subscript	5.3 5.3	7.3 7.3
·	ე.ა	7.3
Declarations	0.4	0.4
CHAN OF protocol [size]CHAN OF protocol	3.1 9.4	3.1 2.2 + 20.2* <b>size</b>
PROC	body+2	2.2 + 20.2* <b>S12e</b> 0
	000y+2	U
Primitives	0	0
assignment input	4	26.5
output	1	26.5
STOP	2	25
SKIP	0	0
Arithmetic operators		
+ -	1	1
*	2	39
/	2	40
REM	2	38
<b>&gt;&gt;                                   </b>	2	3+ <b>p</b>
Modulo Arithmetic operators		
PLUS	2	2
MINUS	1	1
TIMES (fast multiply)	1	4+Tb
Boolean operators		_
OR	4	8 2
AND NOT	l	2
Comparison operators		
= constant	0 2	1 3
= variable <> constant	1	3 3
<> variable	3	5
> <	1	2
>= <=	2	4
Bit operators		
/\ \/ × ~	2	2
Expressions		
constant in expression	w	w
check if error	4	6

11 Performance 389

Table 11.3 Performance

	Size (bytes)	Time (cycles)
Timers		
timer input	2	3
timer AFTER		
if past time	2	4
with empty timer queue	2 2 2	31
non-empty timer queue	2	38+ <b>ne</b> ∗9
ALT (timer)		
with empty timer queue	6	52
non-empty timer queue	6	59+ <b>ne</b> ∗9
timer alt guard	8+2 <b>Eg</b> +2 <b>Et</b>	34+2 <b>Eg</b> +2 <b>Et</b>
Constructs		
SEQ	0	0
IF	1.3	1.4
if guard	3	4.3
ALT (non timer)	6	26
alt channel guard	10.2+2 <b>Eg</b>	20+2 <b>Eg</b>
skip alt guard	8+2 <b>Eg</b>	10+2 <b>Eg</b>
PAR	11.5+( <b>np</b> -1)∗7.5	19.5+( <b>np</b> -1)*30.5
WHILE	4	12
Procedure or function call		
	3.5+(nsp-2)*1.1	16.5+(nsp-2)*1.1
	+nap∗2.3	+nap∗2.3
Replicators	1	
replicated SEQ	7.3{+5.1}	(-3.8)+15.1*count{+7.1}
replicated IF	12.3(+5.1)	(-2.6)+19.4*count{+7.1}
replicated <b>ALT</b>	24.8{+10.2}	25.4+33.4*count{+14.2}
replicated timer <b>ALT</b>	24.8{+10.2}	62.4+33.4*count(+14.2)
replicated PAR	39.1{+5.1}	(-6.4)+70.9*count{+7.1}

## 11.2 Fast multiply, TIMES

The IMS T414 has a fast integer multiplication instruction *product*. The time taken for a fast multiply is 4+**Tb**. The time taken for a multiplication by zero is 3 cycles. For example, if the multiplier is 1 the time taken is 4 cycles, if the multiplier is -1 (all bits set) the time taken is 35 cycles.

Implementations of high level languages on the transputer may take advantage of this instruction. For example, the OCCAM modulo arithmetic operator **TIMES** is implemented by the instruction and the right-hand operand is treated as the multiplier. The fast multiplication instruction is also used in high level language implementations for the multiplication implicit in multi-dimensional array access.

# 11.3 Arithmetic

A set of functions are provided within the development system to support the efficient implementation of multiple length integer arithmetic and floating point arithmetic. In table 11.4 **n** gives the number of places shifted and all arguments and results are assumed to be local. Full details of these functions are provided in the OCCAM reference manual, supplied as part of the development system and available as a separate publication.

When calculating the execution time of the predefined maths functions, no time needs to be added for calling overhead. These functions are compiled directly into special purpose instructions which are designed to support the efficient implementation of multiple length integer arithmetic and floating point arithmetic.

Function		Cycles	+ cycles for parameter access †
LONGADD		2	7
LONGSUM		3	8
LONGSUB		2	7
LONGDIFF		3	8
LONGPROD		34	8
LONGDIV		36	8
SHIFTRIGHT	(n<32)	4+n	8
	( <b>n</b> >=32)	<b>n</b> -27	
SHIFTLEFT	(n<32)	4+n	8
	( <b>n</b> >=32)	<b>n</b> -27	
NORMALISE	(n<32)	<b>n</b> +6	7
	( <b>n</b> >=32)	n-25	
	( <b>n</b> =64)	4	
ASHIFTRIGHT		SHIFTRIGHT+2	5
ASHIFTLEFT		SHIFTLEFT+4	5
ROTATERIGHT		SHIFTRIGHT	7
ROTATELEFT		SHIFTLEFT	7
FRACMUL		LONGPROD+4	5

Table 11.4 Arithmetic performance

# 11.4 Floating point operations

Floating point operations for the IMS T414 are provided by a run-time package. This requires approximately 400 bytes of memory for the single length arithmetic operations, and 2500 bytes for the double length arithmetic operations. Table 11.5 summarizes the estimated performance of the package.

	(191	Processo	or cycles
		IMS	T414
		Typical	Worst
REAL32	+ -	230	300
	*	200	240
	/	245	280
	< > = >= <= <>	60	60
REAL 64	+ -	565	700
-	*	760	940
	/	1115	1420
	< > = >= <= <>	60	60

Table 11.5 IMS T414 floating point operations performance

<sup>†</sup> Assuming local variables.

11 Performance 391

## 11.5 Effect of external memory

Extra processor cycles may be needed when program and/or data are held in external memory, depending both on the operation being performed, and on the speed of the external memory. After a processor cycle which initiates a write to memory, the processor continues execution at full speed until at least the next memory access.

Whilst a reasonable estimate may be made of the effect of external memory, the actual performance will depend upon the exact nature of the given sequence of operations.

External memory is characterized by the number of extra processor cycles per external memory cycle, denoted as **e**. For the IMS T414, with the fastest external memory the value of **e** is 2; a typical value for a large external memory is 5.

If program is stored in external memory, and **e** has the value 2 or 3, then no extra cycles need be estimated for linear code sequences. For larger values of **e**, the number of extra cycles required for linear code sequences may be estimated at (**e**-3)/4. A transfer of control may be estimated as requiring **e**+3 cycles.

These estimates may be refined for various constructs. In table 11.6 n denotes the number of components in a construct. In the case of **IF**, the n'th conditional is the first to evaluate to **TRUE**, and the costs include the costs of the conditionals tested. The number of bytes in an array assignment or communication is denoted by **b**.

	IMS T	414
	Program off chip	Data off chip
Boolean expressions	<b>e</b> -2	0
IF	3 <b>en</b> -8	en
Replicated IF	(6 <b>e</b> -4) <b>n</b> +7	(5 <b>e</b> -2) <b>n</b> +8
Replicated SEQ	(3 <b>e</b> -3) <b>n</b> +2	(4 <b>e</b> -2)n
PAR	(3 <b>e</b> -1) <b>n</b> +8	3en+4
Replicated PAR	(10 <b>e</b> -8) <b>n</b> +8	16en-12
ALT	(2 <b>e</b> -4) <b>n</b> +6 <b>e</b>	(2e-2)n+10e-8
Array assignment and communication in one transputer	0	max (2e, e(b/2))

Table 11.6 External memory performance

For the IMS T414 the effective rate of INMOS links is slowed down on output from external memory by e cycles per word output, and on input to external memory at 10 Mbits/sec by e-6 cycles per word if e≥6.

The following simulation results illustrate the effect of storing program and/or data in external memory. The results are normalized to 1 for both program and data on chip. The first program (Sieve of Erastosthenes) is an extreme case as it is dominated by small, data access intensive loops; it contains no concurrency, communication, or even multiplication or division. The second program is the pipeline algorithm for Newton Raphson square root computation.

	Program	e=2	e=3	e=4	e=5	On chip
Program off chip	1	1.3	1.5	1.7	1.9	1
	2	1.1	1.2	1.2	1.3	1
Data off chip	1 1	1.5	1.8	2.1	2.3	1
·	2	1.2	1.4	1.6	1.7	1
Program and data off chip	1 1	1.8	2.2	2.7	3.2	1
	2	1.3	1.6	1.8	2.0	1

Table 11.7 IMS T414 external memory performance

# 11.6 Interrupt latency

If the process is a high priority one and no other high priority process is running, the latency is as described in table 11.8. The timings given are in full processor cycles **TPCLPCL**; the number of **Tm** states is also given where relevant. Maximum latency assumes all memory accesses are internal ones.

Table 11.8 Interrupt latency

	Typical		Maximum		
	TPCLPCL	Tm	TPCLPCL	Tm	
IMS T414	19	38	53	116	

# 12 Package specifications

# 12.1 84 pin grid array package

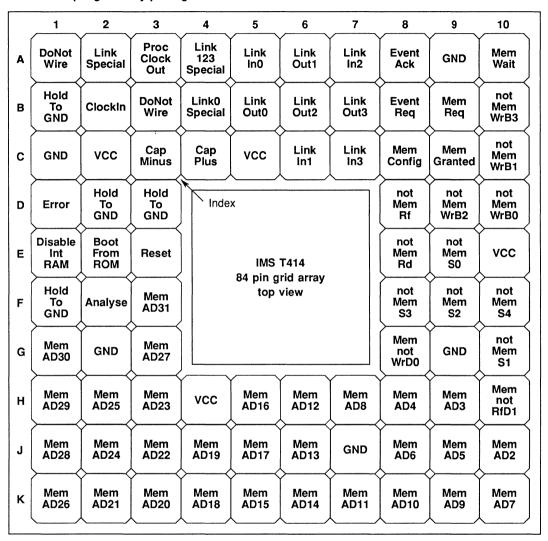


Figure 12.1 IMS T414 84 pin grid array package pinout

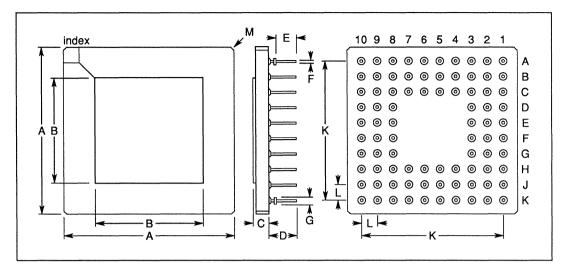


Figure 12.2 84 pin grid array package dimensions

Table 12.1 84 pin grid array package dimensions

	Millim	etres	Inc	nes	
DIM	NOM	TOL	МОМ	TOL	Notes
Α	26.924	±0.254	1.060	±0.010	
В	17.019	±0.127	0.670	±0.005	
C	2.456	±0.278	0.097	±0.011	1
D	4.572	±0.127	0.180	±0.005	
E	3.302	±0.127	0.130	±0.005	
F	0.457	±0.025	0.018	$\pm 0.002$	Pin diameter
G	1.143	±0.127	0.045	±0.005	Flange diameter
K	22.860	±0.127	0.900	$\pm 0.005$	
L	2.540	±0.127	0.100	±0.005	
М	0.508		0.020		Chamfer

Package weight is approximately 7.2 grams

Table 12.2 84 pin grid array package junction to ambient thermal resistance

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTE
$\theta$ JA	At 400 linear ft/min transverse air flow			35	°C/W	

# 12.2 84 pin PLCC J-bend package

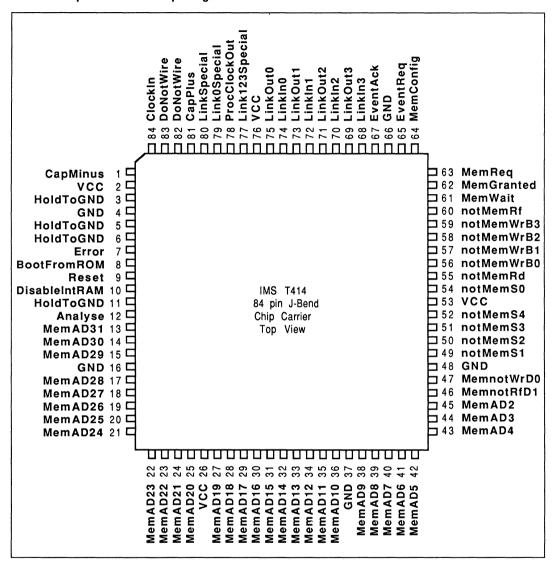


Figure 12.3 IMS T414 84 pin PLCC J-bend package pinout

# Notes

1 Since the manufacture of the IMS T414, the pin numbers for the 84 pin J-bend chip carrier have been rearranged, however the pin functions remain the same.

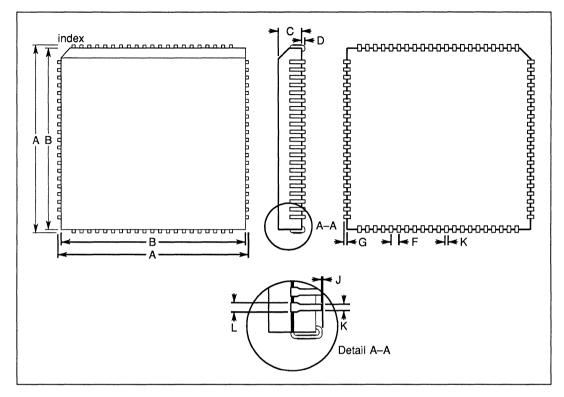


Figure 12.4 84 pin PLCC J-bend package dimensions

Table 12.3 84 pin PLCC J-bend package dimensions

	Millim	etres	Incl	nes	
DIM	NOM	TOL	NOM	TOL	Notes
Α	30.226	±0.127	1.190	±0.005	
В	29.312	±0.127	1.154	±0.005	
С	3.810	±0.127	0.150	±0.005	
D	0.508	±0.127	0.020	±0.005	
F	1.270	±0.127	0.050	±0.005	
G	0.457	±0.127	0.018	±0.005	
J	0.000	±0.051	0.000	±0.002	
K	0.457	±0.127	0.018	±0.005	
L	0.762	±0.127	0.030	±0.005	

Package weight is approximately 7.0 grams

Table 12.4 84 pin PLCC J-bend package junction to ambient thermal resistance

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTE
$\theta$ JA	At 400 linear ft/min transverse air flow		35		°C/W	

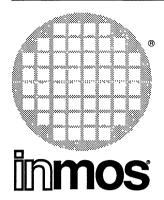
# 13 Ordering

This section indicates the designation of speed and package selections for the various devices. Speed of **ClockIn** is 5 MHz for all parts. Transputer processor cycle time is nominal; it can be calculated more exactly using the phase lock loop factor **PLLx**, as detailed in the external memory section.

For availability contact local INMOS sales office or authorised distributor.

Table 13.1 IMS T414 ordering details

INMOS designation	Processor clock speed	Processor cycle time	PLLx	Package
IMS T414-G15S	15 MHz	67 ns	3.0	Ceramic Pin Grid
IMS T414-G20S	20 MHz	50 ns	4.0	Ceramic Pin Grid
IMS T414-J15S	15 MHz	67 ns	3.0	Plastic PLCC J-Bend
IMS T414-J20S	20 MHz	50 ns	4.0	Plastic PLCC J-Bend



# IMS T222 transputer

# **Engineering Data**

# **FEATURES**

16 bit architecture

50 ns internal cycle time

20 MIPS (peak) instruction rate

IMS T222-20 is pin compatible with IMS T225-20

4 Kbytes on-chip static RAM

40 Mbytes/sec sustained data rate to internal memory

64 Kbytes directly addressable external memory

20 Mbytes/sec sustained data rate to external memory

950 ns response to interrupts

Four INMOS serial links 5/10/20 Mbits/sec Bi-directional data rate of 2.4 Mbytes/sec per link

Internal timers of 1  $\mu$ s and 64  $\mu$ s

Boot from ROM or communication links

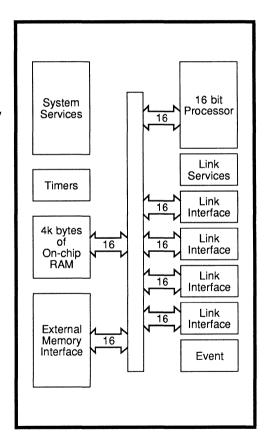
Single 5 MHz clock input

Single +5V ±5% power supply

MIL-STD-883C processing is available

## **APPLICATIONS**

Real time processing
Microprocessor applications
High speed multi processor systems
Industrial control
Robotics
System simulation
Digital signal processing
Telecommunications
Fault tolerant systems
Medical instrumentation



# 1 Introduction

The IMS T222 transputer is a 16 bit CMOS microcomputer with 4 Kbytes on-chip RAM for high speed processing, an external memory interface and four standard INMOS communication links. The instruction set achieves efficient implementation of high level languages and provides direct support for the OCCAM model of concurrency when using either a single transputer or a network. Procedure calls, process switching and typical interrupt latency are sub-microsecond. A device running at 20 MHz achieves an instruction throughput of 20 MIPS peak. The extended temperature version of the device complies with MIL-STD-883C.

For convenience of description, the IMS T222 operation is split into the basic blocks shown in figure 1.1.

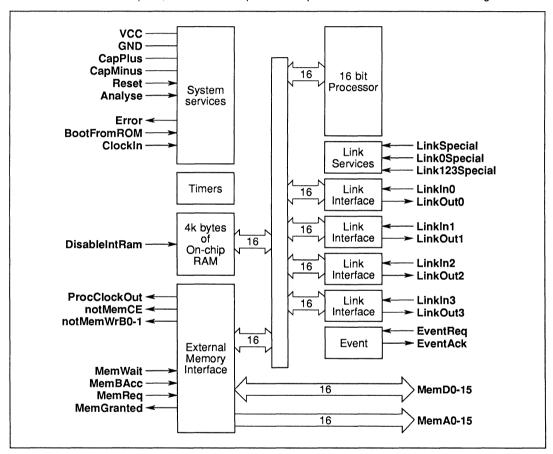


Figure 1.1 IMS T222 block diagram

The IMS T222 can directly access a linear address space of 64 Kbytes. The 16 bit wide non-multiplexed external memory interface provides a data rate of up to 2 bytes every 100 nanoseconds (20 Mbytes/sec) for a 20 MHz device.

System Services include processor reset and bootstrap control, together with facilities for error analysis.

1 Introduction 401

The INMOS communication links allow networks of transputers to be constructed by direct point to point connections with no external logic. The links support the standard operating speed of 10 Mbits/sec, but also operate at 5 or 20 Mbits/sec. The links support overlapped acknowledge; each IMS T222 link can transfer data bi-directionally at up to 2.05 Mbytes/sec.

The IMS T222 is designed to implement the OCCAM language, detailed in the OCCAM Reference Manual, but also efficiently supports other languages such as C and Pascal. Access to the transputer at machine level is seldom required, but if necessary refer to the *Transputer Instruction Set - A Compiler Writers' Guide*.

This data sheet supplies hardware implementation and characterisation details for the IMS T222. It is intended to be read in conjunction with the Transputer Architecture chapter, which details the architecture of the transputer and gives an overview of OCCAM.

# 2 Pin designations

Table 2.1 IMS T222 system services

Pin	In/Out	Function
VCC, GND		Power supply and return
CapPlus, CapMinus		External capacitor for internal clock power supply
Clockin	in	Input clock
Reset	in	System reset
Error	out	Error indicator
Analyse	in	Error analysis
BootFromRom	in	Bootstraps from external ROM or from link
DisableIntRAM	in	Disable internal RAM
HoldToGND		Must be connected to GND

Table 2.2 IMS T222 external memory interface

Pin	In/Out	Function
ProcClockOut	out	Processor clock
MemA0-15	out	Sixteen address lines
MemD0-15	in/out	Sixteen data lines
notMemWrB0-1	out	Two byte-addressing write strobes
notMemCE	out	Chip enable
MemBAcc	in	Byte access mode selector
MemWait	in	Memory cycle extender
MemReq	in	Direct memory access request
MemGranted	out	Direct memory access granted

Table 2.3 IMS T222 event

Pin	In/Out	Function
EventReq	in	Event request
EventAck	out	Event request acknowledge

Table 2.4 IMS T222 link

Pin	In/Out	Function
LinkIn0-3	in	Four serial data input channels
LinkOut0-3	out	Four serial data output channels
LinkSpecial	in	Select non-standard speed as 5 or 20 Mbits/sec
Link0Special	l in	Select special speed for Link 0
Link123Special	in	Select special speed for Links 1,2,3

Signal names are prefixed by **not** if they are active low, otherwise they are active high. Pinout details for various packages are given on page 448.

# 3 Processor

The 16 bit processor contains instruction processing logic, instruction and work pointers, and an operand register. It directly accesses the high speed 4 Kbyte on-chip memory, which can store data or program. Where larger amounts of memory or programs in ROM are required, the processor has access to 64 Kbytes of memory via the External Memory Interface (EMI).

# 3.1 Registers

The design of the transputer processor exploits the availability of fast on-chip memory by having only a small number of registers; six registers are used in the execution of a sequential process. The small number of registers, together with the simplicity of the instruction set, enables the processor to have relatively simple (and fast) data-paths and control logic. The six registers are:

The workspace pointer which points to an area of store where local variables are kept.

The instruction pointer which points to the next instruction to be executed.

The operand register which is used in the formation of instruction operands.

The A, B and C registers which form an evaluation stack.

A, B and C are sources and destinations for most arithmetic and logical operations. Loading a value into the stack pushes B into C, and A into B, before loading A. Storing a value from A, pops B into A and C into B.

Expressions are evaluated on the evaluation stack, and instructions refer to the stack implicitly. For example, the *add* instruction adds the top two values in the stack and places the result on the top of the stack. The use of a stack removes the need for instructions to respecify the location of their operands. Statistics gathered from a large number of programs show that three registers provide an effective balance between code compactness and implementation complexity.

No hardware mechanism is provided to detect that more than three values have been loaded onto the stack. It is easy for the compiler to ensure that this never happens.

Any location in memory can be accessed relative to the workpointer register, enabling the workspace to be of any size.

Further register details are given in Transputer Instruction Set - A Compiler Writers' Guide.

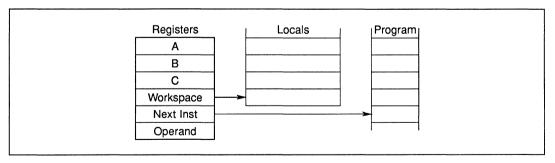


Figure 3.1 Registers

#### 3.2 Instructions

The instruction set has been designed for simple and efficient compilation of high-level languages. All instructions have the same format, designed to give a compact representation of the operations occurring most frequently in programs.

Each instruction consists of a single byte divided into two 4-bit parts. The four most significant bits of the byte are a function code and the four least significant bits are a data value.

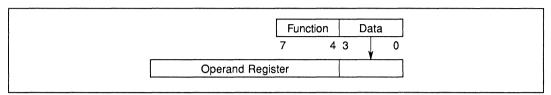


Figure 3.2 Instruction format

## 3.2.1 Direct functions

iump

The representation provides for sixteen functions, each with a data value ranging from 0 to 15. Ten of these, shown in table 3.1, are used to encode the most important functions.

load constant add constant
load local store local load local pointer
load non-local store non-local

conditional jump

call

Table 3.1 Direct functions

The most common operations in a program are the loading of small literal values and the loading and storing of one of a small number of variables. The *load constant* instruction enables values between 0 and 15 to be loaded with a single byte instruction. The *load local* and *store local* instructions access locations in memory relative to the workspace pointer. The first 16 locations can be accessed using a single byte instruction.

The *load non-local* and *store non-local* instructions behave similarly, except that they access locations in memory relative to the *A* register. Compact sequences of these instructions allow efficient access to data structures, and provide for simple implementations of the static links or displays used in the implementation of high level programming languages such as OCCAM, C or Pascal.

## 3.2.2 Prefix functions

Two more function codes allow the operand of any instruction to be extended in length; *prefix* and *negative prefix*.

All instructions are executed by loading the four data bits into the least significant four bits of the operand register, which is then used as the instruction's operand. All instructions except the prefix instructions end by clearing the operand register, ready for the next instruction.

The *prefix* instruction loads its four data bits into the operand register and then shifts the operand register up four places. The *negative prefix* instruction is similar, except that it complements the operand register before shifting it up. Consequently operands can be extended to any length up to the length of the operand register by a sequence of prefix instructions. In particular, operands in the range -256 to 255 can be represented using one prefix instruction.

The use of prefix instructions has certain beneficial consequences. Firstly, they are decoded and executed in the same way as every other instruction, which simplifies and speeds instruction decoding. Secondly, they simplify language compilation by providing a completely uniform way of allowing any instruction to take an operand of any size. Thirdly, they allow operands to be represented in a form independent of the processor wordlength.

## 3.2.3 Indirect functions

The remaining function code, *operate*, causes its operand to be interpreted as an operation on the values held in the evaluation stack. This allows up to 16 such operations to be encoded in a single byte instruction. However, the prefix instructions can be used to extend the operand of an *operate* instruction just like any other. The instruction representation therefore provides for an indefinite number of operations.

Encoding of the indirect functions is chosen so that the most frequently occurring operations are represented without the use of a prefix instruction. These include arithmetic, logical and comparison operations such as add, exclusive or and greater than. Less frequently occurring operations have encodings which require a single prefix operation.

# 3.2.4 Expression evaluation

Evaluation of expressions sometimes requires use of temporary variables in the workspace, but the number of these can be minimised by careful choice of the evaluation order.

Program	Mnem	onic
x := 0	ldc	0
	stl	X
x := #24	pfix Idc	2 4
	ldc	4
	stl	X
x := y + z	ldl	y z
	ldl	Z
	add	
	stl	X

Table 3.2 Expression evaluation

## 3.2.5 Efficiency of encoding

Measurements show that about 70% of executed instructions are encoded in a single byte; that is, without the use of prefix instructions. Many of these instructions, such as *load constant* and *add* require just one processor cycle.

The instruction representation gives a more compact representation of high level language programs than more conventional instruction sets. Since a program requires less store to represent it, less of the memory bandwidth is taken up with fetching instructions. Furthermore, as memory is word accessed the processor will receive two instructions for every fetch.

Short instructions also improve the effectiveness of instruction pre-fetch, which in turn improves processor performance. There is an extra word of pre-fetch buffer, so the processor rarely has to wait for an instruction fetch before proceeding. Since the buffer is short, there is little time penalty when a jump instruction causes the buffer contents to be discarded.

## 3.3 Processes and concurrency

A process starts, performs a number of actions, and then either stops without completing or terminates complete. Typically, a process is a sequence of instructions. A transputer can run several processes in parallel (concurrently). Processes may be assigned either high or low priority, and there may be any number of each (page 407).

The processor has a microcoded scheduler which enables any number of concurrent processes to be executed together, sharing the processor time. This removes the need for a software kernel.

At any time, a concurrent process may be

Active - Being executed.

On a list waiting to be executed.

Inactive - Ready to input.

Ready to output.

- Waiting until a specified time.

The scheduler operates in such a way that inactive processes do not consume any processor time. It allocates a portion of the processor's time to each process in turn. Active processes waiting to be executed are held in two linked lists of process workspaces, one of high priority processes and one of low priority processes (page 407). Each list is implemented using two registers, one of which points to the first process in the list, the other to the last. In the Linked Process List figure 3.3, process S is executing and P, Q and R are active, awaiting execution. Only the low priority process queue registers are shown; the high priority process ones perform in a similar manner.

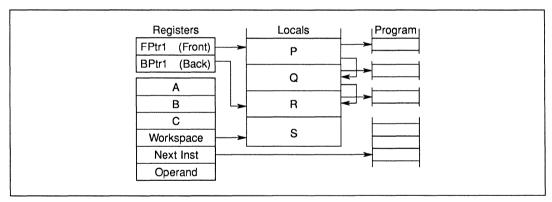


Figure 3.3 Linked process list

Table 3.3 Priority queue control registers

Function	High Priority	Low Priority
Pointer to front of active process list	Fptr0	Fptr1
Pointer to back of active process list	Bptr0	Bptr1

Each process runs until it has completed its action, but is descheduled whilst waiting for communication from another process or transputer, or for a time delay to complete. In order for several processes to operate in parallel, a low priority process is only permitted to run for a maximum of two time slices before it is forcibly descheduled at the next descheduling point (page 410). The time slice period is 5120 cycles of the external 5 MHz clock, giving ticks approximately 1 ms apart.

3 Processor 407

A process can only be descheduled on certain instructions, known as descheduling points (page 410). As a result, an expression evaluation can be guaranteed to execute without the process being timesliced part way through.

Whenever a process is unable to proceed, its instruction pointer is saved in the process workspace and the next process taken from the list. Process scheduling pointers are updated by instructions which cause scheduling operations, and should not be altered directly. Actual process switch times are less than 1  $\mu$ s, as little state needs to be saved and it is not necessary to save the evaluation stack on rescheduling.

The processor provides a number of special operations to support the process model, including *start process* and *end process*. When a main process executes a parallel construct, *start process* instructions are used to create the necessary additional concurrent processes. A *start process* instruction creates a new process by adding a new workspace to the end of the scheduling list, enabling the new concurrent process to be executed together with the ones already being executed. When a process is made active it is always added to the end of the list, and thus cannot pre-empt processes already on the same list.

The correct termination of a parallel construct is assured by use of the *end process* instruction. This uses a workspace location as a counter of the parallel construct components which have still to terminate. The counter is initialised to the number of components before the processes are *started*. Each component ends with an *end process* instruction which decrements and tests the counter. For all but the last component, the counter is non zero and the component is descheduled. For the last component, the counter is zero and the main process continues.

# 3.4 Priority

The IMS T222 supports two levels of priority. Priority 1 (low priority) processes are executed whenever there are no active priority 0 (high priority) processes.

High priority processes are expected to execute for a short time. If one or more high priority processes are able to proceed, then one is selected and runs until it has to wait for a communication, a timer input, or until it completes processing.

If no process at high priority is able to proceed, but one or more processes at low priority are able to proceed, then one is selected.

Low priority processes are periodically timesliced to provide an even distribution of processor time between computationally intensive tasks.

If there are **n** low priority processes, then the maximum latency from the time at which a low priority process becomes active to the time when it starts processing is 2**n**-2 timeslice periods. It is then able to execute for between one and two timeslice periods, less any time taken by high priority processes. This assumes that no process monopolises the transputer's time; i.e. it has a distribution of descheduling points (page 410).

Each timeslice period lasts for 5120 cycles of the external 5 MHz input clock (approximately 1 ms at the standard frequency of 5 MHz).

If a high priority process is waiting for an external channel to become ready, and if no other high priority process is active, then the interrupt latency (from when the channel becomes ready to when the process starts executing) is typically 19 processor cycles, a maximum of 53 cycles (assuming use of on-chip RAM).

## 3.5 Communications

Communication between processes is achieved by means of channels. Process communication is point-to-point, synchronised and unbuffered. As a result, a channel needs no process queue, no message queue and no message buffer.

A channel between two processes executing on the same transputer is implemented by a single word in memory; a channel between processes executing on different transputers is implemented by point-to-point

links. The processor provides a number of operations to support message passing, the most important being input message and output message.

The *input message* and *output message* instructions use the address of the channel to determine whether the channel is internal or external. Thus the same instruction sequence can be used for both, allowing a process to be written and compiled without knowledge of where its channels are connected.

The process which first becomes ready must wait until the second one is also ready. A process performs an input or output by loading the evaluation stack with a pointer to a message, the address of a channel, and a count of the number of bytes to be transferred, and then executing an *input message* or *output message* instruction. Data is transferred if the other process is ready. If the channel is not ready or is an external one the process will deschedule.

## 3.6 Timers

The transputer has two 16 bit timer clocks which 'tick' periodically. The timers provide accurate process timing, allowing processes to deschedule themselves until a specific time.

One timer is accessible only to high priority processes and is incremented every microsecond, cycling completely in approximately 65 milliseconds. The other is accessible only to low priority processes and is incremented every 64 microseconds, giving exactly 15625 ticks in one second. It has a full period of approximately four seconds.

Table 3.4 Timer registers

Clock0	Current value of high priority (level 0) process clock
	Current value of low priority (level 1) process clock
TNextReg0	Indicates time of earliest event on high priority (level 0) timer queue
TNextReg1	Indicates time of earliest event on low priority (level 1) timer queue

The current value of the processor clock can be read by executing a *load timer* instruction. A process can arrange to perform a *timer input*, in which case it will become ready to execute after a specified time has been reached. The *timer input* instruction requires a time to be specified. If this time is in the 'past' then the instruction has no effect. If the time is in the 'future' then the process is descheduled. When the specified time is reached the process is scheduled again.

Figure 3.4 shows two processes waiting on the timer queue, one waiting for time 21, the other for time 31.

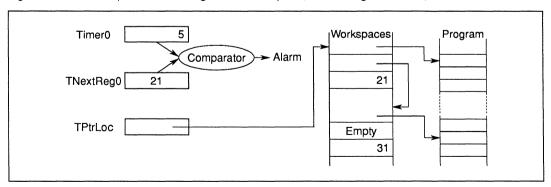


Figure 3.4 Timer registers

# 4 Instruction set summary

The Function Codes table 4.7. gives the basic function code set (page 404). Where the operand is less than 16, a single byte encodes the complete instruction. If the operand is greater than 15, one prefix instruction (*pfix*) is required for each additional four bits of the operand. If the operand is negative the first prefix instruction will be *nfix*.

**Function** Memory **Mnemonic** code code ldc #3 #4 #43 ldc. #35 is coded as pfix #3 #2 #23 ldc #5 #4 #45 #987 ldc is coded as #9 #2 #29 pfix pfix #8 #2 #28 ldc #7 #4 #47 ldc -31 (Idc #FFE1) is coded as nfix #1 #6 #61 ldc #1 #4 #41

Table 4.1 prefix coding

Tables 4.8 to 4.17 give details of the operation codes. Where an operation code is less than 16 (e.g. add: operation code 05), the operation can be stored as a single byte comprising the operate function code F and the operand (5 in the example). Where an operation code is greater than 15 (e.g. ladd: operation code 16), the prefix function code 2 is used to extend the instruction.

Mnemonic		Function code	Memory code
add is coded as	(op. code #5)		#F5
opr	add	#F	#F5
ladd is coded as	(op. code #16)		#21F6
pfix	#1	#2	#21
opr	#6	#F	#F6

Table 4.2 operate coding

The Processor Cycles column refers to the number of periods **TPCLPCL** taken by an instruction executing in internal memory. The number of cycles is given for the basic operation only; where the memory code for an instruction is two bytes, the time for the *prefix* function (one cycle) should be added. For a 20 MHz transputer one cycle is 50 ns. Some instruction times vary. Where a letter is included in the cycles column it is interpreted from table 4.3.

ldent	Interpretation
b	Bit number of the highest bit set in register A. Bit 0 is the least significant bit.
n	Number of places shifted.
w	Number of words in the message. Part words are counted as full words. If the message is not word aligned the number of words is increased to include the part words at either end of the message.

Table 4.3 Instruction set interpretation

The **DE** column of the tables indicates the descheduling/error features of an instruction as described in table 4.4.

Table 4.4 Instruction features

ldent	Feature	See page:
D	The instruction is a descheduling point	410
E	The instruction will affect the Error flag	410, 419

## 4.1 Descheduling points

The instructions in table 4.5 are the only ones at which a process may be descheduled (page 406). They are also the ones at which the processor will halt if the **Analyse** pin is asserted (page 418).

Table 4.5 Descheduling point instructions

input message	output message	output byte	output word
timer alt wait	timer input	stop on error	alt wait
jump	loop end	end process	stop process

### 4.2 Error instructions

The instructions in table 4.6 are the only ones which can affect the Error flag (page 419) directly.

Table 4.6 Error setting instructions

add multiply	add constant	subtract divide	remainder
long add set error	long subtract testerr	long divide	
check word	check subscript from 0	check single	check count from 1

Table 4.7 IMS T222 function codes

Function Code	Memory Code	Mnemonic	Processor Cycles	Name	D E
0	0X	j	3	jump	D
1	1X	ldlp	1	load local pointer	
2	2X	pfix	1	prefix	
3	3X	ldnl	2	load non-local	
4	4X	ldc	1	load constant	
5	5X	ldnlp	1	load non-local pointer	
6	6X	nfix	1	negative prefix	
7	7X	ldl	2	load local	
8	8X	adc	1	add constant	E
9	9X	call	7	call	
Α	AX	cj	2	conditional jump (not taken)	
			4	conditional jump (taken)	
В	вх	ajw	1	adjust workspace	
C	CX	eqc	2	equals constant	
D	DX	sti	1	store local	
E	EX	stnl	2	store non-local	
F	FX	opr	-	operate	

Table 4.8 IMS T222 arithmetic/logical operation codes

Operation Code	Memory Code	Mnemonic	Processor Cycles	Name	D E
46	24F6	and	1	and	
4B	24FB	or	1	or	
33	23F3	xor	1	exclusive or	1 1
32	23F2	not	1	bitwise not	ĺĺ
41	24F1	shl	n+2	shift left	
40	24F0	shr	n+2	shift right	
05	F5	add	1	add	E
0C	FC	sub	1	subtract	E
53	25F3	mul	23	multiply	E
2C	22FC	div	24	divide	E
1F	21FF	rem	21	remainder	E
09	F9	gt	2	greater than	
04	F4	diff	1	difference	
52	25F2	sum	1	sum	
08	F8	prod	b+4	product	

Operation Code	Memory Code	Mnemonic	Processor Cycles	Name	D E
16	21F6	ladd	2	long add	E
38	23F8	Isub	2	long subtract	E
37	23F7	Isum	3	long sum	
4F	24FF	Idiff	3	long diff	
31	23F1	imul	17	long multiply	
1A	21FA	ldiv	19	long divide	E
36	23F6	Ishl	n+3	long shift left (n<16)	
			n-12	long shift left(n≥16)	
35	23F5	ishr	n+3	long shift right (n<16)	
			n-12	long shift right (n≥16)	l
19	21F9	norm	n+5	normalise (n<16)	ļ
			<b>n</b> -10	normalise ( <b>n</b> ≥16)	l
			3	normalise (n=32)	

Table 4.9 IMS T222 long arithmetic operation codes

Table 4.10 IMS T222 general operation codes

Operation Code	Memory Code	Mnemonic	Processor Cycles	Name	D E
00	F0	rev	1	reverse	
3A	23FA	xword	4	extend to word	
56	25F6	cword	5	check word	E
1D	21FD	xdble	2	extend to double	
4C	24FC	csngl	3	check single	E
42	24F2	mint	1	minimum integer	

Table 4.11 IMS T222 indexing/array operation codes

Operation Code	Memory Code	Mnemonic	Processor Cycles	Name	D E
02	F2	bsub	1	byte subscript	
0A	FA	wsub	2	word subscript	
34	23F4	bcnt	2	byte count	
3F	23FF	wcnt	4	word count	
01	F1	lb	5	load byte	
3B	23FB	sb	4	store byte	
4A	24FA	move	2 <b>w</b> +8	move message	

Table 4.12 IMS T222 timer handling operation cod	es:
--	-----

Operation Code	Memory Code	Mnemonic	Processor Cycles	Name	D E
22	22F2	ldtimer	2	load timer	
2B	22FB	tin	30	timer input (time future)	D
			4	timer input (time past)	D
4E	24FE	talt	4	timer alt start	
51	25F1	taltwt	15	timer alt wait (time past)	D
			48	timer alt wait (time future)	D
47	24F7	enbt	8	enable timer	
2E	22FE	dist	23	disable timer	

Table 4.13 IMS T222 input/output operation codes

Operation Code	Memory Code	Mnemonic	Processor Cycles	Name	D E
07	F7	in	2 <b>w</b> +19	input message	D
0B	FB	out	2w+19	output message	D
0F	FF	outword	23	output word	D
0E	FE	outbyte	23	output byte	D
43	24F3	alt	2	alt start	
44	24F4	altwt	5	alt wait (channel ready)	D
			17	alt wait (channel not ready)	D
45	24F5	altend	4	alt end	
49	24F9	enbs	3	enable skip	
30	23F0	diss	4	disable skip	
12	21F2	resetch	3	reset channel	
48	24F8	enbc	7	enable channel (ready)	
			5	enable channel (not ready)	
2F	22FF	disc	8	disable channel	

Table 4.14 IMS T222 control operation codes

Operation Code	Memory Code	Mnemonic	Processor Cycles	Name	D E
20	22F0	ret	5	return	
1B	21FB	ldpi	2	load pointer to instruction	
3C	23FC	gajw	2	general adjust workspace	
06	F6	gcall	4	general call	
21	22F1	lend	10	loop end (loop)	D
			5	loop end (exit)	D

Operation Code	Memory Code	Mnemonic	Processor Cycles	Name	D E
0D	FD	startp	12	start process	D
03	F3	endp	13	end process	D
39	23F9	runp	10	run process	
15	21F5	stopp	11	stop process	
1E	21FE	ldpri	1	load current priority	

Table 4.15 IMS T222 scheduling operation codes

Table 4.16 IMS T222 error handling operation codes

Operation Code	Memory Code	Mnemonic	Processor Cycles	Name	D E
13	21F3	csub0	2	check subscript from 0	E
4D	24FD	ccnt1	3	check count from 1	E
29	22F9	testerr	2	test error false and clear (no error)	
			3	test error false and clear (error)	
10	21F0	seterr	1	set error	E
55	25F5	stoperr	2	stop on error (no error)	D
57	25F7	clrhalterr	1	clear halt-on-error	
58	25F8	sethalterr	1	set halt-on-error	
59	25F9	testhalterr	2	test halt-on-error	

Table 4.17 IMS T222 processor initialisation operation codes

Operation Code	Memory Code	Mnemonic	Processor Cycles	Name	D E
2A	22FA	testpranal	2	test processor analysing	
3E	23FE	saveh	4	save high priority queue registers	
3D	23FD	savel	4	save low priority queue registers	
18	21F8	sthf	1	store high priority front pointer	
50	25F0	sthb	1	store high priority back pointer	
1C	21FC	stif	1	store low priority front pointer	
17	21F7	stib	1	store low priority back pointer	1 1
54	25F4	sttimer	1	store timer	

## 5 System services

System services include all the necessary logic to initialise and sustain operation of the device. They also include error handling and analysis facilities.

#### 5.1 Power

Power is supplied to the device via the VCC and GND pins. Several of each are provided to minimise inductance within the package. All supply pins must be connected. The supply must be decoupled close to the chip by at least one 100 nF low inductance (e.g. ceramic) capacitor between VCC and GND. Four layer boards are recommended; if two layer boards are used, extra care should be taken in decoupling.

Input voltages must not exceed specification with respect to VCC and GND, even during power-up and power-down ramping, otherwise *latchup* can occur. CMOS devices can be permanently damaged by excessive periods of latchup.

### 5.2 CapPlus, CapMinus

The internally derived power supply for internal clocks requires an external low leakage, low inductance  $1\mu F$  capacitor to be connected between **CapPlus** and **CapMinus**. A ceramic capacitor is preferred, with an impedance less than 3 Ohms between 100 KHz and 20 MHz. If a polarised capacitor is used the negative terminal should be connected to **CapMinus**. Total PCB track length should be less than 50 mm. The connections must not touch power supplies or other noise sources.

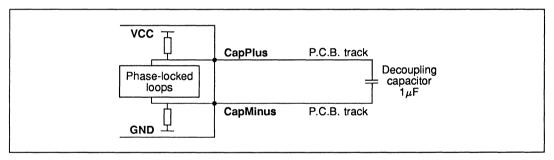


Figure 5.1 Recommended PLL decoupling

### 5.3 Clockin

Transputer family components use a standard clock frequency, supplied by the user on the **Clockin** input. The nominal frequency of this clock for all transputer family components is 5 MHz, regardless of device type, transputer word length or processor cycle time. High frequency internal clocks are derived from **Clockin**, simplifying system design and avoiding problems of distributing high speed clocks externally.

A number of transputer devices may be connected to a common clock, or may have individual clocks providing each one meets the specified stability criteria. In a multi-clock system the relative phasing of **ClockIn** clocks is not important, due to the asynchronous nature of the links. Mark/space ratio is unimportant provided the specified limits of **ClockIn** pulse widths are met.

Oscillator stability is important. ClockIn must be derived from a crystal oscillator; RC oscillators are not sufficiently stable. ClockIn must not be distributed through a long chain of buffers. Clock edges must be monotonic and remain within the specified voltage and time limits.

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTE
TDCLDCH	ClockIn pulse width low	40			ns	1
TDCHDCL	ClockIn pulse width high	40	l		ns	1
TDCLDCL	ClockIn period		200		ns	1,2,4
TDCerror	ClockIn timing error	1	ļ	±0.5	ns	1,3
TDC1DC2	Difference in ClockIn for 2 linked devices	ſ	ŀ	400	ppm	1,4
TDCr	ClockIn rise time	ì	ŀ	10	ns	1,5
TDCf	ClockIn fall time			8	ns	1,5

Table 5.1 Input clock

#### Notes

- 1 These paramters are not tested.
- 2 Measured between corresponding points on consecutive falling edges.
- 3 Variation of individual falling edges from their nominal times.
- 4 This value allows the use of 200ppm crystal oscillators for two devices connected together by a link.
- 5 Clock transitions must be monotonic within the range VIH to VIL (table 10.3).

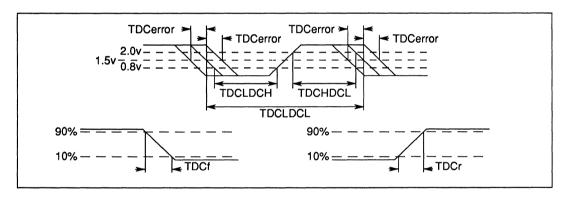


Figure 5.2 ClockIn timing

### 5.4 Reset

Reset can go high with VCC, but must at no time exceed the maximum specified voltage for VIH. After VCC is valid ClockIn should be running for a minimum period TDCVRL before the end of Reset. The falling edge of Reset initialises the transputer and starts the bootstrap routine. Link outputs are forced low during reset; link inputs and EventReq should be held low. Memory request (DMA) must not occur whilst Reset is high but can occur before bootstrap (page 431). If BootFromRom is high bootstrapping will take place immediately after Reset goes low, using data from external memory; otherwise the transputer will await an input from any link. The processor will be in the low priority state.

### 5.5 Bootstrap

The transputer can be bootstrapped either from a link or from external ROM. To facilitate debugging, **Boot-FromRom** may be dynamically changed but must obey the specified timing restrictions. It is sampled once only by the transputer, before the first instruction is executed after **Reset** is taken low.

If **BootFromRom** is connected high (e.g. to **VCC**) the transputer starts to execute code from the top two bytes in external memory, at address #7FFE. This location should contain a backward jump to a program

in ROM. Following this access, **BootFromRom** may be taken low if required. The processor is in the low priority state, and the *W* register points to *MemStart* (page 420).

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTE
TPVRH	Power valid before Reset	10			ms	
TRHRL	Reset pulse width high	8			ClockIn	1
TDCVRL	Clocklin running before Reset end	10			ms	2
TAHRH	Analyse setup before Reset	3			ms	
TRLAL	Analyse hold after Reset end	1			ClockIn	1
TBRVRL	BootFromRom setup	0			ms	
TRLBRX	BootFromRom hold after Reset	0	1		ms	3
TAI RRX	BootFromBom hold after Analyse	1	1	l	1	3

Table 5.2 Reset and Analyse

- 1 Full periods of Clockin TDCLDCL required.
- 2 At power-on reset.
- 3 Must be stable until after end of bootstrap period. See Bootstrap section.

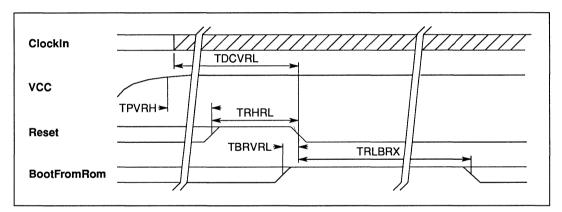


Figure 5.3 Transputer reset timing with Analyse low

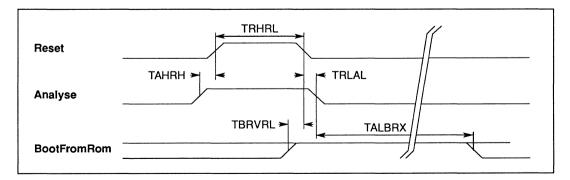


Figure 5.4 Transputer reset and analyse timing

If **BootFromRom** is connected low (e.g. to **GND**) the transputer will wait for the first bootstrap message to arrive on any one of its links. The transputer is ready to receive the first byte on a link within two processor cycles **TPCLPCL** after **Reset** goes low.

If the first byte received (the control byte) is greater than 1 it is taken as the quantity of bytes to be input. The following bytes, to that quantity, are then placed in internal memory starting at location *MemStart*. Following reception of the last byte the transputer will start executing code at *MemStart* as a low priority process. **BootFromRom** may be taken high after reception of the last byte, if required. The memory space immediately above the loaded code is used as work space. Messages arriving on other links after the control byte has been received and on the bootstrapping link after the last bootstrap byte will be retained until a process inputs from them.

## 5.6 Peek and poke

Any location in internal or external memory can be interrogated and altered when the transputer is waiting for a bootstrap from link. If the control byte is 0 then four more bytes are expected on the same link. The first two byte word is taken as an internal or external memory address at which to poke (write) the second two byte word. If the control byte is 1 the next two bytes are used as the address from which to peek (read) a word of data; the word is sent down the output channel of the same link.

Following such a peek or poke, the transputer returns to its previously held state. Any number of accesses may be made in this way until the control byte is greater than 1, when the transputer will commence reading its bootstrap program. Any link can be used, but addresses and data must be transmitted via the same link as the control byte.

### 5.7 Analyse

If Analyse is taken high when the transputer is running, the transputer will halt at the next descheduling point (page 410). From Analyse being asserted, the processor will halt within three time slice periods plus the time taken for any high priority process to complete. As much of the transputer status is maintained as is necessary to permit analysis of the halted machine. Processor flags Error and HaltOnError are not altered at reset, whether Analyse is asserted or not.

Input links will continue with outstanding transfers. Output links will not make another access to memory for data but will transmit only those bytes already in the link buffer. Providing there is no delay in link acknowledgement, the links should be inactive within a few microseconds of the transputer halting.

Reset should not be asserted before the transputer has halted and link transfers have ceased. If BootFrom-Rom is high the transputer will bootstrap as soon as Analyse is taken low, otherwise it will await a control byte on any link. If Analyse is taken low without Reset going high the transputer state and operation are undefined. After the end of a valid Analyse sequence the registers have the values given in table 5.3.

Table 5.3 Register values after Analyse

- I MemStart if bootstrapping from a link, or the external memory bootstrap address if bootstrapping from ROM.
- W MemStart if bootstrapping from ROM, or the address of the first free word after the bootstrap program if bootstrapping from link.
- A The value of I when the processor halted.
- B The value of W when the processor halted, together with the priority of the process when the transputer was halted (i.e. the W descriptor).
- C The ID of the bootstrapping link if bootstrapping from link.

#### 5.8 Error

The **Error** pin is connected directly to the internal *Error* flag and follows the state of that flag. If **Error** is high it indicates an error in one of the processes caused, for example, by arithmetic overflow, divide by zero, array bounds violation or software setting the flag directly (page 410). Once set, the *Error* flag is only cleared by executing the instruction *testerr*. The error is not cleared by processor reset, in order that analysis can identify any errant transputer (page 418).

A process can be programmed to stop if the *Error* flag is set; it cannot then transmit erroneous data to other processes, but processes which do not require that data can still be scheduled. Eventually all processes which rely, directly or indirectly, on data from the process in error will stop through lack of data.

By setting the *HaltOnError* flag the transputer itself can be programmed to halt if *Error* becomes set. If *Error* becomes set after *HaltOnError* has been set, all processes on that transputer will cease but will not necessarily cause other transputers in a network to halt. Setting *HaltOnError* after *Error* will not cause the transputer to halt; this allows the processor reset and analyse facilities to function with the flags in indeterminate states.

An alternative method of error handling is to have the errant process or transputer cause all transputers to halt. This can be done by applying the **Error** output signal of the errant transputer to the **EventReq** pin of a suitably programmed master transputer. Since the process state is preserved when stopped by an error, the master transputer can then use the analyse function to debug the fault. When using such a circuit, note that the *Error* flag is in an indeterminate state on power up; the circuit and software should be designed with this in mind.

Error checks can be removed completely to optimise the performance of a proven program; any unexpected error then occurring will have an arbitrary undefined effect.

If a high priority process pre-empts a low priority one, status of the *Error* and *HaltOnError* flags is saved for the duration of the high priority process and restored at the conclusion of it. Status of the *Error* flag is transmitted to the high priority process but the *HaltOnError* flag is cleared before the process starts. Either flag can be altered in the process without upsetting the error status of any complex operation being carried out by the pre-empted low priority process.

In the event of a transputer halting because of *HaltOnError*, the links will finish outstanding transfers before shutting down. If **Analyse** is asserted then all inputs continue but outputs will not make another access to memory for data.

After halting due to the *Error* flag changing from 0 to 1 whilst *HaltOnError* is set, register *I* points two bytes past the instruction which set *Error*. After halting due to the **Analyse** pin being taken high, register *I* points one byte past the instruction being executed. In both cases *I* will be copied to register *A*.

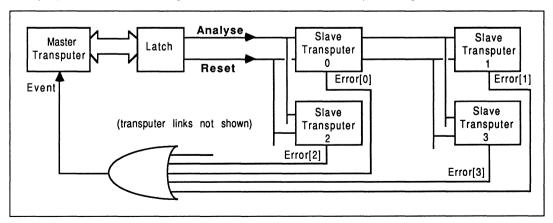


Figure 5.5 Error handling in a multi-transputer system

## 6 Memory

The IMS T222 has 4 Kbytes of fast internal static memory for high rates of data throughput. Each internal memory access takes one processor cycle **ProcClockOut** (page 422). The transputer can also access an additional 60 Kbytes of external memory space. Internal and external memory are part of the same linear address space. Internal RAM can be disabled by holding **DisableIntRAM** high. All internal addresses are then mapped to external RAM. This pin should not be altered after **Reset** has been taken low.

IMS T222 memory is byte addressed, with words aligned on two-byte boundaries. The least significant byte of a word is the lowest addressed byte.

The bits in a byte are numbered 0 to 7, with bit 0 the least significant. The bytes are numbered from 0, with byte 0 the least significant. In general, wherever a value is treated as a number of component values, the components are numbered in order of increasing numerical significance, with the least significant component numbered 0. Where values are stored in memory, the least significant component value is stored at the lowest (most negative) address.

Internal memory starts at the most negative address #8000 and extends to #8FFF. User memory begins at #8024: this location is given the name *MemStart*.

The reserved area of internal memory below MemStart is used to implement link and event channels.

Two words of memory are reserved for timer use, *TPtrLoc0* for high priority processes and *TPtrLoc1* for low priority processes. They either indicate the relevant priority timer is not in use or point to the first process on the timer queue at that priority level.

Values of certain processor registers for the current low priority process are saved in the reserved *IntSaveLoc* locations when a high priority process pre-empts a low priority one.

External memory space starts at #9000 and extends up through #0000 to #7FFF. ROM bootstrapping code must be in the most positive address space, starting at #7FFE. Address space immediately below this is conventionally used for ROM based code.

Figure 6.1 IMS T222 memory map

### Notes

1 These locations are used as auxiliary processor registers and should not be manipulated by the user. Like processor registers, their contents may be useful for implementing debugging tools (Analyse, page 418). For details see *Transputer Instruction Set - A Compiler Writers' Guide*.

# 7 External memory interface

The IMS T222 External Memory Interface (EMI) allows access to a 16 bit address space via separate address and data buses. The data bus can be configured for either 16 bit or 8 bit memory access, allowing the use of a single bank of byte-wide memory. Both word-wide and byte-wide access may be mixed in a single memory system (page 428).

#### 7.1 ProcClockOut

This clock is derived from the internal processor clock, which is in turn derived from **Clockin**. Its period is equal to one internal microcode cycle time, and can be derived from the formula

### TPCLPCL = TDCLDCL / PLLx

where TPCLPCL is the ProcClockOut Period, TDCLDCL is the ClockIn Period and PLLx is the phase lock loop factor for the relevant speed part, obtained from the ordering details (Ordering section).

Edges of the various external memory strobes are synchronised by, but do not all coincide with, rising or falling edges of ProcClockOut.

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTE
TPCLPCL	ProcClockOut period	<b>a</b> -2	а	a+2	ns	1,5
TPCHPCL	ProcClockOut pulse width high	<b>b</b> -7	b	<b>b</b> +7	ns	2,5
TPCLPCH	ProcClockOut pulse width low		С		ns	3,5
TPCstab	ProcClockOut stability			8	%	4,5

Table 7.1 ProcClockOut

- 1 a is TDCLDCL/PLLx.
- 2 b is 0.5\*TPCLPCL (half the processor clock period).
- 3 c is TPCLPCL-TPCHPCL.
- 4 Stability is the variation of cycle periods between two consecutive cycles, measured at corresponding points on the cycles.
- 5 This parameter is sampled and not 100% tested.

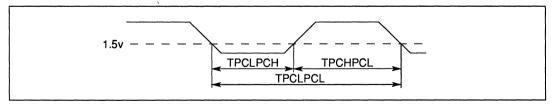


Figure 7.1 IMS T222 ProcClockOut timing

#### 7.2 Tstates

The external memory cycle is divided into four Tstates with the following functions:

- T1 Address and control setup time.
- T2 Data setup time.
- T3 Data read/write.
- T4 Data and address hold after access.

Each **Tstate** is half a processor cycle **TPCLPCL** long. An external memory cycle is always a complete number of cycles **TPCLPCL** in length and the start of **T1** always coincides with a rising edge of **ProcClockOut**. **T2** can be extended indefinitely by adding externally generated wait states of one complete processor cycle each.

### 7.3 Internal access

During an internal memory access cycle the external memory interface address bus **MemA0-15** reflects the word address used to access internal RAM, **notMemWrB0-1** and **notMemCE** are inactive and the data bus **MemD0-15** is tristated. This is true unless and until a DMA (memory request) activity takes place, when the lines will be placed in a high impedance state by the transputer.

Bus activity is not adequate to trace the internal operation of the transputer in full, but may be used for hardware debugging in conjuction with peek and poke (page 418).

## 7.4 MemA0-15

External memory addresses are output on a non-multiplexed 16 bit bus. The address is valid at the start of T1 and remains so until the end of T4, with the timing shown. Byte addressing is carried out internally by the IMS T222 for read cycles. For write cycles the relevant bytes in memory are addressed by the write enables notMemWrB0-1.

The transputer places the address bus in a high impedance state during DMA.

#### 7.5 MemD0-15

The non-multiplexed data bus is 16 bits wide. Read cycle data may be set up on the bus at any time after the start of T1, but must be valid when the IMS T222 reads it during T4. Data can be removed any time after the rising edge of notMemCE, but must be off the bus no later than the middle of T1, which allows for bus turn-around time before the data lines are driven at the start of T2 in a processor write cycle.

Write data is placed on the bus at the start of T2 and removed at the end of T4. It is normally written into memory in synchronism with notMemCE going high.

The data bus is high impedance except when the transputer is writing data. If only one byte is being written, the unused 8 bits of the bus are high impedance at that time. In byte access mode **MemD8-15** are high impedance during the external memory cycle which writes the most significant (second) byte (page 428).

If the data setup time for read or write is too short it can be extended by inserting wait states at the end of T2 (page 429).

9

		T22	2-20	T222-17			
SYMBOL	PARAMETER	MIN	MAX	MIN	MAX	UNITS	NOTE
TAVEL	Address valid before chip enable low	8		12		ns	1
TELEH	Chip enable low	68	80	83	88	ns	1
TEHEL	Delay before chip enable re-assertion	19		24		ns	1,2
TEHAX	Address hold after chip enable high	3		12		ns	1
TELDrV	Data valid from chip enable low	0	50	0	53	ns	
TAVDrV	Data valid from address valid	0	63	0	65	ns	
TDrVEH	Data setup before chip enable high	22		30		ns	
TEHDrZ	Data hold after chip enable high	0	20	0	24	ns	
TWEHEL	Write enable setup before chip enable low	18		24		ns	3
TPCHEL	ProcClockOut high to chip enable low	8		12		ns	1

Table 7.2 Read

#### Notes

- 1 This parameter is common to read and write cycles and to byte-wide memory accesses.
- 2 These values assume back-to-back external memory accesses.
- 3 Timing is for both write enables notMemWrB0-1.

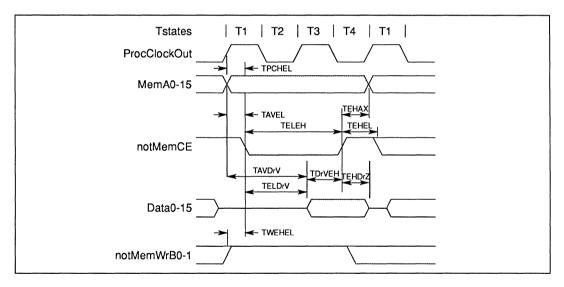


Figure 7.2 IMS T222 external read cycle

### 7.6 notMemWrB0-1

Two write enables are provided, one to write each byte of the word. When writing a word, both write enables are asserted; when writing a byte only the appropriate write enable is asserted. **notMemWrB0** addresses the least significant byte. The write enables are active before the chip enable signal **notMemCE** becomes active, thus reducing memory access time and the risk of bus contention.

The write enables are synchronised with the chip enable signal **notMemCE**, allowing them to be used without **notMemCE** for simple designs.

Data may be strobed into memory using **notMemWrB0-1** without the use of **notMemCE**, as the write enables go high between consecutive external memory write cycles. The write enables are placed in a high impedance state during DMA, and are inactive during internal memory access.

Table 7.3 Write

		T222-20		T222-17			
SYMBOL	PARAMETER	MIN	MAX	MIN	MAX	UNITS	NOTE
TDwVEH	Data setup before chip enable high	50		57		ns	
TEHDwZ	Data hold after write	5	25	12	17	ns	
TDwZEL	Write data invalid to next chip enable	1		12	1	ns	
TWELEL	Write enable setup before chip enable low	-8	3	-4	0	ns	1
TEHWEH	Write enable hold after chip enable high	-3	6	0	4	ns	1

#### Notes

1 Timing is for both write enables notMemWrB0-1.

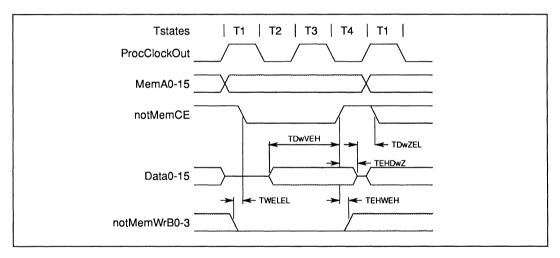


Figure 7.3 IMS T222 external write cycle

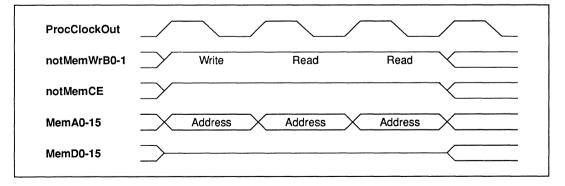


Figure 7.4 IMS T222 bus activity for 3 internal memory cycles

### 7.7 notMemCE

The active low signal notMemCE is used to enable external memory on both read and write cycles.

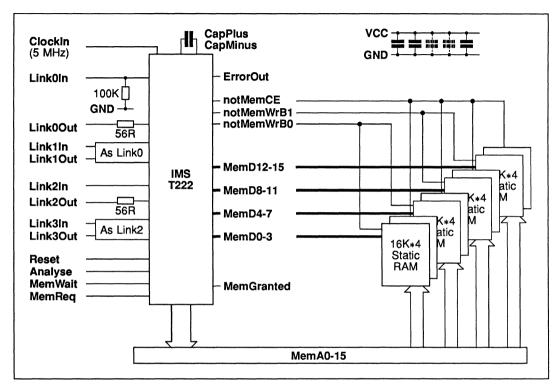


Figure 7.5 IMS T222 static RAM application

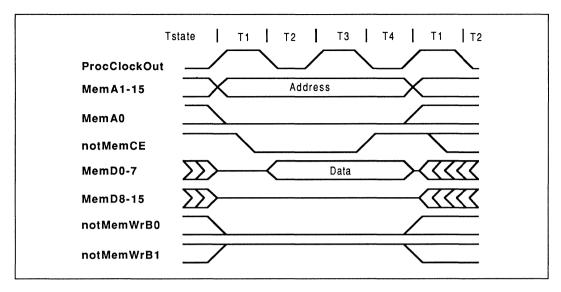


Figure 7.6 IMS T222 Least significant byte write in word access mode

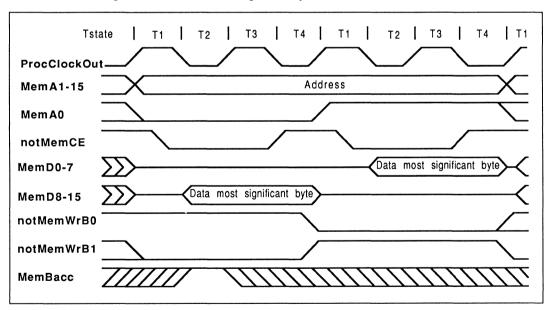


Figure 7.7 IMS T222 Most significant byte write to byte-wide memory

### 7.8 MemBAcc

The IMS T222 will, by default, perform word access at even memory locations. Access to byte-wide memory can be achieved by taking **MemBAcc** high with the timing shown. Where all external memory operations are to byte-wide memory, **MemBAcc** may be wired permanently high. The state of this signal is latched during **T2**.

If **MemBAcc** is low then a full word will be accessed in one external memory cycle, otherwise the high and low bytes of the word will be separately accessed during two consecutive cycles. The first (least significant) byte is accessed at the word address (**MemA0** is low). The second (most significant) byte is accessed at the word address +1 (**MemA0** is high).

With MemBAcc high, the first cycle is identical with a normal word access cycle. However, it will be immediately followed by another memory cycle, which will use MemD0-7 to read or write the second (most significant) byte of data. During this second cycle notMemWrB1 remains high, both for read and write, and MemD8-15 are high impedance. When writing a single byte with MemBAcc high, both the first and second cycles are performed with notMemWrB0 asserted in the appropriate cycle.

		T222-20		T222-20 T222-17		2-17		
SYMBOL	PARAMETER	MIN	MAX	MIN	MAX	UNITS	NOTE	
TELBAH	MemBAcc high from chip enable		12		15	ns		
TELBAL	MemBAcc low from chip enable	32	1	29		ns		

Table 7.4 Byte-wide memory access

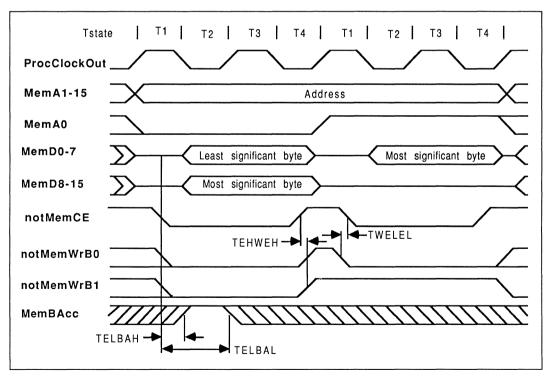


Figure 7.8 IMS T222 word write to byte-wide memory

#### 7.9 MemWait

Taking MemWait high with the timing shown in the diagram will extend the duration of T2 by one processor cycle TPCLPCL. One wait state comprises the pair W1 and W2. MemWait is sampled during T2, and should not change state in this region. If MemWait is still high when sampled in W2 then another wait period will be inserted. This can continue indefinitely. Internal memory access is unaffected by the number of wait states selected.

The wait state generator can be a simple digital delay line, synchronised to **notMemCE**. The **Single Wait State Generator** circuit in figure 7.10 can be extended to provide two or more wait states, as shown in figure 7.11.

Tab	le 7	.5 1	Иer	nory	wait
-----	------	------	-----	------	------

		T22	2-20	T22	2-17		
SYMBOL	PARAMETER	MIN	MAX	MIN	MAX	UNITS	NOTE
TPCHWtH	MemWait asserted after ProcClockOut high		25		27	ns	
TPCHWtL	Wait low after ProcClockOut high	45		39		ns	

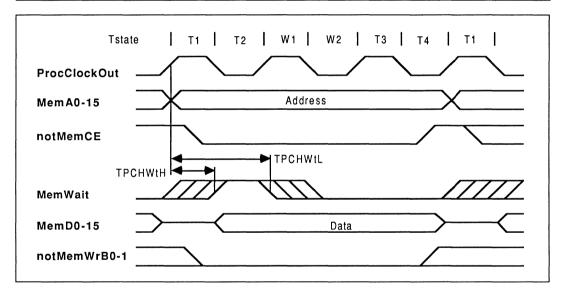


Figure 7.9 IMS T222 memory wait timing

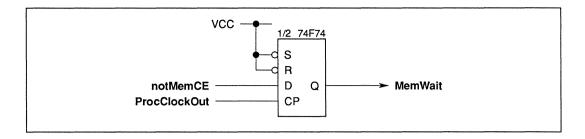


Figure 7.10 Single wait state generator

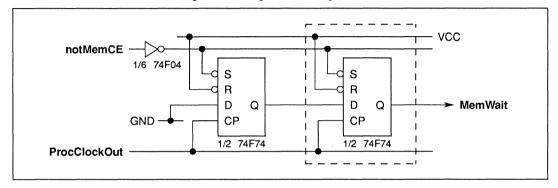


Figure 7.11 Extendable wait state generator

### 7.10 MemReg, MemGranted

Direct memory access (DMA) can be requested at any time by taking the asynchronous MemReq input high. For external memory cycles, the IMS T222 samples MemReq during the first high phase of ProcClockOut after notMemCE goes low. In the absence of an external memory cycle, MemReq is sampled during every rising edge of ProcClockOut. MemA0-15, MemD0-15, notMemWrB0-1 and notMemCE are tristated before MemGranted is asserted.

Removal of **MemReq** is sampled at each rising edge of **ProcClockOut** and **MemGranted** removed with the timing shown. Further external bus activity, either external cycles or reflection of internal cycles, will commence during the next low phase of **ProcClockOut**.

Chip enable, write enables, address bus and data bus are in a high impedance state during DMA. External circuitry must ensure that **notMemCE** and **notMemWrB0-1** do not become active whilst control is being transferred; it is recommended that a 10K resistor is connected from **VCC** to each pin. DMA cannot interrupt an external memory cycle. DMA does not interfere with internal memory cycles in any way, although a program running in internal memory would have to wait for the end of DMA before accessing external memory. DMA cannot access internal memory.

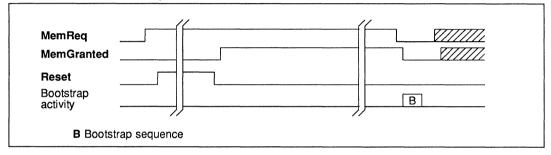


Figure 7.12 IMS T222 DMA sequence at reset

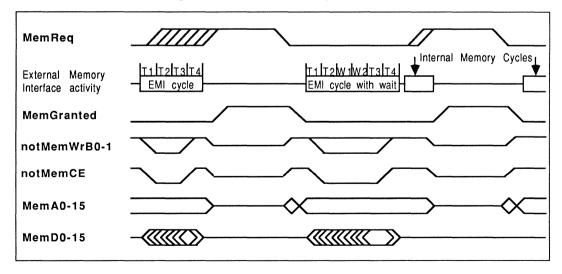


Figure 7.13 IMS T222 operation of MemReg and MemGranted with external and internal memory cycles

DMA allows a bootstrap program to be loaded into external RAM ready for execution after reset. If **MemReq** is held high throughout reset, **MemGranted** will be asserted before the bootstrap sequence begins. **MemReq** must be high at least one period **TDCLDCL** of **ClockIn** before **Reset**. The circuit should be designed to ensure correct operation if **Reset** could interrupt a normal DMA cycle.

Table 7.	6 Memory	request
----------	----------	---------

	T222-20		T22	2-17			
SYMBOL	PARAMETER	MIN	MAX	MIN	MAX	UNITS	NOTE
TMRHMGH	Memory request response time	75	а	100	а	ns	1
TMRLMGL	Memory request end response time	80	155	100	114	ns	
TAZMGH	Addr. bus tristate before MemGranted	0		0		ns	
TAVMGL	Addr. bus active after MemGranted end	0	1	0		ns	
TDZMGH	Data bus tristate before MemGranted	0		0		ns	
TEZMGH	Chip enable tristate before MemGranted	0	1	0		ns	2
TEVMGL	Chip enable active after MemGranted end	-6		0		ns	
TWEZMGH	Write enable tristate before MemGranted	0	l	0		ns	2
TWEVMGL	Write enable active after MemGranted end	-6		0		ns	

- 1 Maximum response time a depends on whether an external memory cycle is in progress and whether byte access is active. Maximum time is (2 processor cycles) + (number of wait state cycles) for word access; in byte access mode this time is doubled.
- 2 When using DMA, notMemCE and notMemWrB0-1 should be pulled up with a resistor (typically 1.2k). Capacitance should be limited to a maximum of 50pF.

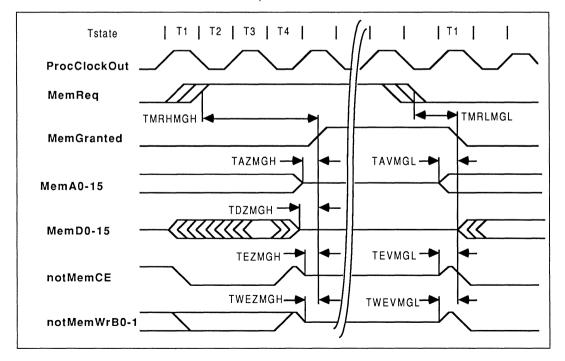


Figure 7.14 IMS T222 memory request timing

### 8 Events

**EventReq** and **EventAck** provide an asynchronous handshake interface between an external event and an internal process. When an external event takes **EventReq** high the external event channel (additional to the external link channels) is made ready to communicate with a process. When both the event channel and the process are ready the processor takes **EventAck** high and the process, if waiting, is scheduled. **EventAck** is removed after **EventReq** goes low.

Only one process may use the event channel at any given time. If no process requires an event to occur **EventAck** will never be taken high. Although **EventReq** triggers the channel on a transition from low to high, it must not be removed before **EventAck** is high. **EventReq** should be low during **Reset**; if not it will be ignored until it has gone low and returned high. **EventAck** is taken low when **Reset** occurs.

If the process is a high priority one and no other high priority process is running, the latency is as described on page 407. Setting a high priority task to wait for an event input allows the user to interrupt a transputer program running at low priority. The time taken from asserting **EventReq** to the execution of the microcode interrupt handler in the CPU is four cycles. The following functions take place during the four cycles:

- Cycle 1 Sample EventReg at pad on the rising edge of ProcClockOut and synchronise.
- Cycle 2 Edge detect the synchronised EventReq and form the interrupt request.
- Cycle 3 Sample interrupt vector for microcode ROM in the CPU.
- **Cycle 4** Execute the interrupt routine for Event rather than the next instruction.

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTE
TVHKH	Event request response	0			ns	
TKHVL	Event request hold	0	ŀ		ns	
TVLKL	Delay before removal of event acknowledge	0		<b>a</b> +7ns		1
TKLVH	Delay before re-assertion of event request	0			l ne	

Table 8.1 Event

#### Notes

1 a is 3 processor cycles TPCLPCL.

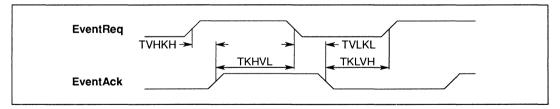


Figure 8.1 IMS T222 event timing

### 9 Links

Four identical INMOS bi-directional serial links provide synchronized communication between processors and with the outside world. Each link comprises an input channel and output channel. A link between two transputers is implemented by connecting a link interface on one transputer to a link interface on the other transputer. Every byte of data sent on a link is acknowledged on the input of the same link, thus each signal line carries both data and control information.

The quiescent state of a link output is low. Each data byte is transmitted as a high start bit followed by a one bit followed by eight data bits followed by a low stop bit. The least significant bit of data is transmitted first. After transmitting a data byte the sender waits for the acknowledge, which consists of a high start bit followed by a zero bit. The acknowledge signifies both that a process was able to receive the acknowledged data byte and that the receiving link is able to receive another byte. The sending link reschedules the sending process only after the acknowledge for the final byte of the message has been received.

The IMS T222 links allow an acknowledge packet to be sent before the data packet has been fully received. This overlapped acknowledge technique is fully compatible with all other INMOS transputer links. The hard output channels are not double buffered. There is thus a pause between transmission of the last byte of a word of the message and the first byte of the next word. This pause time is related to memory speed. Hard input channels have one byte of double buffering and are unlikely to affect the data rate. The dominant factor affecting link bandwidth is therefore the memory bandwidth of the transmitting transputer, as shown in table 9.1. Internal memory access time is similar to zero wait state external access time. Times are for two interconnected IMS T222's with 20 Mbits/sec link speed.

Memory Speed (20MHz device)	Byte Output Time nS	Word Memory Read nS	Unidirectioal Data Rate Mbytes/sec
1 cycle (0 wait)	575	200	1.48
2 cycle (1 wait)	575	250	1.42
3 cycle (2 wait)	575	300	1.38

Table 9.1 Memory/Link speed relationship

The IMS T222 links support the standard INMOS communication speed of 10 Mbits/sec. In addition they can be used at 5 or 20 Mbits/sec. Links are not synchronised with **ClockIn** or **ProcClockOut** and are insensitive to their phases. Thus links from independently clocked systems may communicate, providing only that the clocks are nominally identical and within specification.

Links are TTL compatible and intended to be used in electrically quiet environments, between devices on a single printed circuit board or between two boards via a backplane. Direct connection may be made between devices separated by a distance of less than 300 millimetres. For longer distances a matched 100 ohm transmission line should be used with series matching resistors **RM**. When this is done the line delay should be less than 0.4 bit time to ensure that the reflection returns before the next data bit is sent.

Buffers may be used for very long transmissions. If so, their overall propagation delay should be stable within the skew tolerance of the link, although the absolute value of the delay is immaterial.

Link speeds can be set by LinkSpecial, LinkOSpecial and Link123Special. The link 0 speed can be set independently. Table 9.2 shows uni-directional and bi-directional data rates in Kbytes/sec for each link speed; LinknSpecial is to be read as LinkOSpecial when selecting link 0 speed and as Link123Special for the others. Data rates are quoted for a transputer using internal memory, and will be affected by a factor depending on the number of external memory accesses and the length of the external memory cycle.

Link	Linkn	Kbytes/sec		
Special	Special	Mbits/sec	Uni	Bi
0	0	10	800	1130
0	1	5	430	590
1	0	10	800	1130
1	1	20	1480	2050

Table 9.2 Speed Settings for Transputer Links

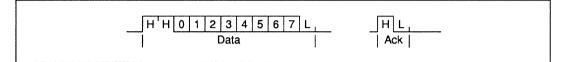


Figure 9.1 IMS T222 link data and acknowledge packets

Tab	le 9	.3 L	.ink
-----	------	------	------

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTE
TJQr	LinkOut rise time			20	ns	1
TJQf	LinkOut fall time			10	ns	1
TJDr	LinkIn rise time			20	ns	1
TJDf	LinkIn fall time			20	ns	1
TJQJD	Buffered edge delay	0			ns	
TJBskew	Variation in TJQJD 20 Mbits/s			3	ns	2
	10 Mbits/s			10	ns	2
	5 Mbits/s		į	30	ns	2
CLIZ	LinkIn capacitance @ f=1MHz			7	pF	1
CLL	LinkOut load capacitance			50	pF	
RM	Series resistor for $100\Omega$ transmission line		56		ohms	

- 1 These paramters are sampled, but are not 100% tested.
- 2 This is the variation in the total delay through buffers, transmission lines, differential receivers etc., caused by such things as short term variation in supply voltages and differences in delays for rising and falling edges.

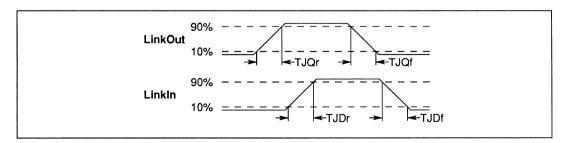


Figure 9.2 IMS T222 link timing

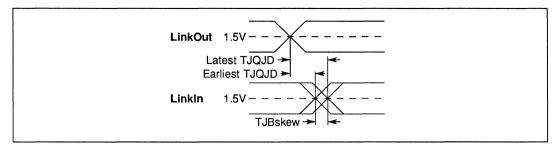


Figure 9.3 IMS T222 buffered link timing

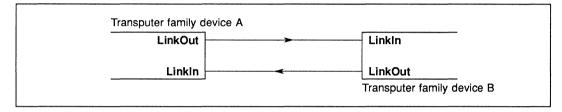


Figure 9.4 IMS T222 Links directly connected

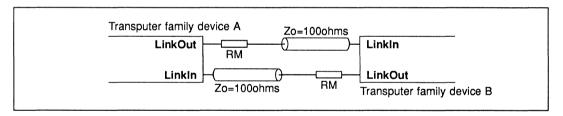


Figure 9.5 IMS T222 Links connected by transmission line

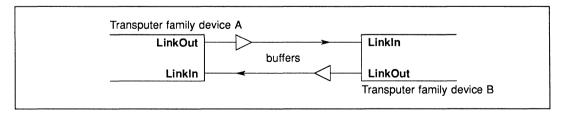


Figure 9.6 IMS T222 Links connected by buffers

# 10 Electrical specifications

#### 10.1 DC electrical characteristics

Table 10.1 Absolute maximum ratings

SYMBOL	PARAMETER	MIN	MAX	UNITS	NOTE
VCC	DC supply voltage	0	7.0	V	1,2,3
VI, VO	Voltage on input and output pins	-0.5	VCC+0.5	V	1,2,3
11	Input current		±25	mA	4
OSCT	Output short circuit time (one pin)		1	s	2
TS	Storage temperature	-65	150	°C	2
TA	Ambient temperature under bias	-55	125	°C	2
PDmax	Maximum allowable dissipation		2	W	

#### Notes

- 1 All voltages are with respect to GND.
- 2 This is a stress rating only and functional operation of the device at these or any other conditions beyond those indicated in the operating sections of this specification is not implied. Stresses greater than those listed may cause permanent damage to the device. Exposure to absolute maximum rating conditions for extended periods may affect reliability.
- 3 This device contains circuitry to protect the inputs against damage caused by high static voltages or electrical fields. However, it is advised that normal precautions be taken to avoid application of any voltage higher than the absolute maximum rated voltages to this high impedance circuit. Unused inputs should be tied to an appropriate logic level such as VCC or GND.
- 4 The input current applies to any input or output pin and applies when the voltage on the pin is between GND and VCC.

SYMBOL UNITS NOTE PARAMETER MIN MAX VCC 4.75 5.25 DC supply voltage 1 VCC ν 1,2 VI, VO Input or output voltage 0 CĹ Load capacitance on any pin 60 pF ်င TA Operating temperature range IMS T222-S 0 70 3 ٥C -55 125 3 TA Operating temperature range IMS T222-M

Table 10.2 Operating conditions

- 1 All voltages are with respect to GND.
- 2 Excursions beyond the supplies are permitted but not recommended; see DC characteristics.
- 3 Air flow rate 400 linear ft/min transverse air flow.

SYMBOL	PARAMI	ETER	MIN	MAX	UNITS	NOTE
VIH	High level input voltage		2.0	VCC+0.5	V	1,2
VIL	Low level input voltage		-0.5	0.8	V	1,2
11	Input current	@ GND <vi<vcc< td=""><td></td><td>±10</td><td><math>\mu</math>A</td><td>1,2</td></vi<vcc<>		±10	$\mu$ A	1,2
VOH	Output high voltage	@ IOH=2mA	VCC-1		١V	1,2
VOL	Output low voltage	@ IOL=4mA		0.4	V	1,2
IOS	Output short circuit currer	nt @ GND <vo<vcc< td=""><td>36</td><td>65</td><td>mA</td><td>1,2,3,6</td></vo<vcc<>	36	65	mA	1,2,3,6
			65	100	mA	1,2,4,6
IOZ	Tristate output current	@ GND <vo<vcc< td=""><td></td><td>±10</td><td><math>\mu</math>A</td><td>1,2</td></vo<vcc<>		±10	$\mu$ A	1,2
PD	Power dissipation			700	mW	2,5
CIN	Input capacitance	@ f=1MHz	1	7	pF	6
COZ	Output capacitance	@ f=1MHz		10	рF	6

Table 10.3 DC characteristics

#### Notes

- 1 All voltages are with respect to GND.
- 2 Parameters for IMS T222-S measured at 4.75V<VCC<5.25V and 0°C<TA<70°C. Input clock frequency = 5 MHz.
- 3 Current sourced from non-link outputs.
- 4 Current sourced from link outputs.
- 5 Power dissipation varies with output loading and program execution.
- 6 This parameter is sampled and not 100% tested.

## 10.2 Equivalent circuits

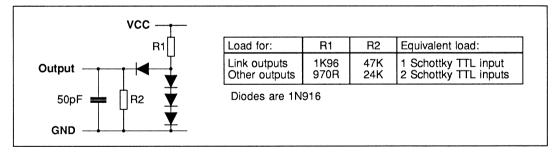


Figure 10.1 Load circuit for AC measurements

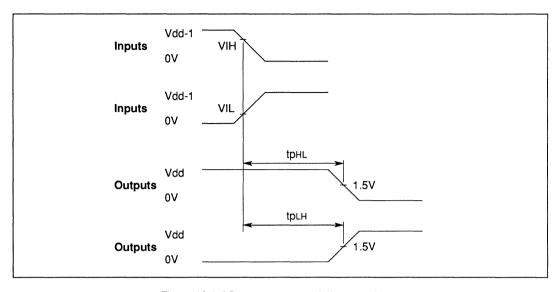


Figure 10.2 AC measurements timing waveforms

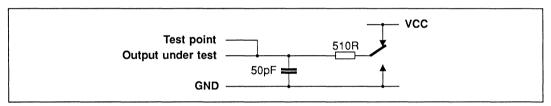


Figure 10.3 Tristate load circuit for AC measurements

## 10.3 AC timing characteristics

Table 10.4 Input, output edges

SYMBOL	PARAMETER	MIN	MAX	UNITS	NOTE
TDr	Input rising edges	2	20	ns	1,2
TDf	Input falling edges	2	20	ns	1,2
TQr	Output rising edges		25	ns	1
TQf	Output falling edges	1	15	ns	1

- 1 Non-link pins; see section on links.
- 2 All inputs except Clockin; see section on Clockin.

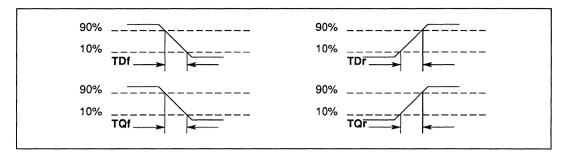
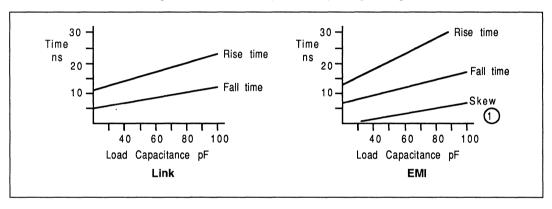


Figure 10.4 IMS T222 input and output edge timing



### Notes

Figure 10.5 Typical rise/fall times

1 Skew is measured between **notMemCE** with a standard load (2 Schottky TTL inputs and 30pF) and **notMemCE** with a load of 2 Schottky TTL inputs and varying capacitance.

## 10.4 Power rating

Internal power dissipation  $P_{INT}$  of transputer and peripheral chips depends on VCC, as shown in figure 10.6.  $P_{INT}$  is substantially independent of temperature.

Total power dissipation  $P_D$  of the chip is

$$P_D = P_{INT} + P_{IO}$$

where  $P_{IO}$  is the power dissipation in the input and output pins; this is application dependent.

Internal working temperature  $T_J$  of the chip is

$$T_J = T_A + \theta J_A * P_D$$

where  $T_A$  is the external ambient temperature in °C and  $\theta J_A$  is the junction-to-ambient thermal resistance in °C/W.  $\theta J_A$  for each package is given in the Packaging Specifications section.

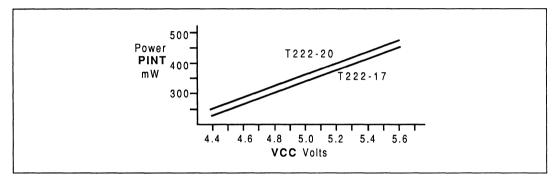


Figure 10.6 IMS T222 internal power dissipation vs VCC

## 11 Performance

The performance of the transputer is measured in terms of the number of bytes required for the program, and the number of (internal) processor cycles required to execute the program. The figures here relate to OCCAM programs. For the same function, other languages should achieve approximately the same performance as occam.

### 11.1 Performance overview

These figures are averages obtained from detailed simulation, and should be used only as an initial guide; they assume operands are of type **INT**. The abbreviations in table 11.1 are used to represent the quantities indicated. In the replicator section of the table, figures in braces {} are not necessary if the number of replications is a compile time constant. To estimate performance, add together the time for the variable references and the time for the operation.

Table 11.1 Key to performance table

np	number of component processes
ne	number of processes earlier in queue
r	1 if INT parameter or array parameter, 0 if not
ts	number of table entries (table size)
w	width of constant in nibbles
p	number of places to shift
Eg	expression used in a guard
Et	timer expression used in a guard
Tb	most significant bit set of multiplier ((-1) if the multiplier is 0)
Tbp	most significant bit set in a positive multiplier when counting from zero ((-1) if the multiplier is 0)
Tbc	most significant bit set in the two's complement of a negative multiplier
nsp	Number of scalar parameters in a procedure
nap	Number of array parameters in a procedure

11 Performance 443

Table 11.2 Performance

	Size (bytes)	Time (cycles)
Names		
variables		
in expression	1.1+r	2.1+2(r)
assigned to or input to in PROC or FUNCTION call,	1.1+r	1.1+( <b>r</b> )
corresponding to an INT parameter	1.1+r	1.1+(r)
channels	1.1	2.1
Array Variables (for single dimension arrays)	,.,	
constant subscript	0	0
variable subscript	5.3	7.3
expression subscript	5.3	7.3
Declarations		
CHAN OF protocol	3.1	3.1
[size] CHAN OF protocol	9.4	2.2 + 20.2*size
PROC	body+2	0
Primitives		
assignment	0	l o
input	4	26.5
output	1	26
STOP	2	25
SKIP	0	0
Arithmetic operators		ļ
<del> </del>	1 2	1
Ĵ	2 2	23 24
/ REM	2	22
>> <<	2	3+ <b>p</b>
Modulo Arithmetic operators	_	1
PLUS	2	2
MINUS	<u> </u>	1 1
TIMES (fast multiply)	1	4+Tb
Boolean operators		
OR	4	8
AND NOT	1	2
Comparison operators		
= constant	0	1
= variable	2	3
<> constant	1 3	3 5
<pre>&lt;&gt; variable &gt;  &lt;</pre>	) 3 1	2
>= <=	2	4
Bit operators	_	,
/\ \/ <b>&gt;&lt;</b> ~	2	2
Expressions	_	_
constant in expression	w	w
check if error	4	6

	Size (bytes)	Time (cycles)
Timers		
timer input	2	3
timer AFTER		
if past time	2	4
with empty timer queue	2 2 2	31
non-empty timer queue	2	38+ <b>ne</b> *9
ALT (timer)		
with empty timer queue	6	52
non-empty timer queue	6	59+ <b>ne</b> ∗9
timer alt guard	8+2 <b>Eg</b> +2 <b>Et</b>	34+2 <b>Eg</b> +2 <b>Et</b>
Constructs		
SEQ	0	0
IF	1.3	1.4
if guard	3	4.3
ALT (non timer)	) 6	26
alt channel guard	10.2+2 <b>Eg</b>	20+2 <b>Eg</b>
skip alt guard	8+2 <b>Eg</b>	10+2 <b>Eg</b>
PAR	11.5+( <b>np</b> -1)∗7.5	19.5+( <b>np</b> -1)*30.5
WHILE	4	12
Procedure or function call		
	3.5+(nsp-2)*1.1	16.5+(nsp-2)*1.1
	+nap∗2.3	+nap*2.3
Replicators		
replicated SEQ	7.3{+5.1}	(-3.8)+15.1*count{+7.1}
replicated IF	12.3{+5.1}	(-2.6)+19.4*count{+7.1}
replicated ALT	24.8{+10.2}	25.4+33.4*count{+14.2}
replicated timer ALT	24.8{+10.2}	62.4+33.4*count(+14.2)
replicated PAR	39.1(+5.1)	(-6.4)+70.9*count(+7.1)

Table 11.3 Performance

### 11.2 Fast multiply, TIMES

The IMS T222 has a fast integer multiplication instruction *product*. The time taken for a fast multiply is 4+Tb. The time taken for a multiplication by zero is 3 cycles. For example, if the multiplier is 1 the time taken is 4 cycles, if the multiplier is -1 (all bits set) the time taken is 19 cycles.

Implementations of high level languages on the transputer may take advantage of this instruction. For example, the OCCAM modulo arithmetic operator **TIMES** is implemented by the instruction and the right-hand operand is treated as the multiplier. The fast multiplication instruction is also used in high level language implementations for the multiplication implicit in multi-dimensional array access.

#### 11.3 Arithmetic

A set of functions are provided within the development system to support the efficient implementation of multiple length integer arithmetic and floating point arithmetic where relevant. In table 11.4 n gives the number of places shifted and all arguments and results are assumed to be local. Full details of these functions are provided in the OCCAM reference manual, supplied as part of the development system and available as a separate publication.

When calculating the execution time of the predefined maths functions, no time needs to be added for calling overhead. These functions are compiled directly into special purpose instructions which are designed to support the efficient implementation of multiple length integer arithmetic and floating point arithmetic.

11 Performance 445

Function		Cycles	+ cycles for parameter access †
LONGADD		2	7
LONGSUM		3	8
LONGSUB		2	7
LONGDIFF		3	8
LONGPROD		18	8
LONGDIV		20	8
SHIFTRIGHT	( <b>n</b> <16)	4+n	8
	( <b>n</b> >=16)	n-11	8
SHIFTLEFT	(n<16)	4+n	8
	( <b>n</b> >=16)	n-11	8
NORMALISE	(n<16)	n+6	7
	(n>=16)	<b>n</b> -9	7
	(n=32)	4	7
ASHIFTRIGHT		SHIFTRIGHT+2	5
ASHIFTLEFT		SHIFTLEFT+4	5
ROTATERIGHT		SHIFTRIGHT	7
ROTATELEFT		SHIFTLEFT	7

Table 11.4 Arithmetic performance

#### 11.4 Floating point operations

Floating point operations for the IMS T222 are provided by a run-time package. This requires approximately 2000 bytes of memory for the double length arithmetic operations, and 2500 bytes for the quadruple length arithmetic operations. Table 11.5 summarizes the estimated performance of the package.

							Processo	or cycles	
							IMS T222		
							Typical Wors		
REAL32	+	_					530	705	
	*						650	705	
	/						1000	1410	
	<	>	=	>=	<=	<b>&lt;&gt;</b>	60	60	
REAL 64	+	_					875	1190	
	*						1490	1950	
	1						2355	3255	
	<	>	=	>=	<=	<b>&lt;&gt;</b>	60	60	

Table 11.5 IMS T222 floating point operations performance

<sup>†</sup> Assuming local variables.

#### 11.5 Effect of external memory

Extra processor cycles may be needed when program and/or data are held in external memory, depending both on the operation being performed, and on the speed of the external memory. After a processor cycle which initiates a write to memory, the processor continues execution at full speed until at least the next memory access.

Whilst a reasonable estimate may be made of the effect of external memory, the actual performance will depend upon the exact nature of the given sequence of operations.

External memory is characterized by the number of extra processor cycles per external memory cycle, denoted as **e**. The value of **e** for the IMS T222 with no wait states is 1.

If a program is stored in external memory, and **e** has the value 2 or 3, then no extra cycles need be estimated for linear code sequences. For larger values of **e**, the number of extra cycles required for linear code sequences may be estimated at (2**e**-1)/4 per byte of program. A transfer of control may be estimated as requiring **e**+3 cycles.

These estimates may be refined for various constructs. In table 11.6 n denotes the number of components in a construct. In the case of **IF**, the n'th conditional is the first to evaluate to **TRUE**, and the costs include the costs of the conditionals tested. The number of bytes in an array assignment or communication is denoted by b.

	IMS T	222
	Program off chip	Data off chip
Boolean expressions	e-1	0
IF	3en-1	en
Replicated IF	6 <b>en</b> +9e-12	(5 <b>e</b> -2) <b>n</b> +6
Replicated SEQ	(4 <b>e</b> -3) <b>n</b> +3e	(4 <b>e</b> -2) <b>n</b> +3-e
PAR	4en	3en
Replicated PAR	(17 <b>e</b> -12)n+9	16 <b>en</b>
ALT	(4e-1)n+9e-4	(4 <b>e</b> -1) <b>n</b> +9 <b>e</b> -3
Array assignment and communication in one transputer	0	max (2 <b>e, eb</b> )

Table 11.6 External memory performance

The following simulation results illustrate the effect of storing program and/or data in external memory. The results are normalized to 1 for both program and data on chip. The first program (Sieve of Erastosthenes) is an extreme case as it is dominated by small, data access intensive loops; it contains no concurrency, communication, or even multiplication or division. The second program is the pipeline algorithm for Newton Raphson square root computation.

**Program** e=1 e=2 e=3 e=4 On chip Program off chip 1.2 1.4 1.8 2.1 2 1.2 1.4 1.6 1.1 1 1.2 1.5 1.8 2.1 Data off chip 1 1 Ź 1.6 1.1 1.3 1.4 1 Program and data off chip 1 2.5 3.0 1 1.4 1.9 2 1.2 1.5 1.8 2.1 1

Table 11.7 IMS T222 external memory performance

#### 11.6 Interrupt latency

If the process is a high priority one and no other high priority process is running, the latency is as described in table 11.8. The timings given are in full processor cycles **TPCLPCL**; the number of **Tm** states is also given where relevant. Maximum latency assumes all memory accesses are internal ones.

Table 11.8 Interrupt latency

	Тур	ical	Maximum		
	TPCLPCL	Tm	TPCLPCL	Tm	
IMS T222	19		53		

#### 12 Package specifications

#### 12.1 68 pin grid array package

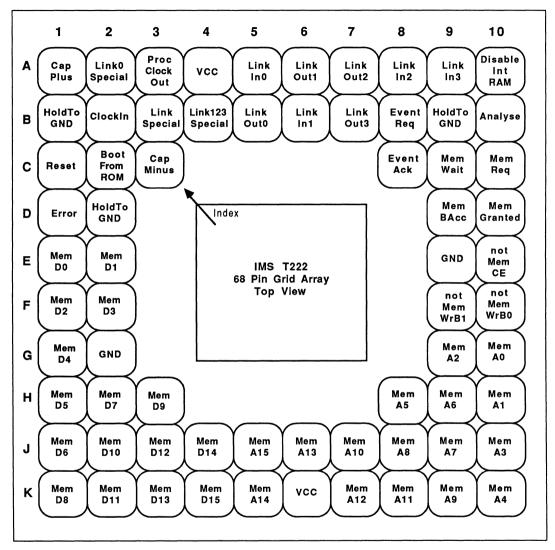


Figure 12.1 IMS T222 68 pin grid array package pinout

12

Figure 12.2 68 pin grid array package dimensions

Table 12.1 68 pin grid array package din	mensions
--	----------

	Millim	netres	Inches		
DIM	NOM	TOL	МОМ	TOL	Notes
Α	26.924	±0.254	1.060	±0.010	
В	17.019	±0.127	0.670	±0.008	
С	2.466	±0.279	0.097	±0.011	
D	4.572	±0.127	0.180	±0.005	
E	3.302	±0.127	0.130	±0.005	
F	0.457	±0.051	0.018	$\pm 0.002$	Pin diameter
G	1.270	±0.127	0.050	$\pm 0.005$	Flange diameter
K	22.860	±0.127	0.900	±0.005	
L	2.540	±0.127	0.100	±0.005	
M	0.508		0.020		Chamfer

Package weight is approximately 6.8 grams

Table 12.2 68 pin grid array package junction to ambient thermal resistance

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTE
$\theta$ JA	At 400 linear ft/min transverse air flow			35	°C/W	

#### 12.2 68 pin PLCC J-bend package

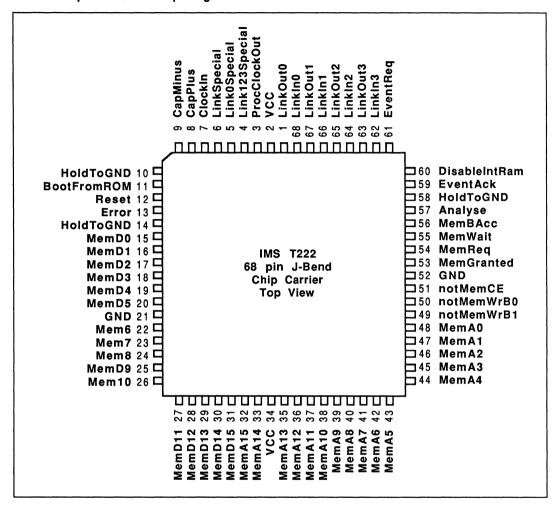


Figure 12.3 IMS T222 68 pin PLCC J-bend package pinout

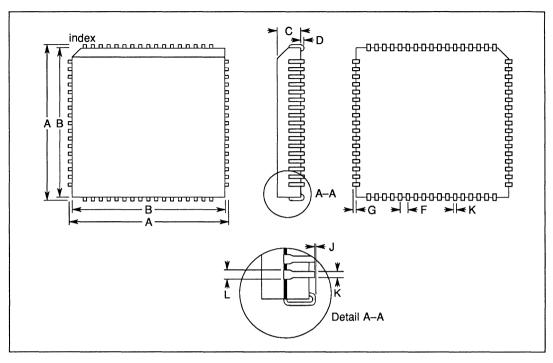


Figure 12.4 68 pin PLCC J-bend package dimensions

Table 12.3 68 pin PLCC J-bend package dimensions

Millimetres		Inches		
МОМ	TOL	МОМ	TOL	Notes
25.146	±0.127	0.990	±0.005	
24.232	±0.127	0.954	±0.005	
3.810	±0.127	0.150	±0.005	
0.508	±0.127	0.020	±0.005	
1.270	±0.127	0.050	±0.005	
0.457	±0.127	0.018	±0.005	
0.000	$\pm 0.051$	0.000	±0.002	
0.457	±0.127	0.018	±0.005	
0.762	±0.127	0.030	±0.005	
	25.146 24.232 3.810 0.508 1.270 0.457 0.000 0.457	NOM         TOL           25.146         ±0.127           24.232         ±0.127           3.810         ±0.127           0.508         ±0.127           1.270         ±0.127           0.457         ±0.127           0.051         ±0.051           0.457         ±0.127	NOM         TOL         NOM           25.146         ±0.127         0.990           24.232         ±0.127         0.954           3.810         ±0.127         0.150           0.508         ±0.127         0.020           1.270         ±0.127         0.050           0.457         ±0.127         0.018           0.000         ±0.051         0.000           0.457         ±0.127         0.018	NOM         TOL         NOM         TOL           25.146         ±0.127         0.990         ±0.005           24.232         ±0.127         0.954         ±0.005           3.810         ±0.127         0.150         ±0.005           0.508         ±0.127         0.020         ±0.005           1.270         ±0.127         0.050         ±0.005           0.457         ±0.127         0.018         ±0.005           0.000         ±0.051         0.000         ±0.002           0.457         ±0.127         0.018         ±0.005

Package weight is approximately 5.0 grams

Table 12.4 68 pin PLCC J-bend package junction to ambient thermal resistance

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTE
$\theta$ JA	At 400 linear ft/min transverse air flow		35		°C/W	

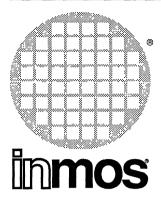
#### 13 Ordering

This section indicates the designation of speed and package selections for the various devices. Speed of **ClockIn** is 5 MHz for all parts. Transputer processor cycle time is nominal; it can be calculated more exactly using the phase lock loop factor **PLLx**, as detailed in the external memory section.

For availability contact local INMOS sales office or authorised distributor.

Table 13.1 IMS T222 ordering details

INMOS designation	Processor clock speed	Processor cycle time	PLLx	Package
IMS T222-G17S	17.5 MHz	57 ns	3.5	Ceramic Pin Grid
IMS T222-G20S	20.0 MHz	50 ns	4.0	Ceramic Pin Grid
IMS T222-J17S	17.5 MHz	57 ns	3.5	Plastic J-Bend
IMS T222-J20S	20.0 MHz	50 ns	4.0	Plastic J-Bend
IMS T222-G17M	17.5 MHz	57 ns	3.5	Ceramic Pin Grid MIL Spec
IMS T222-G20M	20.0 MHz	50 ns	4.0	Ceramic Pin Grid MIL Spec



## IMS T225 transputer

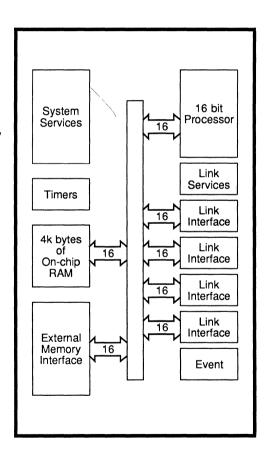
#### **Product Preview**

#### **FEATURES**

16 bit architecture 33 ns internal cycle time 30 MIPS (peak) instruction rate IMS T225-20 is pin compatible with IMS T222-20 Debugging support 4 Kbytes on-chip static RAM 60 Mbytes/sec sustained data rate to internal memory 64 Kbytes directly addressable external memory 30 Mbytes/sec sustained data rate to external memory 630 ns response to interrupts Four INMOS serial links 5/10/20 Mbits/sec Bi-directional data rate of 2.4 Mbytes/sec per link Internal timers of 1  $\mu$ s and 64  $\mu$ s Boot from ROM or communication links Single 5 MHz clock input Single +5V ±5% power supply MIL-STD-883C processing will be available

#### **APPLICATIONS**

Real time processing
Microprocessor applications
High speed multi processor systems
Industrial control
Robotics
System simulation
Digital signal processing
Telecommunications
Fault tolerant systems
Medical instrumentation



#### 1 Introduction

The IMS T225 transputer is a 16 bit CMOS microcomputer with 4 Kbytes on-chip RAM for high speed processing, an external memory interface and four standard INMOS communication links. The instruction set achieves efficient implementation of high level languages and provides direct support for the OCCAM model of concurrency when using either a single transputer or a network. Procedure calls, process switching and typical interrupt latency are sub-microsecond. A device running at 30 MHz achieves an instruction throughput of 15 MIPS.

For convenience of description, the IMS T225 operation is split into the basic blocks shown in figure 1.1.

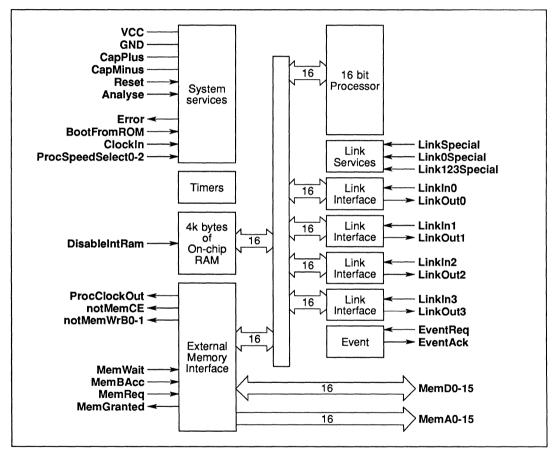


Figure 1.1 IMS T225 block diagram

The IMS T225 is functionally equivalent to the IMS T222 but has the addition of three speed select pins (**ProcSpeedSelect0-2**) and improved links. The IMS T225 is pin compatible with the IMS T222 and is a direct replacement in many applications. The IMS T225 can directly access a linear address space of 64 Kbytes. The 16 bit wide non-multiplexed external memory interface provides a data rate of up to 2 bytes every 100 nanoseconds (20 Mbytes/sec) for a 20 MHz device.

System Services include processor reset and bootstrap control, together with facilities for error analysis.

The INMOS communication links allow networks of transputers to be constructed by direct point to point connections with no external logic. The links support the standard operating speed of 10 Mbits/sec, but also

1 Introduction 455

operate at 5 or 20 Mbits/sec. The links have been improved over those of the IMS T222 and fully support overlapped acknowledge; each IMS T225 link can transfer data bi-directionally at up to 2.4 Mbytes/sec. The link speed settings are the same as those on the IMS T800 (see page 241).

The IMS T225 instruction set contains a number of instructions to facilitate the implementation of breakpoints. For further information concerning breakpointing, refer to *Support for debugging/breakpointing in transputers* (technical note 61).

#### 2 Pin designations

Table 2.1 IMS T225 system services

Pin	in/Out	Function
VCC, GND		Power supply and return
CapPlus, CapMinus	ì	External capacitor for internal clock power supply
Clockin	in	Input clock
ProcSpeedSelect0-2	in	Processor speed selectors
Reset	in	System reset
Error	out	Error indicator
Analyse	in	Error analysis
BootFromRom	in	Bootstraps from external ROM or from link
DisableIntRAM	in	Disable internal RAM

Table 2.2 IMS T225 external memory interface

Pin	In/Out	Function
ProcClockOut	out	Processor clock
MemA0-15	out	Sixteen address lines
MemD0-15	in/out	Sixteen data lines
notMemWrB0-1	out	Two byte-addressing write strobes
notMemCE	out	Chip enable
MemBAcc	in	Byte access mode selector
MemWait	in	Memory cycle extender
MemReq	in	Direct memory access request
MemGranted	out	Direct memory access granted

Table 2.3 IMS T225 event

Pin	In/Out	Function	
EventReq	in	Event request	
EventAck	out	Event request acknowledge	

Table 2.4 IMS T225 link

Pin In/Out		Function		
Linkln0-3 in		Four serial data input channels		
LinkOut0-3 out		Four serial data output channels		
LinkSpecial	in	Select non-standard speed as 5 or 20 Mbits/sec		
Link0Special in		Select special speed for Link 0		
Link123Special	in	Select special speed for Links 1,2,3		

Signal names are prefixed by **not** if they are active low, otherwise they are active high. Pinout details for various packages are given on page 459.

#### 3 Instruction set summary

The instruction set of the IMS T225 is the same as that of the IMS T222 with a number of additions. The instructions additional to those of the IMS T222 are listed below.

The load device identity (*Iddevid*) instruction (table 3.4) pushes the device type identity into the A register. Each product is allocated a unique group of numbers for use with the *Iddevid* instruction. The product identity numbers for the IMS T225 are 40 to 49 inclusive.

Table 3.5 contains a number of instructions to facilitate the implementation of breakpoints. These instructions overload the operation of *j0*. Normally *j0* is a no-op which might cause descheduling. *Setj0break* enables the breakpointing facilities and causes *j0* to act as a breakpointing instruction. When breakpointing is enabled, *j0* swaps the current **lptr** and **Wptr** with an **lptr** and **Wptr** stored above MemStart. The breakpoint instruction does not cause descheduling, and preserves the state of the registers. It is possible to single step the processor at machine level using these instructions. Refer to *Support for debugging/breakpointing in transputers* (technical note 61) for more detailed information regarding debugger support.

Table 3.1 IMS T225 arithmetic/logical operation codes

Operation Code	Memory Code	Mnemonic	Processor Cycles	Name	D E
08	F8	prod	b+4	product for positive register A	
			m+5	product for negative register A	

Table 3.2 IMS T225 general operation codes

Operation Code	Memory Code	Mnemonic	Processor Cycles	Name	D E
5A	25FA	dup	1	duplicate top of stack	
79	27F9	pop	1	pop processor stack	j

Table 3.3 IMS T225 CRC and bit operation codes

Operation Code	Memory Code	Mnemonic	Processor Cycles	Name	D E
74	27F4	crcword	35	calculate crc on word	
75	27F5	crcbyte	11	calculate crc on byte	
76	27F6	bitcnt	b+2	count bits set in word	
77	27F7	bitrevword	36	reverse bits in word	ı
78	27F8	bitrevnbits	n+4	reverse bottom n bits in word	

Table 3.4 IMS T225 processor initialisation operation codes

Operation Code	Memory Code	Mnemonic	Processor Cycles	Name	D E
17C	2127FC	lddevid	1	load device identity	
7E	27FE	Idmemstartval	1	load value of memstart address	

Table 3.5 IMS T225 debugger support codes

Operation Code	Memory Code	Mnemonic	Processor Cycles	Name	DE
0	00	jump 0	3	jump 0 (break not enabled)	D
			11	jump 0 (break enabled, high priority)	
			13	jump 0 (break enabled, low priority)	1
B1	2BF1	break	9	break (high priority)	
			11	break (low priority)	1
B2	2BF2	clrj0break	1	clear jump 0 break enable flag	
B3	2BF3	setj0break	1	set jump 0 break enable flag	i
B4	2BF4	testj0break	2	test jump 0 break enable flag set	
7A	27FA	timerdisableh	1	disable high priority timer interrupt	
7B	27FB	timerdisablel	1 disable low priority timer interrupt		1
7C	27FC	timerenableh	6 enable high priority timer interrupt		
7D	27FD	timerenablel	6	enable low priority timer interrupt	

#### 4 Package specifications

#### 4.1 68 pin grid array package

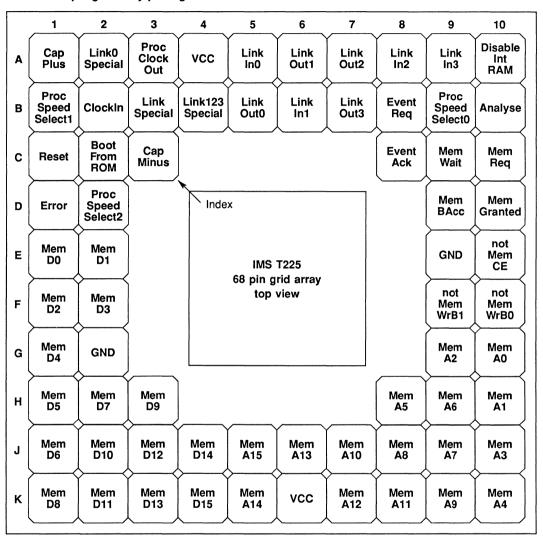


Figure 4.1 IMS T225 68 pin grid array package pinout

Details of the 68 pin grid array package dimensions are given on page 449

#### 4.2 68 pin PLCC J-bend package

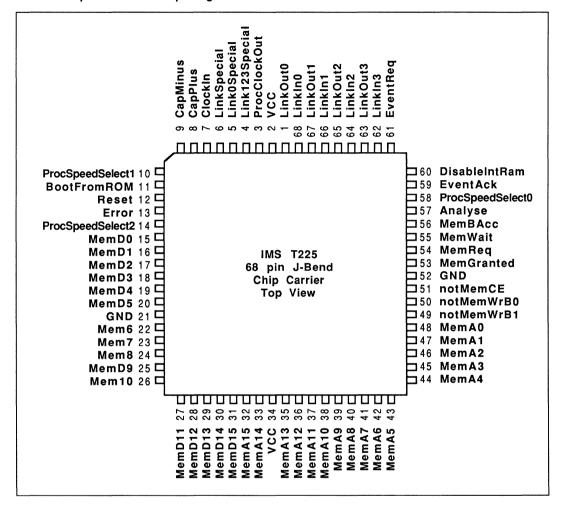


Figure 4.2 IMS T225 68 pin PLCC J-bend package pinout

Details of the 68 pin PLCC J-bend package dimensions are given on page 451.

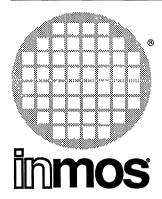
#### 5 Ordering

This section indicates the designation of speed and package selections for the various devices. Speed of **ClockIn** is 5 MHz for all parts. Transputer processor cycle time is nominal; it can be calculated more exactly using the phase lock loop factor **PLLx**, as detailed in the external memory section.

For availability contact local INMOS sales office or authorised distributor.

Table 5.1 IMS T225 ordering details

INMOS designation	Processor clock speed	Processor cycle time	PLLx	Package
IMS T225-G17S	17.5 MHz	57 ns	3.5	Ceramic Pin Grid
IMS T225-G20S	20.0 MHz	50 ns	4.0	Ceramic Pin Grid
IMS T225-G25S	25.0 MHz	40 ns	5.0	Ceramic Pin Grid
IMS T225-G30S	30.0 MHz	33 ns	6.0	Ceramic Pin Grid
IMS T225-J17S	17.5 MHz	57 ns	3.5	Plastic J-Bend
IMS T225-J20S	20.0 MHz	50 ns	4.0	Plastic J-Bend

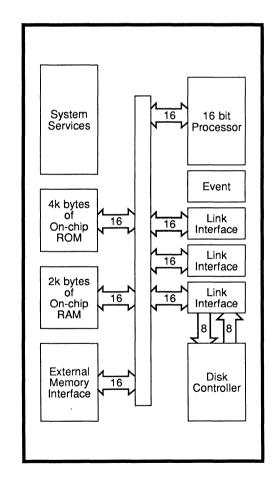


### IMS M212 disk processor

**Product Preview** 

#### **FEATURES**

ST506/ST412, SA400/450 compatible interface Full disk interface logic on chip Minimum of external components required On-chip 16 bit processor 2 Kbytes on-chip RAM 4 Kbytes on-chip ROM disk control software External memory interface Hardware CRC/ECC generator Two bi-directional 8 bit data ports Two 10/20 Mbits/sec INMOS serial links External event interrupt Variable wait states for slow memory Internal timers Support for run-time error diagnostics Bootstraps from ROM, link or disk Single 5 MHz processor clock input Power dissipation less than 1 Watt



#### 1 Introduction

The IMS M212 peripheral processor is an intelligent peripheral controller of the INMOS transputer family, configured for connection to soft sectored winchester and floppy disk drives. It satisfies the demand for increasing intelligence in peripheral controllers and maintains a high degree of flexibility, allowing designers to modify the controller function without altering the hardware.

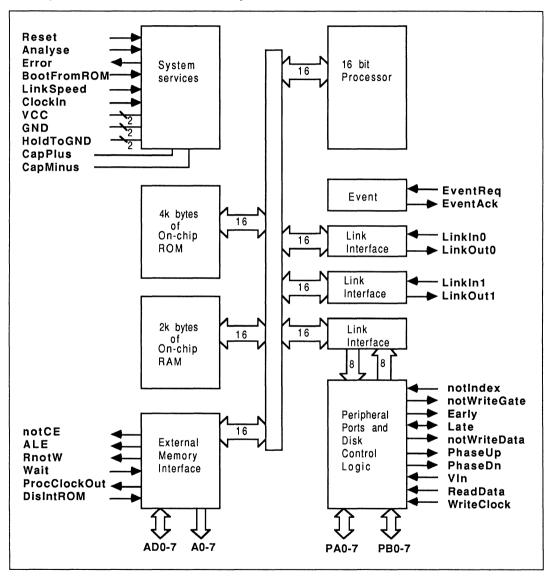


Figure 1.1 IMS M212 block diagram

The disk control function has been designed to provide easy connection, with minimal external hardware, to a standard winchester and/or floppy disk interface. Two byte-wide programmable bidirectional ports are provided to control and monitor disk functions such as head position, drive selection and disk status. A dedicated port is provided for serial data interfaces and critical timing signals.

1 Introduction 465

The IMS M212 is programmed as a normal transputer, permitting extremely powerful peripheral control facilities to be built into the device and thus reducing the load on the traditional central processor of a computer. Full details are given in the IMS M212 Disk Processor Product Data manual.

#### 1.1 IMS M212 peripheral processor

#### 1.1.1 Central processor

At the heart of the IMS M212 is a 16 bit processor which is compatible with the transputer family. Its design achieves compact programs, efficient high level language implementation and provides direct support for the OCCAM model of concurrency. The processor shares its time between any number of concurrent processes. A process waiting for communication or a timer does not consume any processor time. Two levels of process priority enable fast interrupt response to be achieved.

The IMS M212 has been designed so that the on-chip processor performs as many functions as possible, providing flexible operation and minimising on-chip disk-specific hardware.

#### 1.1.2 Peripheral interface

The two 8 bit data ports **PA0-7** and **PB0-7** are controlled by the processor via a pair of channels. This allows the programmer to modify the function of these ports in order to implement a wide variety of applications.

The peripheral interface includes data output registers and TTL compatible input ports, as well as facilities for defining the direction of the pins on a bit-selectable basis. The interface contains logic to detect a change of state on the input pins and to store this change for interrogation by the program.

In addition to this, the external memory interface can support memory mapped peripherals on its byte-wide data bus. An event pin is also provided, so that peripherals can request attention.

#### 1.1.3 Disk controller

The disk interface provides a simple interconnection to ST506/ST412 and SA400/SA450 compatible disk drives via ten dedicated disk control lines and the two general purpose 8 bit bidirectional data ports **PA0-7** and **PB0-7**. Although the on-chip disk control hardware handles much of the specialised data conversion, as many disk operations as possible are controlled by the processor, using sequences of control and data information.

The processor can program and interrogate all the registers controlling the disk functions and data ports, and thereby control the external interface lines. As a result of this versatility, the IMS M212 can also be used in applications other than disk control ones.

A versatile hardware 32 bit Error Correcting Codes (ECC) and 16 bit Cyclic Redundancy Codes (CRC) generator is included to check data integrity. ECC's allow certain classes of errors to be corrected as well as detected, whilst CRC's only allow detection.

When writing data to the disk the hardware serialises the data and encodes it into a Frequency Modulated (FM) or Modified Frequency Modulated (MFM) data stream. Any necessary precompensation is performed internally before outputting the data together with the necessary control signals. Any necessary modification of the data, for instance writing the Address Marks (AM) or inserting the CRC/ECC bytes, is automatically performed by the hardware.

When reading data from disk the raw read data is input and the function known as data separation is performed internally. The hardware examines the data stream for an Address Mark to achieve byte synchronisation and then searches for the desired sector information. When the required data is located it is decoded and a serial to parallel conversion is performed before the data is transferred to the processor.

#### 1.1.4 Links

The IMS M212 uses a DMA block transfer mechanism to transfer messages between memory and another transputer product via the INMOS links. The link interfaces and the processor all operate concurrently, allowing processing to continue while data is being transferred on all of the links.

The host interface of the IMS M212 is via two INMOS standard links, providing simple connection to any transputer based system or, via a link adaptor, to a conventional microprocessor system. Link speeds of 10 Mbits/sec and 20 Mbits/sec are available, making the device compatible with all other INMOS transputer products.

The on-chip disk control logic is controlled by the processor, using simple command sequences, via two channels which appear to the processor as a normal pair of hardware channels.

#### 1.1.5 Memory system

The 2 Kbytes of on-chip static RAM can be used for program or data storage, as a sector buffer or to store parameter and format information. It can be extended off chip, via the external memory interface, to provide a total of 64 Kbytes. Internal and external memory appear as a single contiguous address space.

Software contained in 4 Kbytes of internal ROM enables the IMS M212 to be used as a stand alone disk processor. The ROM can be disabled to free the address space for external memory.

#### 1.1.6 Error handling

High level language execution is made secure with array bounds checking, arithmetic overflow detection etc. A flag is set when an error is detected, and the error can be handled internally by software or externally by sensing the error pin. System state is preserved for subsequent analysis.

#### 2 Operation

The IMS M212 can be used in two modes: Mode 1, which uses the software in the internal ROM, and Mode 2, which relies upon custom designed software.

#### 2.1 Mode 1

Mode 1 operation uses code in the on-chip ROM to control the disk controller hardware, and little knowledge of the hardware is required to implement winchester and floppy disk drivers. The programming interface to all drive types is identical, and there is sufficient flexibility to allow a wide variety of formats and drive types to be used.

Both ST506/412 compatible winchester and SA400/450 compatible floppy drives are supported in standard double density formats; this includes common 5.25 and 3.5 inch drives. Up to 4096 cylinders are allowed. Floppy drives can have up to 8 heads and winchesters up to 16 heads. There can be between 1 and 256 sectors per track, with sector sizes of 128 to 16384 bytes in powers of 2. Drives with or without 'seek complete' and 'ready' lines are supported, and step rates can be from 64  $\mu$ s to 16 ms. A range of non-standard formats can also be set up for user-specific requirements.

As with transputers, the IMS M212 can be bootstrapped from ROM or via a link. In addition, the Mode 1 monitor process also provides a facility whereby the disk processor can bootstrap itself with code read from a disk; this code runs instead of the Mode 1 process. Another option sends a standard bootstrap message, read from a disk, out of link 0; the Mode 1 process then continues as normal. It is also possible in Mode 1 to send a command, at any time, to bootstrap from code in the sector buffer.

General workspace for Mode 1 is contained in on-chip RAM, which also provides 1280 bytes of sector buffer. Contiguous external RAM immediately past the internal RAM will automatically be used to extend the size of the sector buffer. As many sectors as will fit into the sector buffer can be stored in it at the same time.

In Mode 1 a separate data area, in on-chip RAM, contains all the required control information (parameters) for each of the four possible drives. Parameters may be read from or written to via the links, and contain such information as the capacity of the disk, current position of the heads, desired sector for reading or writing, drive type, timing details etc.

Command and data bytes are accepted down either of the IMS M212 links; an interlock system prevents conflict between commands received on both links simultaneously. Any results are returned on the link which received the command. Available commands are

EndOfSequence ReadBuffer Restore PollDrives Initialise WriteBuffer Seek FormatTrack ReadParameter ReadSector SelectHead Boot WriteParameter WriteSector SelectDrive

Disk access commands implicitly select the drive, perform a seek and select the head. If an ECC or CRC error is found when reading a sector, a programmable number of automatic retries are performed and a subsequent correction attempted if possible. Mode 1 supports two of the four IMS M212 ECC/CRC modes - ECC and CRC. Either CRC or ECC can be specified in either of the ID or Data fields, making it possible to have floppies with correctable Data fields.

All appropriate parameters are checked to ensure that, for example, an attempt is not made to access a non-existent sector, relieving the host processor of such checking. Another feature which reduces the load on the host processor is the logical sector mode, in which all the sectors are specified as a single linear address space rather than physical cylinder/head/sector.

The logical address can also be auto-incremented if desired, as can the sector buffer. This allows a number of consecutive sectors to be read from or written to the disk with little overhead. As a *sticky status* checking technique is used, the status only has to be checked once at the end of a stream of commands; if an error occurred then reading and writing is inhibited, so that the logical address can be inspected to find where the error occurred.

#### 2.2 Mode 2

In Mode 2 operation the internal ROM is bypassed, allowing the device to utilise user-defined software. This software can be held in external ROM, bootstrapped from a floppy or winchester disk, or loaded from the host processor via a link into internal or external RAM.

In this mode the user services the disk control hardware via a pair of on-chip high bandwidth channels. Using these channels the processor has access to the 49 registers which control the operation of the disk controller. Sequences of control codes and data bytes are sent by the processor to the disk controller logic via one of the hardware channels and data returned to the IMS M212 processor via the other. Each control code is a single byte, and may be followed by one or more data bytes.

In Mode 2 the designer can define new commands which are more complex than otherwise available. Examples include a *Format Disk* command as an extension to the *Format Track*; an application-specific directory structure; a software interface to optimise a particular file structure. Mode 2 also allows the user to optimise data transfer; thus, data could be read from a disk with no interleave, or data transfers could be re-ordered to minimise head movement. Disk searches can be arranged such that data transfer back to the host is minimised, as data comparisons can be performed by the on-chip processor.

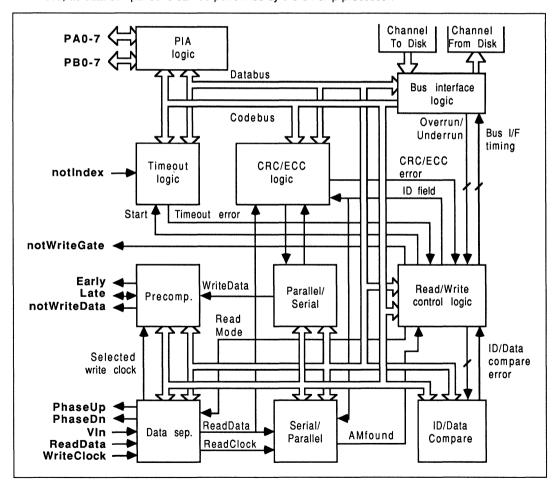


Figure 2.1 Disk controller interface

#### 3 Applications

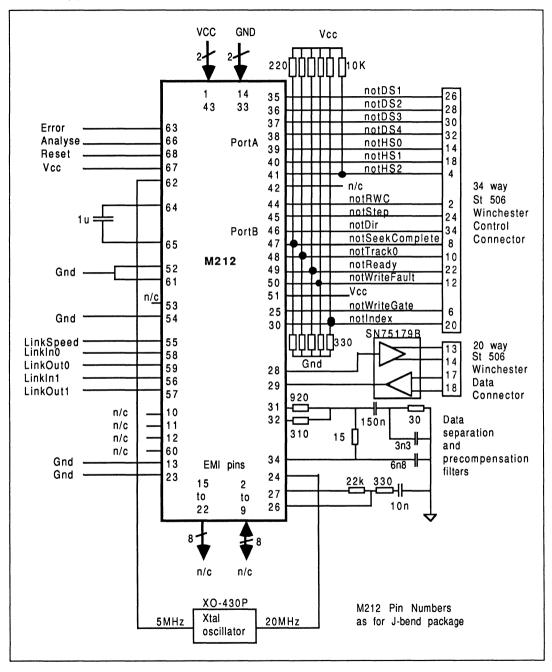


Figure 3.1 Winchester disk controller

The IMS M212 can interface to a floppy or winchester disk with very little external circuitry when used in Mode 1 or if a program is bootstrapped from a link. A typical arrangement is shown in figure 3.1. Note the absence of any control port buffers; this is possible provided the drive characteristics are not infringed.

Additional external memory can easily be added to the IMS M212. In both Modes 1 and 2, external RAM can be added for extra sector storage, whilst in Mode 2 extra RAM or ROM can be provided for program storage.

With the addition of control buffers and suitable clocks, a single IMS M212 can interface to both floppy and winchester drives. Link adaptors provide a means of interfacing to conventional microprocessors.

The IMS B005 evaluation board is an example of an application with control for both types of drive. The board also has a fully populated memory interface.

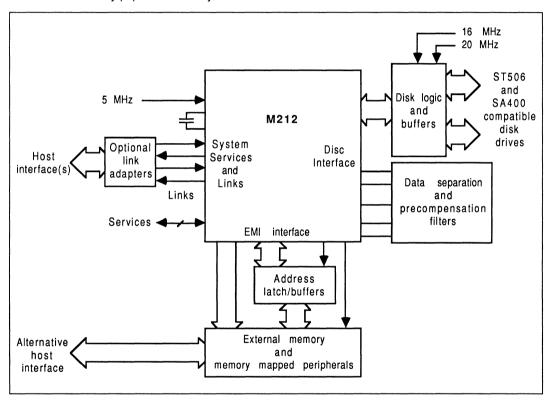


Figure 3.2 Enhanced disk controller interface

The IMS M212 can interface with both floppy and winchester disk drives, and the data rate to and from the disk can be selected by software. As a result the device is suitable for interfacing to the new generation of floppy disk drives which use vertical recording. These drives have an increased data rate of 1 Mbit/sec, and quadruple the capacity of existing floppy disk drives to 4 Mbytes. A single IMS M212 can be used to control a mixture of standard floppy drives, winchester drives and the new high speed high capacity drives. This eases compatibility and portability problems, and provides a simple upgrade path from standard floppies to high capacity floppies to winchesters.

The IMS M212 provides a very simple and compact disk controller solution, making it very easy to replace a single large disk drive with an array of IMS M212's, each controlling a single smaller disk drive. This has several advantages: cheaper drives can be used; overall available disk bandwidth is increased; local processing is provided by a high performance processor at each disk node; fault tolerant operation. The latter can be achieved by holding duplicated data on several drives. This prevents the whole system from stopping, as would be the case if the single large drive failed.

These advantages are particularly applicable when transputers are connected in arrays to provide high performance concurrent systems (figure 3.3). The IMS M212's can be directly connected to the array via INMOS links and the spare link used to communicate with the adjacent IMS M212 to provide the fault tolerant operation.

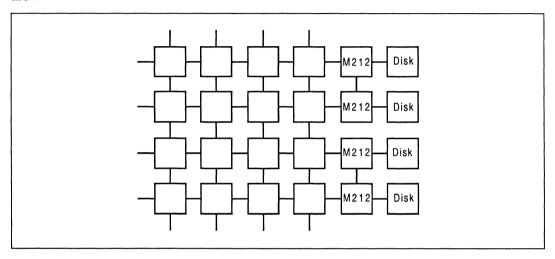


Figure 3.3 Transputer network with disk processors

A high performance processor allows many operations to be performed locally to the disk. This not only frees the host processor for other work but also removes the need for large amounts of data to be needlessly transferred to the host. Operations which can be performed by the IMS M212 include: file management with directory management and pre-reading; data manipulation such as compression/de-compression and encryption/de-cryption; data search such as database key searching; performance optimisation such as head scheduling and cacheing.

The IMS M212 external memory interface can be used to connect to memory mapped peripherals. One application of this is interfacing to a SCSI bus controller, permitting direct connection to the SCSI bus in a low part count system. The processor is used to control the SCSI bus controller and implements the required command interface, as well as controlling the disk or other peripheral.

This arrangement allows floppy and winchester disks to be simply connected to a SCSI bus. Because the command interface is controlled by a process running in the IMS M212, any future command upgrades can easily be incorporated.

The design can be used both as a target and an initiator interface, again controlled by the process running in the IMS M212. It provides a means of implementing a link to SCSI interface, as well as a SCSI controlled disk.

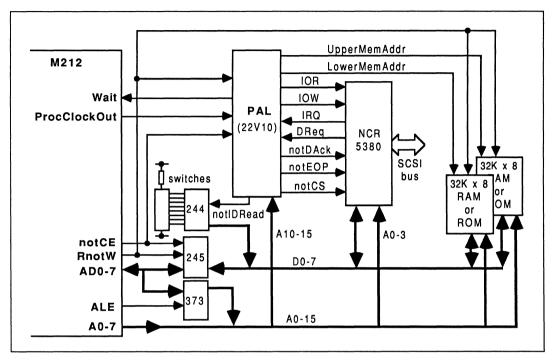


Figure 3.4 SCSI interface

#### 4 Package specifications

#### 4.1 68 pin grid array package

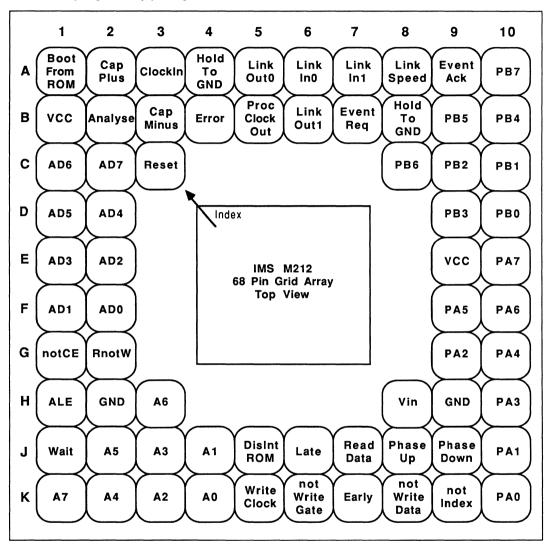


Figure 4.1 IMS M212 68 pin grid array package pinout

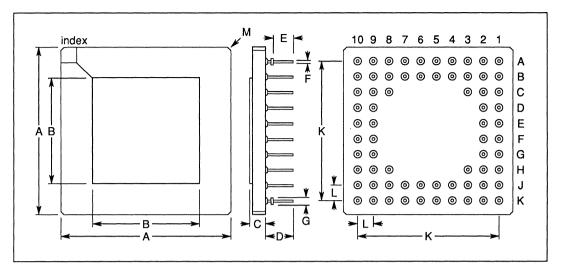


Figure 4.2 68 pin grid array package dimensions

Table 4.1	68	nia	arid	array	package	dimensions

	Millimetres		Inc	hes	
DIM	NOM	TOL	МОМ	TOL	Notes
Α	26.924	±0.254	1.060	±0.010	
В	17.019	±0.127	0.670	±0.008	
C	2.466	±0.279	0.097	±0.011	
D	4.572	±0.127	0.180	±0.005	
E	3.302	±0.127	0.130	±0.005	
F	0.457	±0.051	0.018	$\pm 0.002$	Pin diameter
G	1.270	±0.127	0.050	±0.005	Flange diameter
K	22.860	±0.127	0.900	$\pm 0.005$	
L	2.540	±0.127	0.100	±0.005	
М	0.508		0.020		Chamfer

Package weight is approximately 6.8 grams

Table 4.2 68 pin grid array package junction to ambient thermal resistance

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTE
θJA	At 400 linear ft/min transverse air flow			35	°C/W	

#### 4.2 68 pin PLCC J-bend package

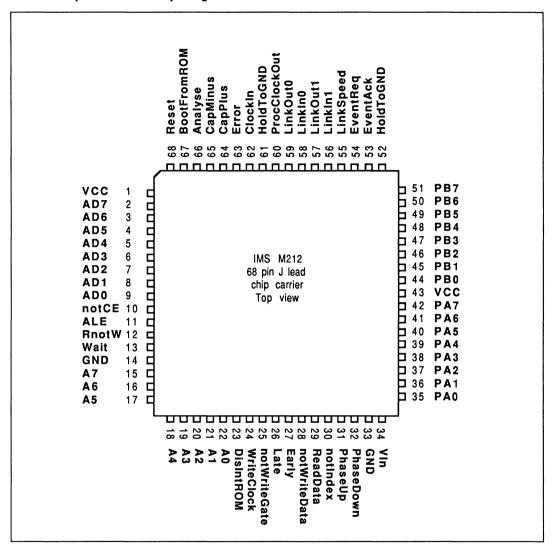


Figure 4.3 IMS M212 68 pin PLCC J-bend package pinout

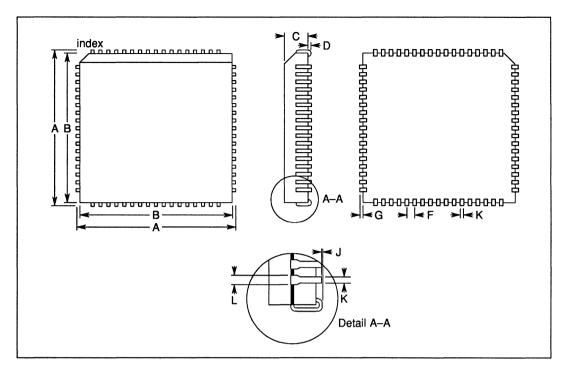


Figure 4.4 68 pin PLCC J-bend package dimensions

Table 4.3 68 pin PLCC J-bend package dimensions

	Millim	netres	Inc	hes	
DIM	NOM	TOL	NOM	TOL	Notes
Α	25.146	±0.127	0.990	±0.005	
В	24.232	±0.127	0.954	±0.005	
C	3.810	±0.127	0.150	±0.005	
D	0.508	±0.127	0.020	±0.005	
F	1.270	±0.127	0.050	±0.005	
G	0.457	±0.127	0.018	±0.005	
J	0.000	±0.051	0.000	±0.002	
K	0.457	±0.127	0.018	±0.005	
L	0.762	±0.127	0.030	±0.005	

Package weight is approximately 5.0 grams

Table 4.4 68 pin PLCC J-bend package junction to ambient thermal resistance

SYMBOL	PARAMETER		NOM	MAX	UNITS	NOTE
θJA	At 400 linear ft/min transverse air flow		35		°C/W	

#### 5 Ordering

This section indicates the designation of speed and package selections for the various devices. Speed of **ClockIn** is 5 MHz for all parts. Transputer processor cycle time is nominal; it can be calculated more exactly using the phase lock loop factor **PLLx**, as detailed in the external memory section.

For availability contact local INMOS sales office or authorised distributor.

Table 5.1 IMS M212 ordering details

INMOS designation	Processor clock speed	Processor cycle time	PLLx	Package
IMS M212-G15S	15 MHz	67 ns	3.0	Ceramic Pin Grid
IMS M212-G20S	20 MHz	50 ns	4.0	Ceramic Pin Grid
IMS M212-J15S	15 MHz	67 ns	3.0	Plastic PLCC J-Bend
IMS M212-J20S	20 MHz	50 ns	4.0	Plastic PLCC J-Bend



# IMS C004 programmable link switch

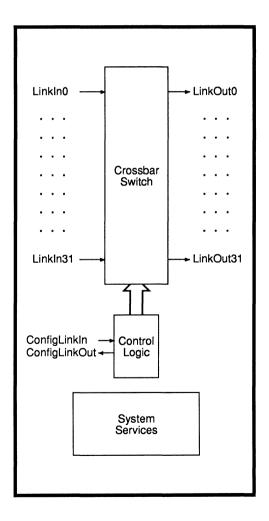
#### **Engineering Data**

#### **FEATURES**

Standard INMOS serial links 32 way crossbar switch Regenerates input signal Cascadable to any depth No loss of signal integrity 10 or 20 Mbits/sec operating speed Separate INMOS configuration link Single +5V ±5% power supply TTL and CMOS compatibility 1W power dissipation Standard 84 pin ceramic PGA MIL-STD-883C device is available

#### **APPLICATIONS**

Programmable crossbar switch Component of larger switch Reconfigurable supercomputers Message routing system High speed multiprocessor systems Telecommunications Robotics Fault tolerant systems Additional links for transputers



#### 1 Introduction

The INMOS communication link is a high speed system interconnect which provides full duplex communication between members of the INMOS transputer family, according to the INMOS serial link protocol. The IMS C004, a member of this family, is a transparent programmable link switch designed to provide a full crossbar switch between 32 link inputs and 32 link outputs.

The IMS C004 will switch links running at either the standard speed of 10 Mbits/sec or at the higher speed of 20 Mbits/sec. It introduces, on average, only a 1.75 bit time delay on the signal. Link switches can be cascaded to any depth without loss of signal integrity and can be used to construct reconfigurable networks of arbitrary size. The switch is programmed via a separate serial link called the *configuration link*.

All INMOS products which use communication links, regardless of device type, support a standard communications frequency of 10 Mbits/sec; most products also support 20 Mbits/sec. Products of different type or performance can, therefore, be interconnected directly and future systems will be able to communicate directly with those of today.

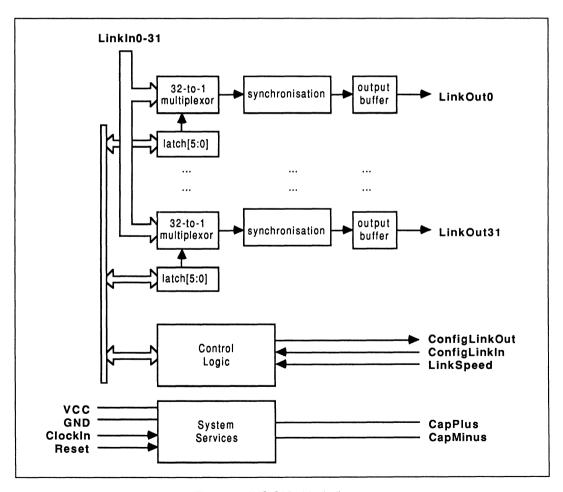


Figure 1.1 IMS C004 block diagram

# 2 Pin designations

Table 2.1 IMS C004 system services

Pin	In/Out	Function
VCC, GND		Power supply and return
CapPlus, CapMinus		External capacitor for internal clock power supply
Clockin	in	Input clock
Reset	in	System reset
DoNotWire		Must not be wired

# Table 2.2 IMS C004 configuration

Pin	In/Out	Function
ConfigLinkIn	in	INMOS configuration link input
ConfigLinkOut	out	INMOS configuration link output

# Table 2.3 IMS C004 link

Pin	In/Out	Function
Linkin0-31	in	INMOS link inputs to the switch
LinkOut0-31	out	INMOS link outputs from the switch
LinkSpeed	in	Link speed selection

Signal names are prefixed by **not** if they are active low, otherwise they are active high. Pinout details for various packages are given on page 498.

# 3 System services

System services include all the necessary logic to start up and maintain the IMS C004.

#### 3.1 Power

Power is supplied to the device via the VCC and GND pins. Several of each are provided to minimise inductance within the package. All supply pins must be connected. The supply must be decoupled close to the chip by at least one 100 nF low inductance (e.g. ceramic) capacitor between VCC and GND. Four layer boards are recommended; if two layer boards are used, extra care should be taken in decoupling.

Input voltages must not exceed specification with respect to VCC and GND, even during power-up and power-down ramping, otherwise *latchup* can occur. CMOS devices can be permanently damaged by excessive periods of latchup.

#### 3.2 CapPlus, CapMinus

The internally derived power supply for internal clocks requires an external low leakage, low inductance  $1\mu F$  capacitor to be connected between **CapPlus** and **CapMinus**. A ceramic capacitor is preferred, with an impedance less than 3 Ohms between 100 KHz and 10 MHz. If a polarised capacitor is used the negative terminal should be connected to **CapMinus**. Total PCB track length should be less than 50 mm. The connections must not touch power supplies or other noise sources.

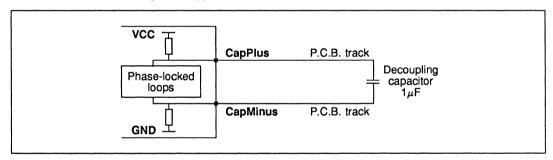


Figure 3.1 Recommended PLL decoupling

#### 3.3 Clockin

Transputer family components use a standard clock frequency, supplied by the user on the **ClockIn** input. The nominal frequency of this clock for all transputer family components is 5 MHz, regardless of device type, transputer word length or processor cycle time. High frequency internal clocks are derived from **ClockIn**, simplifying system design and avoiding problems of distributing high speed clocks externally.

A number of transputer family devices may be connected to a common clock, or may have individual clocks providing each one meets the specified stability criteria. In a multi-clock system the relative phasing of **ClockIn** clocks is not important, due to the asynchronous nature of the links. Mark/space ratio is unimportant provided the specified limits of **ClockIn** pulse widths are met.

Oscillator stability is important. Clockin must be derived from a crystal oscillator; RC oscillators are not sufficiently stable. Clockin must not be distributed through a long chain of buffers. Clock edges must be monotonic and remain within the specified voltage and time limits.

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTE
TDCLDCH	ClockIn pulse width low	40			ns	1
TDCHDCL		40			ns	1
TDCLDCL	ClockIn period		200		ns	1,2,4
TDCerror	ClockIn timing error	ł	ì	±0.5	ns	1,3
TDC1DC2	Difference in ClockIn for 2 linked devices	1		400	ppm	1,4
TDCr	ClockIn rise time	1		10	ns	1,5
TDCf	ClockIn fall time	ì		8	ns	1.5

Table 3.1 Input clock

- 1 These paramters are not tested.
- 2 Measured between corresponding points on consecutive falling edges.
- 3 Variation of individual falling edges from their nominal times.
- 4 This value allows the use of 200ppm crystal oscillators for two devices connected together by a link.
- 5 Clock transitions must be monotonic within the range VIH to VIL (table 7.3).

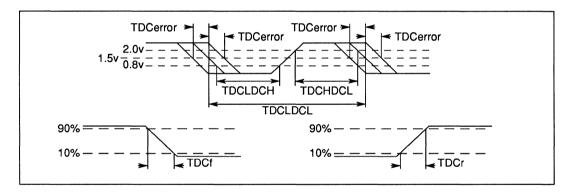


Figure 3.2 ClockIn timing

#### 3.4 Reset

The **Reset** pin can go high with **VCC**, but must at no time exceed the maximum specified voltage for **VIH**. After **VCC** is valid **ClockIn** should be running for a minimum period **TDCVRL** before the end of **Reset**.

Reset initialises the IMS C004 to a state where all link outputs from the switch are disconnected and held low; the control link is then ready to receive a configuration message.

т,	h	ما	2	2	D	^	se	٠
Ιċ	w	ıe	ъ.	_	n	ы	56	ч

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTE
TPVRH	Power valid before Reset	10			ms	
TRHRL	Reset pulse width high	8			Clockin	1
TDCVRL	ClockIn running before Reset end	10			ms	2

- 1 Full periods of Clockin TDCLDCL required.
- 2 At power-on reset.

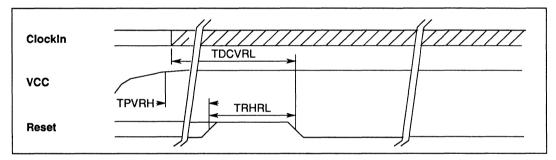


Figure 3.3 Reset Timing

#### 4 Links

INMOS bi-directional serial links provide synchronized communication between INMOS products and with the outside world. Each link comprises an input channel and output channel. A link between two devices is implemented by connecting a link interface on one device to a link interface on the other device. Every byte of data sent on a link is acknowledged on the input of the same link, thus each signal line carries both data and control information.

A receiver can transmit an acknowledge as soon as it starts to receive a data byte. In this way the transmission of an acknowledge can be overlapped with receipt of a data byte to provide continuous transmission of data. This technique is fully compatible with all other INMOS transputer family links.

The quiescent state of a link output is low. Each data byte is transmitted as a high start bit followed by a one bit followed by eight data bits followed by a low stop bit. The least significant bit of data is transmitted first. After transmitting a data byte the sender waits for the acknowledge, which consists of a high start bit followed by a zero bit. The acknowledge signifies that the receiving link is able to receive another byte.

Links are not synchronised with **ClockIn** and are insensitive to its phase. Thus links from independently clocked systems may communicate, providing only that the clocks are nominally identical and within specification.

Links are TTL compatible and intended to be used in electrically quiet environments, between devices on a single printed circuit board or between two boards via a backplane. Direct connection may be made between devices separated by a distance of less than 300 millimetres. For longer distances a matched 100 ohm transmission line should be used with series matching resistors **RM**. When this is done the line delay should be less than 0.4 bit time to ensure that the reflection returns before the next data bit is sent.

Buffers may be used for very long transmissions. If so, their overall propagation delay should be stable within the skew tolerance of the link, although the absolute value of the delay is immaterial.

The IMS C004 links support the standard INMOS communication speed of 10 Mbits/sec. In addition they can be used at 20 Mbits/sec. When the **LinkSpeed** pin is low, all links operate at the standard 10 Mbits/sec; when high they operate at 20 Mbits/sec.

A single IMS C004 inserted between two transputers which fully implement overlapped acknowledges will cause some reduction in data bandwidth, see table 4.2 and figure 4.7.

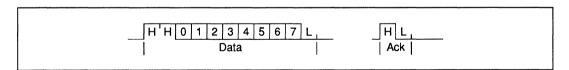


Figure 4.1 IMS C004 link data and acknowledge packets

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTE
TJQr	LinkOut rise time			20	ns	1
TJQf	LinkOut fall time		1	10	ns	1
TJDr	LinkIn rise time		İ	20	ns	1
TJDf	LinkIn fall time	1		20	ns	1
TJQJD	Buffered edge delay	0			ns	
TJBskew	Variation in TJQJD 20 Mbits/s			3	ns	2
	10 Mbits/s		ļ	10	ns	2
CLIZ	LinkIn capacitance @ f=1MHz			7	pF	1
CLL	LinkOut load capacitance			50	pF	
RM	Series resistor for 100Ω transmission line		56		ohms	

Table 4.1 Link

- 1 These paramters are sampled, but are not 100% tested.
- 2 This is the variation in the total delay through buffers, transmission lines, differential receivers etc., caused by such things as short term variation in supply voltages and differences in delays for rising and falling edges.

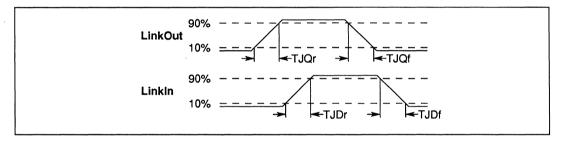


Figure 4.2 IMS C004 link timing

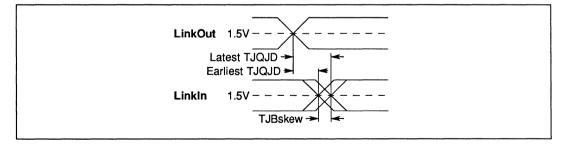


Figure 4.3 IMS C004 buffered link timing

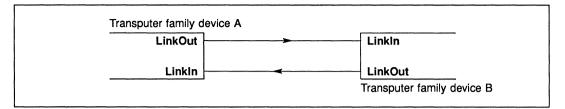


Figure 4.4 IMS C004 Links directly connected

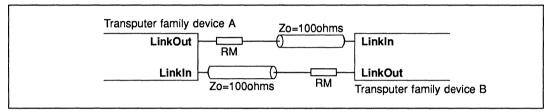


Figure 4.5 IMS C004 Links connected by transmission line

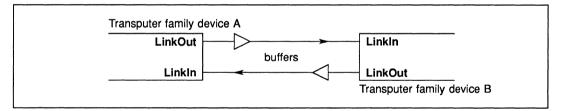


Figure 4.6 IMS C004 Links connected by buffers

	Without C004	With C004	Degradation
Unidirectional	1.7	1.3	25%
Bidirectional	2.3	2.1	10%

Table 4.2 T800 links data transfer rate at 20 Mbits/sec

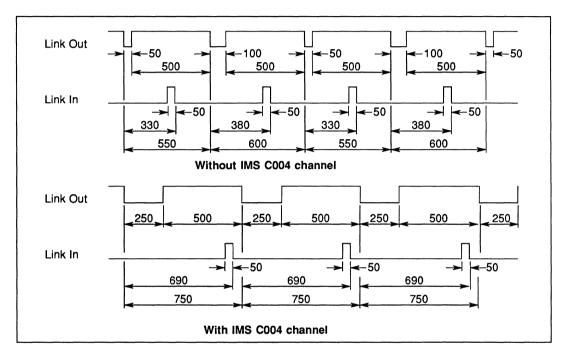


Figure 4.7 IMS C004 link timing

- 1 All values are in ns.
- 2 Timing values shown are for links at 20 Mbits/sec.

# 5 Switch implementation

The IMS C004 is internally organised as a set of thirtytwo 32-to-1 multiplexors. Each multiplexor has associated with it a six bit latch, five bits of which select one input as the source of data for the corresponding output. The sixth bit is used to connect and disconnect the output. These latches can be read and written by messages sent on the configuration link via **ConfigLinkIn** and **ConfigLinkOut**.

The output of each multiplexor is synchronised with an internal high speed clock and regenerated at the output pad. This synchronisation introduces, on average, a 1.75 bit time delay on the signal. As the signal is not electrically degraded in passing through the switch, it is possible to form links through an arbitrary number of link switches.

Each input and output is identified by a number in the range 0 to 31. A configuration message consisting of one, two or three bytes is transmitted on the configuration link. The configuration messages sent to the switch on this link are shown in table 5.1. If an unspecified configuration message is used, the effect of it is undefined.

Table 5.1 IMS C004 configuration messages

Configuration Message	Function
[0] [input] [output]	Connects input to output.
[1] [link1] [link2]	Connects link1 to link2 by connecting the input of link1 to the output of link2 and the input of link2 to the output of link1.
[2] [output]	Enquires which input the <b>output</b> is connected to. The IMS C004 responds with the input. The most signifigant bit of this byte indicates whether the output is connected (bit set high) or disconnected (bit set low).
[3]	This command byte must be sent at the end of every configuration sequence which sets up a connection. The IMS C004 is then ready to accept data on the connected inputs.
[4]	Resets the switch. All outputs are disconnected and held low. This also happens when <b>Reset</b> is applied to the IMS C004.
[5] [output]	Output output is disconnected and held low.
[6] [link1] [link2]	Disconnects the output of link1 and the output of link2.

# 6 Applications

## 6.1 Link switching

The IMS C004 provides full switching capabilities between 32 INMOS links. It can also be used as a component of a larger link switch. For example, three IMS C004's can be connected together to produce a 48 way switch, as shown in figure 6.1. This technique can be extended to the switch shown in figure 6.2.

A fully connected network of 32 INMOS transputers (one in which all four links are used on every transputer) can be completely configured using just four IMS C004's. Figure 6.5 shows the connected transputer network.

In these diagrams each link line shown represents a unidirectional link; i.e. one output to one input. Where a number is also given, that denotes the number of lines.

#### 6.2 Multiple IMS C004 control

Many systems require a number of IMS C004's, each configured via its own configuration link. A simple method of implementing this uses a master IMS C004, as shown in figure 6.3. One of the transputer links is used to configure the master link switch, whilst another transputer link is multiplexed via the master to send configuration messages to any of the other 31 IMS C004 links.

#### 6.3 Bidirectional exchange

Use of the IMS C004 is not restricted to computer configuration applications. The ability to change the switch setting dynamically enables it to be used as a general purpose message router. This may, of course, also find applications in computing with the emergence of the new generation of supercomputers, but a more widespread use may be found as a communication exchange.

In the application shown in figure 6.4, a message into the exchange must be preceded by a destination token *dest*. When this message is passed, the destination token is replaced with a source token so that the receiver knows where the message has come from. The **in.out** device in the diagram and the controller can be implemented easily with a transputer, and the link protocol for establishing communication with these devices can be interfaced with INMOS link adaptors. All messages from **rx[i]** are preceded by the destination output *dest*. On receipt of such a message the **in.out** device requests the controller to connect a bidirectional link path to *dest*. The controller determines what is currently connected to each end of the proposed link. When both ends are free it sets up the IMS C004 and informs both ends of the new link. Note that in this network two channels are placed on each IMS C004 link, one for each direction.

#### 6.4 Bus systems

The IMS C004 can be used in conjunction with the INMOS IMS C011/C012 link adaptors to provide a flexible means of connecting conventional bus based microprocessor systems.

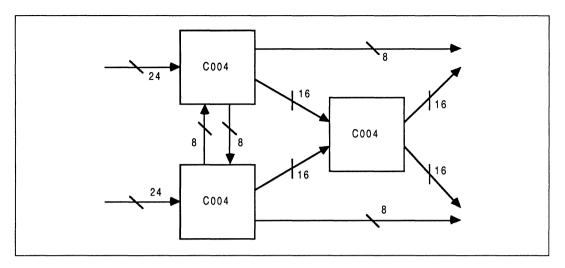


Figure 6.1 48 way link switch

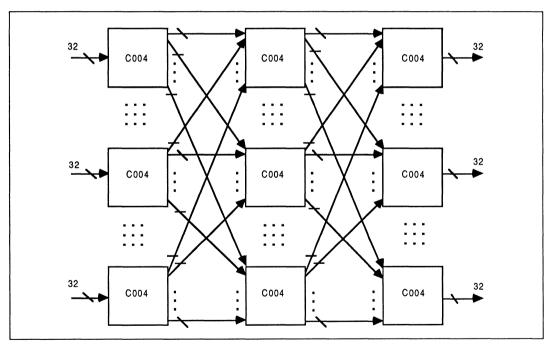


Figure 6.2 Generalised link switch

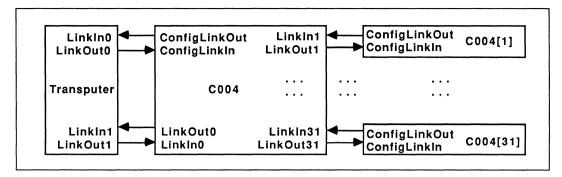


Figure 6.3 Multiple IMS C004 controller

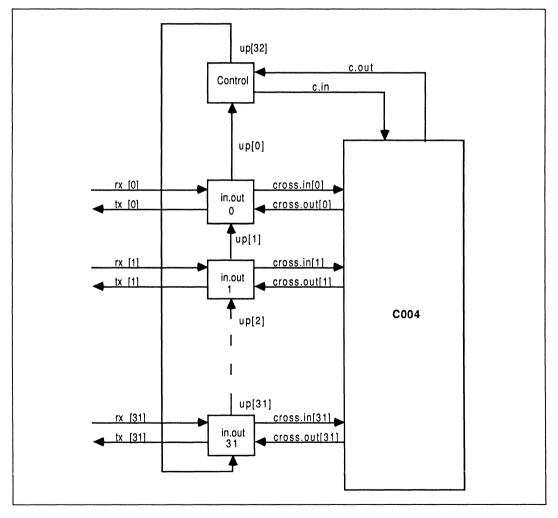


Figure 6.4 32 way bidirectional exchange

Applications 493

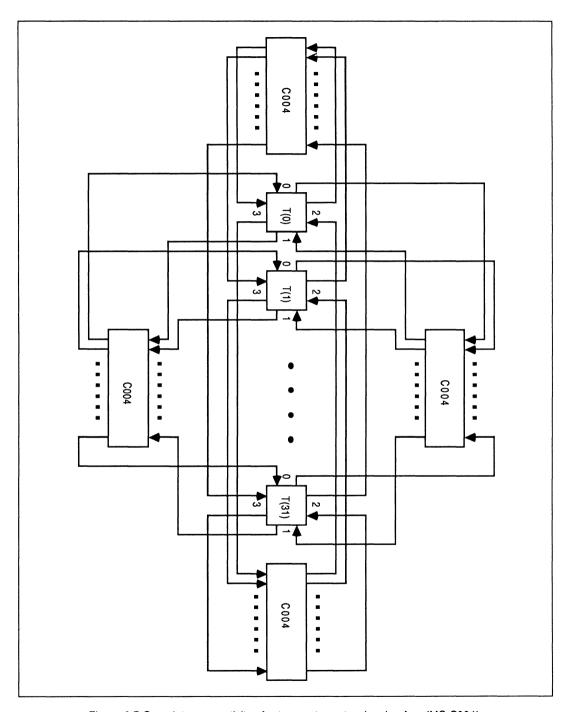


Figure 6.5 Complete connectivity of a transputer network using four IMS C004's

# 7 Electrical specifications

#### 7.1 DC electrical characteristics

Table 7.1 Absolute maximum ratings

SYMBOL	PARAMETER	MIN	MAX	UNITS	NOTE
VCC	DC supply voltage	0	7.0	V	1,2,3
VI, VO	Voltage on input and output pins	-0.5	VCC+0.5	V	1,2,3
11	Input current		±25	mA	4
OSCT	Output short circuit time (one pin)	Į	1	s	2
TS	Storage temperature	-65	150	°C	2
TA	Ambient temperature under bias	-55	125	°C	2
PDmax	Maximum allowable dissipation		2	W	

#### Notes

- 1 All voltages are with respect to GND.
- 2 This is a stress rating only and functional operation of the device at these or any other conditions beyond those indicated in the operating sections of this specification is not implied. Stresses greater than those listed may cause permanent damage to the device. Exposure to absolute maximum rating conditions for extended periods may affect reliability.
- 3 This device contains circuitry to protect the inputs against damage caused by high static voltages or electrical fields. However, it is advised that normal precautions be taken to avoid application of any voltage higher than the absolute maximum rated voltages to this high impedance circuit. Unused inputs should be tied to an appropriate logic level such as VCC or GND.
- 4 The input current applies to any input or output pin and applies when the voltage on the pin is between GND and VCC.

Table 7.2 Operating conditions

SYMBOL	PARAMETER	MIN	MAX	UNITS	NOTE
VCC	DC supply voltage	4.75	5.25	V	1
VI, VO	Input or output voltage	0	VCC	V	1,2
CL	Load capacitance on any pin		60	pF	
TA	Operating temperature range IMS C004-S	0	70	°C	3
TA	Operating temperature range IMS C004-M	-55	125	°C	3

- 1 All voltages are with respect to GND.
- 2 Excursions beyond the supplies are permitted but not recommended; see DC characteristics.
- 3 Air flow rate 400 linear ft/min transverse air flow.

SYMBOL	PARAMETER	MIN	MAX	UNITS	NOTE
VIH	High level input voltage	2.0	VCC+0.5	٧	1,2
VIL	Low level input voltage	-0.5	0.8	V	1,2
11	Input current @ GND <vi<vcc< td=""><td></td><td>±10</td><td>μΑ</td><td>1,2</td></vi<vcc<>		±10	μΑ	1,2
VOH	Output high voltage @ IOH=2mA	VCC-1		V	1,2
VOL	Output low voltage @ IOL=4mA		0.4	V	1,2
IOS	Output short circuit current @ GND <vo<vcc< td=""><td>36</td><td>65</td><td>mA</td><td>1,2,3,6</td></vo<vcc<>	36	65	mA	1,2,3,6
		65	100	mA	1,2,4,6
PD	Power dissipation		1.5	W	2,5
CIN	Input capacitance @ f=1MHz	i	7	pF	6
COZ	Output capacitance @ f=1MHz		10	pF	6

Table 7.3 DC characteristics

- 1 All voltages are with respect to GND.
- 2 Parameters for IMS C004-S measured at 4.75V<VCC<5.25V and 0°C<TA<70°C. Input clock frequency = 5 MHz.
- 3 Current sourced from non-link outputs.
- 4 Current sourced from link outputs.
- 5 Power dissipation varies with output loading and with the number of links active.
- 6 This parameter is sampled and not 100% tested.

# 7.2 Equivalent circuits

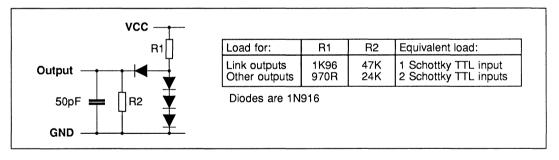


Figure 7.1 Load circuit for AC measurements

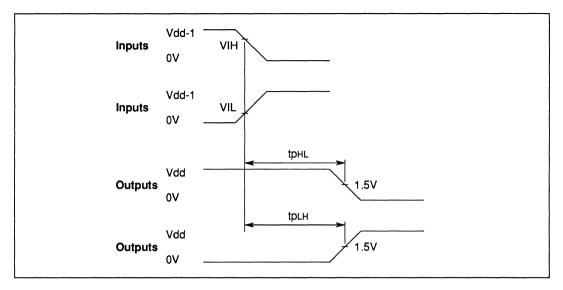


Figure 7.2 AC measurements timing waveforms

# 7.3 AC timing characteristics

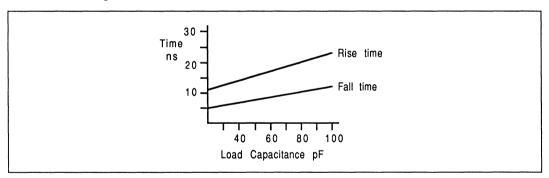


Figure 7.3 Typical link rise/fall times

#### 7.4 Power rating

Internal power dissipation  $P_{INT}$  of transputer and peripheral chips depends on VCC, as shown in figure 7.4.  $P_{INT}$  is substantially independent of temperature.

Total power dissipation  $P_D$  of the chip is

$$P_D = P_{INT} + P_{IO}$$

where  $P_{IO}$  is the power dissipation in the input and output pins; this is application dependent.

Internal working temperature  $T_J$  of the chip is

$$T_J = T_A + \theta J_A * P_D$$

where  $T_A$  is the external ambient temperature in °C and  $\theta J_A$  is the junction-to-ambient thermal resistance in °C/W.  $\theta J_A$  for each package is given in the Packaging Specifications section.

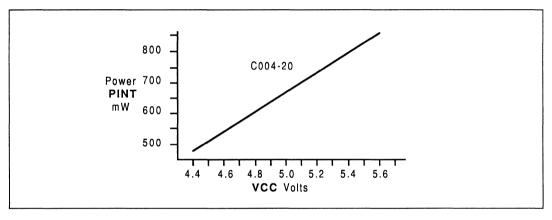


Figure 7.4 IMS C004 internal power dissipation vs VCC

# 8 Package specifications

## 8.1 84 pin grid array package

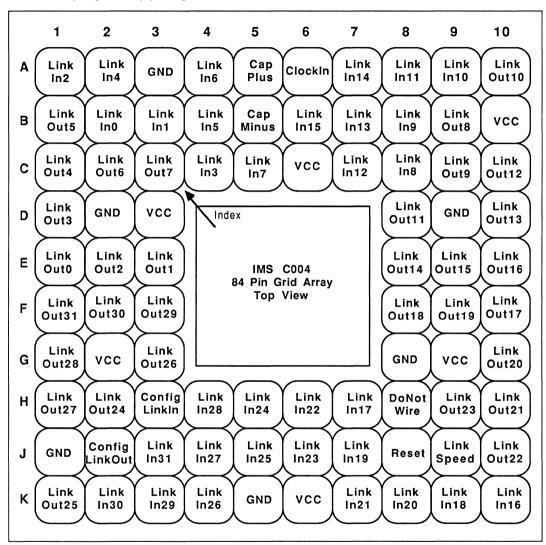


Figure 8.1 IMS C004 84 pin grid array package pinout

Figure 8.2 84 pin grid array package dimensions

Table 8.1 84 pin grid array package dimensions

	Millimetres		Inc	hes	
DIM	МОМ	TOL	NOM	TOL	Notes
Α	26.924	±0.254	1.060	±0.010	
В	17.019	±0.127	0.670	±0.005	
С	2.456	±0.278	0.097 ±0.011		
D	4.572	±0.127	0.180	±0.005	
E	3.302	±0.127	0.130	±0.005	
F	0.457	±0.025	0.018	±0.002	Pin diameter
G	1.143	±0.127	0.045	±0.005	Flange diameter
K	22.860	±0.127	0.900	±0.005	
L	2.540	±0.127	0.100	±0.005	
М	0.508		0.020		Chamfer

Package weight is approximately 7.2 grams

Table 8.2 84 pin grid array package junction to ambient thermal resistance

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTE
$\theta$ JA	At 400 linear ft/min transverse air flow			35	°C/W	

# 8.2 84 lead quad cerpack package

The leads are unformed to allow the user to form them to specific requirements.

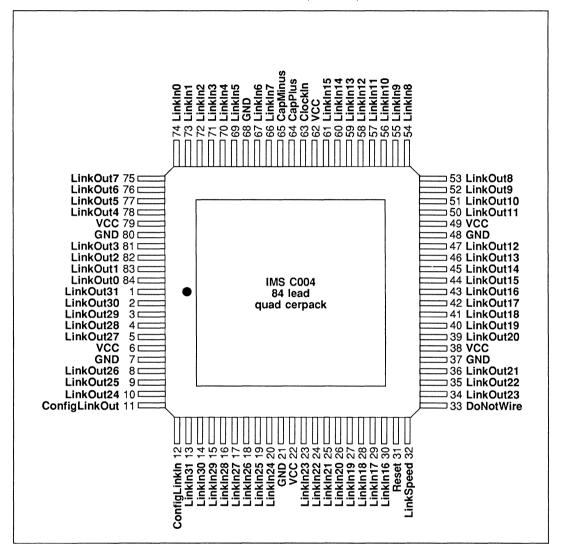


Figure 8.3 IMS C004 84 lead quad cerpack package pinout

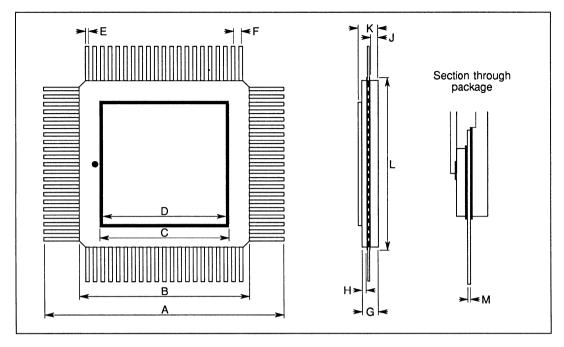


Figure 8.4 84 lead quad cerpack package dimensions

	Millin	netres	Ind	ches	
DIM	NOM	TOL	NOM	TOL	Notes
Α	38.100	±0.508	1.500	±0.020	
В	26.924	±0.305	1.060	±0.012	
С	20.574	±0.203	0.810	±0.008	
D	19.558	±0.254	0.770	±0.010	
E	0.508		0.020		
F	1.270	±0.051	0.050	±0.002	
G	2.489	±0.305	0.098	±0.012	
Н	0.635	±0.076	0.025	±0.003	
J	1.143	±0.102	0.045	±0.004	
К	3.099		0.122		Мах.
L	27.940		1.100		Max.
М	0.178	±0.025	0.007	±0.001	

Table 8.3 84 lead quad cerpack package dimensions

# 9 Ordering

This section indicates the designation of package selections for the IMS C004. Speed of **ClockIn** is 5 MHz for all parts.

For availability contact local INMOS sales office or authorised distributor.

Table 9.1 IMS C004 ordering details

INMOS designation	Package
IMS C004-G20S	Ceramic Pin Grid Array
IMS C004-G20M	Ceramic Pin Grid Array MIL Spec
IMS C004-Q20M	Quad Cerpack MIL Spec



# IMS C011 link adaptor

# **Engineering Data**

#### **FEATURES**

Standard INMOS link protocol
10 or 20 Mbits/sec operating speed
Communicates with INMOS transputers
Converts between serial link and parallel bus
Converts between serial link and parallel device

Two modes of parallel operation:

Mode 1: Peripheral interface
Eight bit parallel input interface
Eight bit parallel output interface
Full handshake on input and output

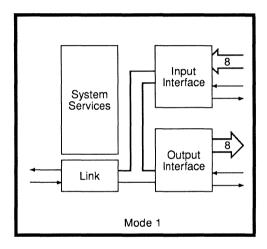
Mode 2: Bus interface
Tristate bidirectional bus interface
Memory mapped registers

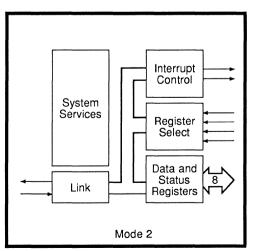
Single +5V ±5% power supply TTL and CMOS compatibility 120mW power dissipation Standard 28 pin 0.6" plastic package MIL-STD-883C device is available

Interrupt capability

### **APPLICATIONS**

Programmable I/O pins for transputer Connecting microprocessors to transputers High speed links between microprocessors Inter-family microprocessor interfacing Interconnecting different speed links





#### 1 Introduction

The INMOS communication link is a high speed system interconnect which provides full duplex communication between members of the INMOS transputer family, according to the INMOS serial link protocol. The IMS C011, a member of this family, provides for full duplex transputer link communication with standard microprocessor and sub-system architectures, by converting bi-directional serial link data into parallel data streams. The extended temperature version of the device complies with MIL-STD-883C.

All INMOS products which use communication links, regardless of device type, support a standard communications frequency of 10 Mbits/sec; most products also support 20 Mbits/sec. Products of different type or performance can, therefore, be interconnected directly and future systems will be able to communicate directly with those of today. The IMS C011 link will run at either the standard speed of 10 Mbits/sec or at the higher speed of 20 Mbits/sec. Data reception is asynchronous, allowing communication to be independent of clock phase.

The link adaptor can be operated in one of two modes. In Mode 1 the IMS C011 converts between a link and two independent fully handshaken byte-wide interfaces, one input and one output. It can be used by a peripheral device to communicate with a transputer, an INMOS peripheral processor or another link adaptor, or it can provide programmable input and output pins for a transputer. Two IMS C011 devices in this mode can be connected back to back via the parallel ports and used as a frequency changer between different speed links.

In Mode 2 the IMS C011 provides an interface between an INMOS serial link and a microprocessor system bus. Status and data registers for both input and output ports can be accessed across the byte-wide bi-directional interface. Two interrupt outputs are provided, one to indicate input data available and one for output buffer empty.

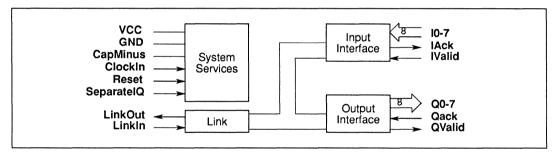


Figure 1.1 IMS C011 Mode 1 block diagram

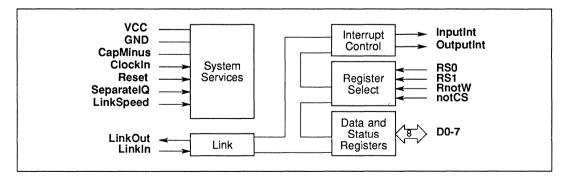


Figure 1.2 IMS C011 Mode 2 block diagram

# 2 Pin designations

Table 2.1 IMS C011 services and link

Pin	In/Out	Function
VCC, GND		Power supply and return
CapMinus		External capacitor for internal clock power supply
Clockin	in	Input clock
Reset	in	System reset
SeparatelQ	in	Select mode and Mode 1 link speed
Linkln	in	Serial data input channel
LinkOut	out	Serial data output channel

Table 2.2 IMS C011 Mode 1 parallel interface

Pin	In/Out	Function
10-7	in	Parallel input bus
IValid	in	Data on I0-7 is valid
IAck	out	Acknowledge 10-7 data received by other link
Q0-7	out	Parallel output bus
QValid	out	Data on Q0-7 is valid
QAck	in	Acknowledge from device: data Q0-7 was read

Table 2.3 IMS C011 Mode 2 parallel interface

Pin	In/Out	Function
D0-7	in/out	Bi-directional data bus
notCS	in	Chip select
RS0-1	in	Register select
RnotW	in	Read/write control signal
InputInt	out	Interrupt on link receive buffer full
Outputint	out	Interrupt on link transmit buffer empty
LinkSpeed	in	Select link speed as 10 or 20 Mbits/sec
HoldToGND		Must be connected to GND
DoNotWire		Must not be wired

Signal names are prefixed by **not** if they are active low, otherwise they are active high. Pinout details for various packages are given on page 524.

# 3 System services

System services include all the necessary logic to start up and maintain the IMS C011.

#### 3.1 Power

Power is supplied to the device via the VCC and GND pins. The supply must be decoupled close to the chip by at least one 100 nF low inductance (e.g. ceramic) capacitor between VCC and GND. Four layer boards are recommended; if two layer boards are used, extra care should be taken in decoupling.

AC noise between VCC and GND must be kept below 200 mV peak to peak at all frequencies above 100 KHz. AC noise between VCC and the ground reference of load capacitances must be kept below 200 mV peak to peak at all frequencies above 30 MHz. Input voltages must not exceed specification with respect to VCC and GND, even during power-up and power-down ramping, otherwise *latchup* can occur. CMOS devices can be permanently damaged by excessive periods of latchup.

# 3.2 CapMinus

The internally derived power supply for internal clocks requires an external low leakage, low inductance  $1\mu F$  capacitor to be connected between VCC and CapMinus. A ceramic capacitor is preferred, with an impedance less than 3 Ohms between 100 KHz and 10 MHz. If a polarised capacitor is used the negative terminal should be connected to CapMinus. Total PCB track length should be less than 50 mm. The positive connection of the capacitor must be connected directly to VCC. Connections must not otherwise touch power supplies or other noise sources.

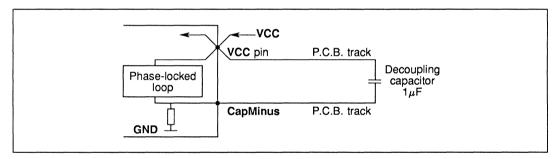


Figure 3.1 Recommended PLL decoupling

#### 3.3 Clockin

Transputer family components use a standard clock frequency, supplied by the user on the **ClockIn** input. The nominal frequency of this clock for all transputer family components is 5 MHz, regardless of device type, transputer word length or processor cycle time. High frequency internal clocks are derived from **ClockIn**, simplifying system design and avoiding problems of distributing high speed clocks externally.

A number of transputer family devices may be connected to a common clock, or may have individual clocks providing each one meets the specified stability criteria. In a multi-clock system the relative phasing of **ClockIn** clocks is not important, due to the asynchronous nature of the links. Mark/space ratio is unimportant provided the specified limits of **ClockIn** pulse widths are met.

Oscillator stability is important. Clockin must be derived from a crystal oscillator; RC oscillators are not sufficiently stable. Clockin must not be distributed through a long chain of buffers. Clock edges must be monotonic and remain within the specified voltage and time limits.

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTE
TDCLDCH	ClockIn pulse width low	40			ns	1
TDCHDCL	ClockIn pulse width high	40		1	ns	1
TDCLDCL	ClockIn period		200	400	ns	1,2,4
TDCerror	ClockIn timing error			±0.5	ns	1,3
TDC1DC2	Difference in ClockIn for 2 linked devices	1	ļ	400	ppm	1,4
TDCr	ClockIn rise time	1		10	ns	1,5
TDCf	ClockIn fall time			8	ns	1,5

Table 3.1 Input clock

- 1 These paramters are not tested.
- 2 Measured between corresponding points on consecutive falling edges.
- 3 Variation of individual falling edges from their nominal times.
- 4 This value allows the use of 200ppm crystal oscillators for two devices connected together by a link.
- 5 Clock transitions must be monotonic within the range VIH to VIL (table 7.3).

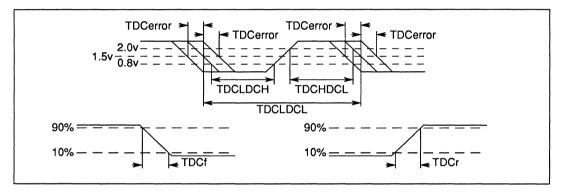


Figure 3.2 ClockIn timing

#### 3.4 SeparatelQ

The IMS C011 link adaptor has two different modes of operation. Mode 1 is basically a link to peripheral adaptor, whilst Mode 2 interfaces between a link and a microprocessor bus system.

Mode 1 can be selected for one of two link speeds by connecting **SeparateIQ** to **VCC** (10 Mbits/sec) or to **Clockin** (20 Mbits/sec).

Mode 2 is selected by connecting **SeparatelQ** to **GND**; in this mode 10 Mbits/sec or 20 Mbits/sec is selected by **LinkSpeed**. Link speeds are specified for a **ClockIn** frequency of 5 MHz.

In order to select the link speed, **SeparateIQ** may be changed dynamically providing the link is in a quiescent state and no input or output is required. **Reset** must be applied subsequent to the selection to initialise the device. If **ClockIn** is gated to achieve this, its skew must be limited to the value **TDCHSIQH** shown in table 3.3. The mode of operation (Mode 1, Mode 2) must not be changed dynamically.

Table 3.2 SeparateIQ mode selection

SeparatelQ	Mode	Link Speed Mbits/sec
VCC	1	10
Clockin	1	20
GND	2	10 or 20

Table 3.3 SeparateIQ

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTE
TDCHSIQH	Skew from ClockIn to ClockIn			20	ns	1

1 Skew between Clockin arriving on the Clockin pin and on the SeparatelQ pin.

#### 3.5 Reset

The **Reset** pin can go high with **VCC**, but must at no time exceed the maximum specified voltage for **VIH**. After **VCC** is valid **ClockIn** should be running for a minimum period **TDCVRL** before the end of **Reset**. **LinkIn** must be held low during **Reset**.

Reset initialises the IMS C011 to the following state: LinkOut is held low; the control outputs (IAck and QValid in Mode 1, InputInt and OutputInt in Mode 2) are held low; interrupts (Mode 2) are disabled; the states of Q0-7 in Mode 1 are unspecified; D0-7 in Mode 2 are high impedance.

Table 3.4 Reset

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTE
TPVRH	Power valid before Reset	10			ms	
TRHRL	Reset pulse width high	8			Clockin	1
TDCVRL	Clockin running before Reset end	10			ms	2

- 1 Full periods of Clockin TDCLDCL required.
- 2 At power-on reset.

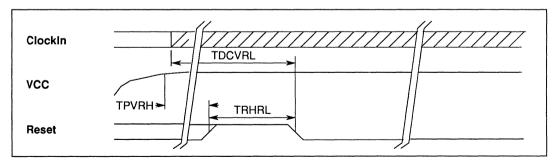


Figure 3.3 Reset Timing

#### 4 Links

INMOS bi-directional serial links provide synchronized communication between INMOS products and with the outside world. Each link comprises an input channel and output channel. A link between two devices is implemented by connecting a link interface on one device to a link interface on the other device. Every byte of data sent on a link is acknowledged on the input of the same link, thus each signal line carries both data and control information.

The quiescent state of a link output is low. Each data byte is transmitted as a high start bit followed by a one bit followed by eight data bits followed by a low stop bit. The least significant bit of data is transmitted first. After transmitting a data byte the sender waits for the acknowledge, which consists of a high start bit followed by a zero bit. The acknowledge signifies both that a process was able to receive the acknowledged data byte and that the receiving link is able to receive another byte.

Links are not synchronised with **ClockIn** and are insensitive to its phase. Thus links from independently clocked systems may communicate, providing only that the clocks are nominally identical and within specification.

Links are TTL compatible and intended to be used in electrically quiet environments, between devices on a single printed circuit board or between two boards via a backplane. Direct connection may be made between devices separated by a distance of less than 300 millimetres. For longer distances a matched 100 ohm transmission line should be used with series matching resistors **RM**. When this is done the line delay should be less than 0.4 bit time to ensure that the reflection returns before the next data bit is sent.

Buffers may be used for very long transmissions. If so, their overall propagation delay should be stable within the skew tolerance of the link, although the absolute value of the delay is immaterial.

The IMS C011 link supports the standard INMOS communication speed of 10 Mbits/sec. In addition it can be used at 20 Mbits/sec. Link speed can be selected in one of two ways. In Mode 1 it is altered by **SeparateIQ** (page 507). In Mode 2 it is selected by **LinkSpeed**; when the **LinkSpeed** pin is low, the link operates at the standard 10 Mbits/sec; when high it operates at 20 Mbits/sec.

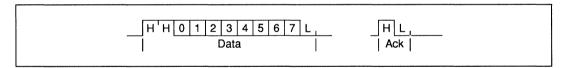


Figure 4.1 IMS C011 link data and acknowledge packets

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTE
TJQr	LinkOut rise time			20	ns	1
TJQf	LinkOut fall time			10	ns	1
TJDr	LinkIn rise time			20	ns	1
TJDf	LinkIn fall time			20	ns	1
TJQJD	Buffered edge delay	0			ns	
TJBskew	Variation in TJQJD 20	Mbits/s		3	ns	2
	10	Mbits/s		10	ns	2
CLIZ	LinkIn capacitance @	f=1MHz	ı	7	pF	1
CLL	LinkOut load capacitance			50	pF	
RM	Series resistor for 100Ω transmission	n line	56		ohms	

Table 4.1 Link

- 1 These paramters are sampled, but are not 100% tested.
- 2 This is the variation in the total delay through buffers, transmission lines, differential receivers etc., caused by such things as short term variation in supply voltages and differences in delays for rising and falling edges.

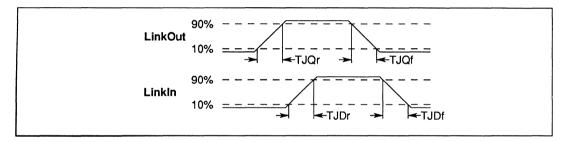


Figure 4.2 IMS C011 link timing

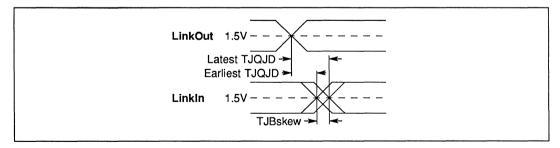


Figure 4.3 IMS C011 buffered link timing

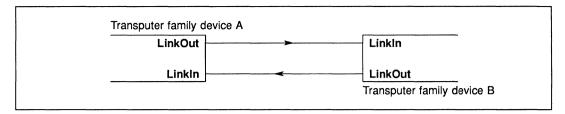


Figure 4.4 IMS C011 Links directly connected

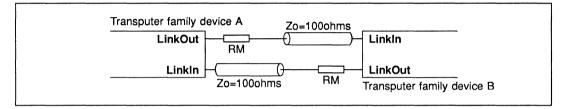


Figure 4.5 IMS C011 Links connected by transmission line

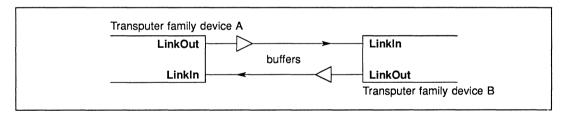


Figure 4.6 IMS C011 Links connected by buffers

# 5 Mode 1 parallel interface

In Mode 1 the IMS C011 link adaptor is configured as a parallel peripheral interface with handshake lines. Communication with a transputer family device is via the serial link. The parallel interface comprises an input port and an output port, both with handshake.

# 5.1 Input port

The eight bit parallel input port I0-7 can be read by a transputer family device via the serial link. IValid and IAck provide a simple two-wire handshake for this port. When data is valid on I0-7, IValid is taken high by the peripheral device to commence the handshake. The link adaptor transmits data presented on I0-7 out through the serial link. When the acknowledge packet is received on the input link, the IMS C011 sets IAck high. To complete the handshake, the peripheral device must return IValid low. The link adaptor will then set IAck low. New data should not be put onto I0-7 until IAck is returned low.

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTE
TldVlvH	Data setup	5			ns	
TIvHLdV	IValid high to link data output	0.8	1	2	bits	1,2
TLaVlaH	Link acknowledge start to IAck high			3	bits	1
TlaHldX	Data hold after IAck high	0	1		ns	
TlaHlvL	IValid hold after IAck high	0			ns	
TivLlaL	IAck hold after IValid low	1 1		4	bits	1
TlaLlvH	Delay before next IValid high	0		ļ	ns	

Table 5.1 Mode 1 parallel data input

- 1 Unit of measurement is one link data bit time; at 10 Mbits/s data link speed, one bit time is nominally 100 ns.
- 2 Maximum time assumes there is no acknowledge packet already on the link. Maximum time with acknowledge on the link is extended by 2 bits.

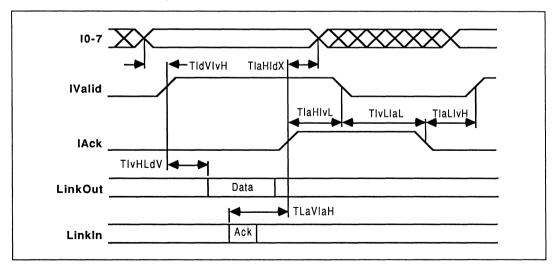


Figure 5.1 IMS C011 Mode 1 parallel data input to link adaptor

#### 5.2 Output port

The eight bit parallel output port **Q0-7** can be controlled by a transputer family device via the serial link. **QValid** and **QAck** provide a simple two-wire handshake for this port.

A data packet received on the input link is presented on **Q0-7**; the link adaptor then takes **QValid** high to initiate the handshake. After reading data from **Q0-7**, the peripheral device sets **QAck** high. The IMS C011 will then send an acknowledge packet out of the serial link to indicate a completed transaction and set **QValid** low to complete the handshake.

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTE
TLdVQvH	Start of link data to QValid	11.5			bits	1
TQdVQvH	Data setup	15			ns	2
TQvHQaH	QAck setup time from QValid high	0			ns	
TQaHQvL	QAck high to QValid low	1.8			bits	1
TQaHLaV	QAck high to Ack on link	0.8		2	bits	1,3
TQvLQaL	QAck hold after QValid low	0			ns	,
TQvLQdX	Data hold	11			bits	1,4

Table 5.2 Mode 1 parallel data output

- 1 Unit of measurement is one link data bit time; at 10 Mbits/s data link speed, one bit time is nominally 100 ns.
- 2 Where an existing data output bit is re-written with the same level there will be no glitch in the output level.
- 3 Maximum time assumes there is no data packet already on the link. Maximum time with data on the link is extended by 11 bits.
- 4 Data output remains valid until overwritten by new data.

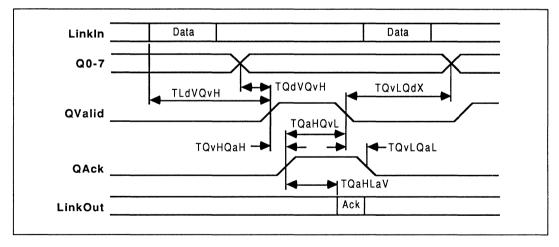


Figure 5.2 IMS C011 Mode 1 parallel data output from link adaptor

## 6 Mode 2 Parallel interface

The IMS C011 provides an interface between a link and a microprocessor style bus. Operation of the link adaptor is controlled through the parallel interface bus lines **D0-7** by reading and writing various registers in the link adaptor. Registers are selected by **RS0-1** and **RnotW**, and the chip enabled with **notCS**.

For convenience of description, the device connected to the parallel side of the link adaptor is presumed to be a microprocessor, although this will not always be the case.

#### 6.1 D0-7

Data is communicated between a microprocessor bus and the link adaptor via the bidirectional bus lines **D0-7**. The bus is high impedance unless the link adaptor chip is selected and the **RnotW** line is high. The bus is used by the microprocessor to access status and data registers.

#### 6.2 notCS

The link adaptor chip is selected when **notCS** is low. Register selectors **RS0-1** and **RnotW** must be valid before **notCS** goes low; **D0-7** must also be valid if writing to the chip (**RnotW** low). Data is read by the link adaptor on the rising edge of **notCS**.

#### 6.3 RnotW

**RnotW**, in conjunction with **notCS**, selects the link adaptor registers for read or write mode. When **RnotW** is high, the contents of an addressed register appear on the data bus **D0-7**; when **RnotW** is low the data on **D0-7** is written into the addressed register. The state of **RnotW** is latched into the link adaptor by **notCS** going low; it may be changed before **notCS** returns high, within the timing restrictions given.

#### 6.4 RS0-1

One of four registers is selected by RS0-1. A register is addressed by setting up RS0-1 and then taking notCS low; the state of RnotW when notCS goes low determines whether the register will be read or written. The state of RS0-1 is latched into the link adaptor by notCS going low; it may be changed before notCS returns high, within the timing restrictions given. The register set comprises a read-only data input register, a write-only data output register and a read/write status register for each.

RS1	RS0	RnotW	Register
0	0	1	Read data
0	0	0	Invalid
0	1	1	Invalid
0	1	0	Write data
1	0	1	Read input status
1	0	0	Write input status
1	1	1	Read output status
1	1	0	Write output status

Table 6.1 IMS C011 Mode 2 register selection

#### 6.4.1 Input Data Register

This register holds the last data packet received from the serial link. It never contains acknowledge packets. It contains valid data only whilst the *data present* flag is set in the input status register. It cannot be assumed to contain valid data after it has been read; a double read may or may not return valid data on the second read. If *data present* is valid on a subsequent read it indicates new data is in the buffer. Writing to this register will have no effect.

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTE
TRSVCSL	Register select setup	5			ns	
TCSLRSX	Register select hold	5			ns	
TRWVCSL	Read/write strobe setup	5			ns	
TCSLRWX	Read/write strobe hold	5			ns	
TCSLCSH	Chip select active	50			ns	
TCSHCSL	Delay before re-assertion of chip select	50		1	ns	

Table 6.2 IMS C011 Mode 2 parallel interface control

Table 6.3 IMS C011Mode 2 parallel interface read

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTE
TLdVIIH	Start of link data to InputInt high			13	bits	1
TCSLIIL	Chip select to InputInt low			30	ns	
TCSLDrX	Chip select to bus active	5	1		ns	
TCSLDrV	Chip select to data valid			40	ns	
TCSHDrZ	Chip select high to bus tristate			25	ns	
TCSHDrX	Data hold after chip select high	5	1		ns	
TCSHLaV	Chip de-select to start of Ack	0.8		2	bits	1,2

- 1 Unit of measurement is one link data bit time; at 10 Mbits/s data link speed, one bit time is nominally 100 ns.
- 2 Maximum time assumes there is no data packet already on the link. Maximum time with data on the link is extended by 11 bits.

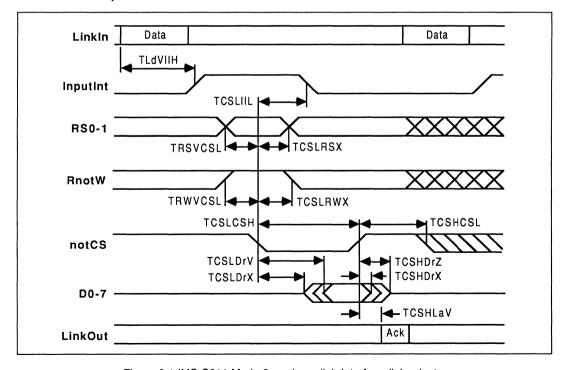


Figure 6.1 IMS C011 Mode 2 read parallel data from link adaptor

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTE
TCSHDwV	Data setup	15			ns	
TCSHDwX	Data hold	5			ns	
TCSLOIL	Chip select to OutputInt low	1		30	ns	
TCSHLdV	Chip select high to start of link data	0.8		2	bits	1,2
TLaVOIH	Start of link Ack to OutputInt high			3	bits	1.3
TLdVOIH	Start of link data to OutputInt high	l		13	bits	1,3

Table 6.4 IMS C011 Mode 2 parallel interface write

- 1 Unit of measurement is one link data bit time; at 10 Mbits/s data link speed, one bit time is nominally 100 ns.
- 2 Maximum time assumes there is no acknowledge packet already on the link. Maximum time with acknowledge on the link is extended by 2 bits.
- 3 Both data transmission and the returned acknowledge must be completed before OutputInt can go high.

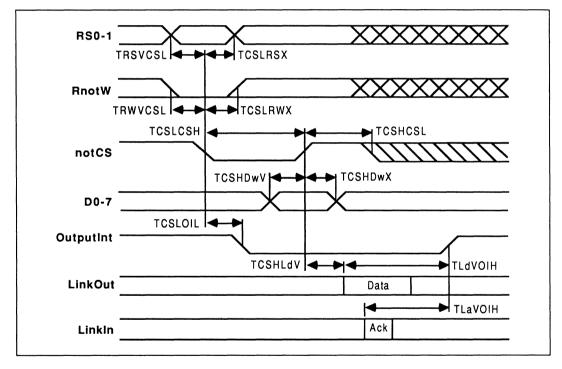


Figure 6.2 IMS C011 Mode 2 write parallel data to link adaptor

#### 6.4.2 Input Status Register

This register contains the *data present* flag and the *interrupt enable* control bit for **InputInt**. The *data present* flag is set to indicate that data in the data input buffer is valid. It is reset low only when the data input buffer is read, or by **Reset**. When writing to this register, the *data present* bit must be written as zero.

The *interrupt enable* bit can be set and reset by writing to the status register with this bit high or low respectively. When the *interrupt enable* and *data present* flags are both high, the **InputInt** output will be high (page 517). Resetting *interrupt enable* will take **InputInt** low; setting it again before reading the data input register will set **InputInt** high again. The *interrupt enable* bit can be read to determine its status.

When writing to this register, bits 2-7 must be written as zero; this ensures that they will be zero when the register is read. Failure to write zeroes to these bits may result in undefined data being returned by these bits during a status register read.

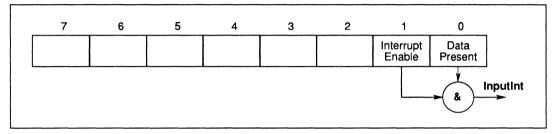


Figure 6.3 IMS C011 input status register

#### 6.4.3 Output Data Register

Data written to this link adaptor register is transmitted out of the serial link as a data packet. Data should only be written to this register when the *output ready* bit in the output status register is high, otherwise data already being transmitted may be corrupted. Reading this register will result in undefined data being read.

#### 6.4.4 Output Status Register

This register contains the *output ready* flag and the *interrupt enable* control bit for **OutputInt**. The *output ready* flag is set to indicate that the data output buffer is empty. It is reset low only when data is written to the data output buffer; it is set high by **Reset**. When writing to this register, the *output ready* bit must be written as zero.

The *interrupt enable* bit can be set and reset by writing to the status register with this bit high or low respectively. When the *interrupt enable* and *output ready* flags are both high, the **OutputInt** output will be high (page 518). Resetting *interrupt enable* will take **OutputInt** low; setting it again whilst the data output register is empty will set **OutputInt** high again. The *interrupt enable* bit can be read to determine it's status.

When writing to this register, bits 2-7 must be written as zero; this ensures that they will be zero when the register is read. Failure to write zeroes to these bits may result in undefined data being returned by these bits during a status register read.

#### 6.5 InputInt

The InputInt output is set high to indicate that a data packet has been received from the serial link. It is inhibited from going high when the *interrupt enable* bit in the input status register is low (page 517). InputInt is reset low when data is read from the input data register (page 514) and by Reset (page 508).

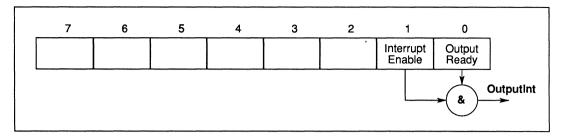


Figure 6.4 IMS C011 output status register

#### 6.6 OutputInt

The **OutputInt** output is set high to indicate that the link is free to receive data from the microprocessor for transmission as a data packet out of the serial link. It is inhibited from going high when the *interrupt enable* bit in the output status register is low (page 517). **OutputInt** is reset low when data is written to the data output register (page 517); it is set low by **Reset** (page 508).

#### 6.7 Data read

A data packet received on the input link sets the *data present* flag in the input status register. If the *interrupt enable* bit in the status register is set, the *InputInt* output pin will be set high. The microprocessor will either respond to the interrupt (if the *interrupt enable* bit is set) or will periodically read the input status register until the *data present* bit is high.

When data is available from the link, the microprocessor reads the data packet from the data input register. This will reset the *data present* flag and cause the link adaptor to transmit an acknowledge packet out of the serial link output. **InputInt** is automatically reset by reading the data input register; it is not necessary to read or write the input status register.

#### 6.8 Data write

When the data output buffer is empty the *output ready* flag in the output status register is set high. If the *interrupt enable* bit in the status register is set, the **OutputInt** output pin will also be set high. The microprocessor will either respond to the interrupt (if the *interrupt enable* bit is set) or will periodically read the output status register until the *output ready* bit is high.

When the *output ready* flag is high, the microprocessor can write data to the data output buffer. This will result in the link adaptor resetting the *output ready* flag and commencing transmission of the data packet out of the serial link. The *output ready* status bit will remain low until an acknowledge packet is received by the input link. This will set the *output ready* flag high; if the *interrupt enable* bit is set, **OutputInt** will also be set high.

#### 7 **Electrical specifications**

#### 7.1 DC electrical characteristics

Table 7.1 Absolute maximum ratings

SYMBOL	PARAMETER :	MIN	MAX	UNITS	NOTE
VCC	DC supply voltage	0	7.0	٧	1,2,3
VI, VO	Voltage on input and output pins	-0.5	VCC+0.5	V	1,2,3
11	Input current		±25	mA	4
OSCT	Output short circuit time (one pin)		1	s	2
TS	Storage temperature	-65	150	°C	2
TA	Ambient temperature under bias	-55	125	°C	2
PDmax	Maximum allowable dissipation		600	mW	

#### Notes

- 1 All voltages are with respect to GND.
- 2 This is a stress rating only and functional operation of the device at these or any other conditions beyond those indicated in the operating sections of this specification is not implied. Stresses greater than those listed may cause permanent damage to the device. Exposure to absolute maximum rating conditions for extended periods may affect reliability.
- 3 This device contains circuitry to protect the inputs against damage caused by high static voltages or electrical fields. However, it is advised that normal precautions be taken to avoid application of any voltage higher than the absolute maximum rated voltages to this high impedance circuit. Unused inputs should be tied to an appropriate logic level such as VCC or GND.
- 4 The input current applies to any input or output pin and applies when the voltage on the pin is between GND and VCC.

SYMBOL PARAMETER MIN MAX UNITS NOTE 4.75 5.25 DC supply voltage VI, VO Input or output voltage 0 VCC ٧ 1,2 рF Load capacitance on any pin 60 ĠС Operating temperature range 0 70 3

Table 7.2 Operating conditions

# TA Notes

VCC

CL

- 1 All voltages are with respect to GND.
- 2 Excursions beyond the supplies are permitted but not recommended; see DC characteristics.
- 3 Air flow rate 400 linear ft/min transverse air flow.

SYMBOL	PARAME	TER	MIN	MAX	UNITS	NOTE
VIH	High level input voltage		2.0	VCC+0.5	V	1,2
VIL	Low level input voltage		-0.5	0.8	l v	1,2
H	Input current	@ GND <vi<vcc< td=""><td></td><td>±10</td><td>μA</td><td>1,2,7</td></vi<vcc<>		±10	μA	1,2,7
	,			±200	μΑ	1,2,8
VOH	Output high voltage	@ IOH=2mA	VCC-1		٠̈٧	1,2
VOL	Output low voltage	@ IOL=4mA		0.4	V	1,2
IOS	Output short circuit current	@ GND <vo<vcc< td=""><td>36</td><td>65</td><td>mA</td><td>1,2,3,6</td></vo<vcc<>	36	65	mA	1,2,3,6
	·		65	100	mA	1,2,4,6
IOZ	Tristate output current	@ GND <vo<vcc< td=""><td></td><td>±10</td><td>μA</td><td>1,2</td></vo<vcc<>		±10	μA	1,2
PD	Power dissipation			120	mW	2,5
CIN	Input capacitance	@ f=1MHz		7	pF	6
COZ	Output capacitance	@ f=1MHz		10	pF	6

Table 7.3 DC characteristics

- 1 All voltages are with respect to GND.
- 2 Parameters for IMS C011-S measured at 4.75V<VCC<5.25V and 0°C<TA<70°C. Input clock frequency = 5 MHz.
- 3 Current sourced from non-link outputs.
- 4 Current sourced from link outputs.
- 5 Power dissipation varies with output loading.
- 6 This parameter is sampled and not 100% tested.
- 7 For inputs other than those in Note 8.
- 8 For pins 2, 3, 5, 6, 7, 9, 11, 13, 15, 16, 25.

#### 7.2 Equivalent circuits

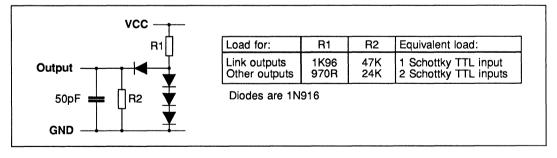


Figure 7.1 Load circuit for AC measurements

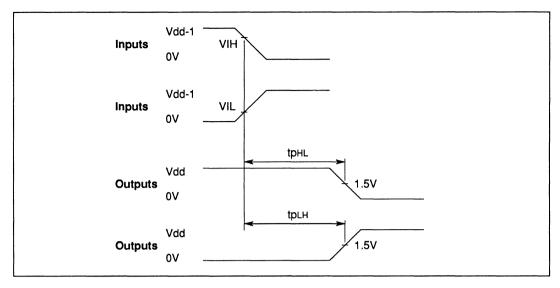


Figure 7.2 AC measurements timing waveforms

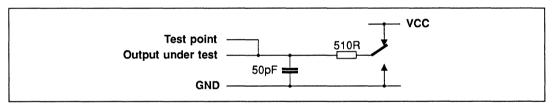


Figure 7.3 Tristate load circuit for AC measurements

#### 7.3 AC timing characteristics

Table 7.4 Input, output edges

SYMBOL	PARAMETER	MIN	MAX	UNITS	NOTE
TDr	Input rising edges	2	20	ns	1,2
TDf	Input falling edges	2	20	ns	1,2
TQr	Output rising edges		25	ns	1
TQf	Output falling edges		15	ns	1
CSLaHZ	Chip select high to tristate		25	ns	
CSLaLZ	Chip select low to tristate		25	ns	

- 1 Non-link pins; see section on links.
- 2 All inputs except Clockin; see section on Clockin.

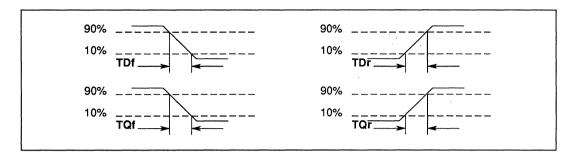


Figure 7.4 IMS C011 input and output edge timing

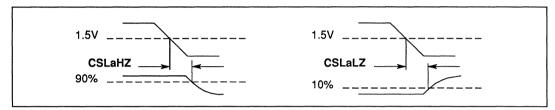


Figure 7.5 IMS C011 tristate timing relative to notCS

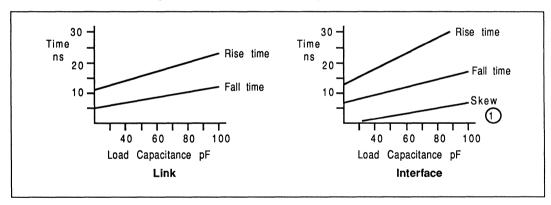


Figure 7.6 Typical rise/fall times

1 Skew is measured between **notCS** with a standard load (2 Schottky TTL inputs and 30pF) and **notCS** with a load of 2 Schottky TTL inputs and varying capacitance.

#### 7.4 Power rating

Internal power dissipation  $P_{INT}$  of transputer and peripheral chips depends on **VCC**, as shown in figure 7.7.  $P_{INT}$  is substantially independent of temperature.

Total power dissipation  $P_D$  of the chip is

$$P_D = P_{INT} + P_{IO}$$

where  $P_{IO}$  is the power dissipation in the input and output pins; this is application dependent.

Internal working temperature  $T_J$  of the chip is

$$T_J = T_A + \theta J_A * P_D$$

where  $T_A$  is the external ambient temperature in °C and  $\theta J_A$  is the junction-to-ambient thermal resistance in °C/W.  $\theta J_A$  for each package is given in the Packaging Specifications section.

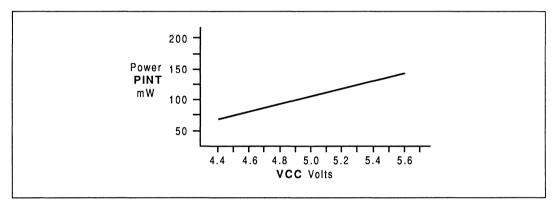


Figure 7.7 IMS C011 internal power dissipation vs VCC

# 8 Package specifications

# 8.1 28 pin plastic dual-in-line package

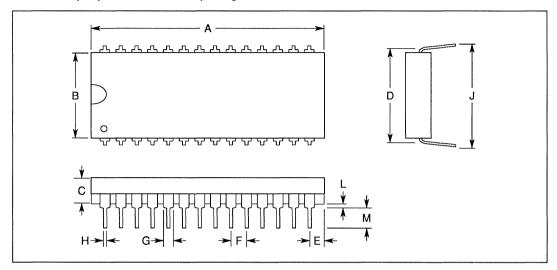


Figure 8.1 28 pin plastic dual-in-line package dimensions

Table 8.1 28 pin plastic dual-in-line package dimensions

	Millimetres		Inc	hes	
DIM	MOM	TOL	NOM	TOL	Notes
Α	36.830	±0.254	1.450	±0.010	
В	13.970	±0.254	0.550	±0.010	
С	4.445	±0.635	0.175	±0.025	
D	15.240	±0.076	0.600	±0.003	
Ε	1.905		0.075		
F	2.540	:	0.100		
G	1.397	±0.254	0.055	±0.010	
Н	0.457		0.018		
J	16.256	±0.508	0.640	±0.020	
L	0.508		0.020		Minimum
М	3.429		0.135		Maximum

Package weight is approximately 4 grams

Table 8.2 28 pin plastic dual-in-line package junction to ambient thermal resistance

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTE
$\theta$ JA	At 400 linear ft/min transverse air flow		110		°C/W	

# 8.2 28 pin ceramic dual-in-line package

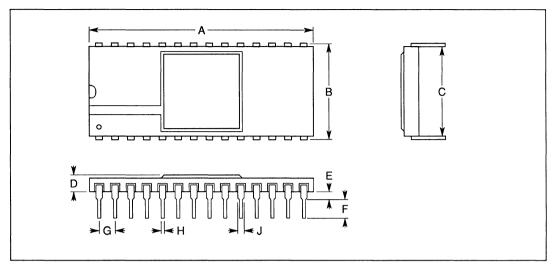


Figure 8.2 28 pin ceramic dual-in-line package dimensions

	Millimetres		Inches		
DIM	NOM	TOL	МОМ	TOL	Notes
Α	35.560	±0.356	1.400	±0.014	
В	15.494	±0.254	0.610	±0.010	
C	14.681	+0.813	0.578	+0.032	
D	2.466	±0.229	0.097	±0.009	
E	1.270	±0.254	0.051	±0.010	
F	3.048		0.120		Minimum
G	2.540		0.100		
Н	0.457	±0.051	0.018	±0.002	
J	1.016	+0.508	0.040	+0.020	

Table 8.3 28 pin ceramic dual-in-line package dimensions

Package weight is approximately 5 grams

Table 8.4 28 pin ceramic dual-in-line package junction to ambient thermal resistance

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTE
$\theta$ JA	At 400 linear ft/min transverse air flow		60		°C/W	

# 8.3 28 pin SOIC package

New product - for availability contact INMOS.

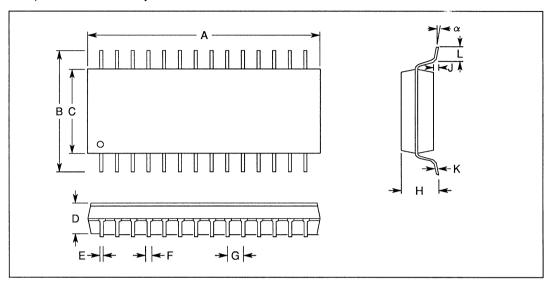


Figure 8.3 28 pin SOIC package dimensions

	Millimetres		Inches		
DIM	MIN	MAX	MIN	MAX	Notes
Α	17.526	18.491	0.697	0.728	1
В	11.506	12.700	0.453	0.500	
С	8.230	8.890	0.324	0.350	1
D	2.337	2.692	0.092	0.106	
E	0.356	0.508	0.014	0.020	
F	0.356	0.610	0.014	0.024	3
G	1.270		0.050		basic
Н		3.048		0.120	
J	0.051	0.356	0.002	0.014	
K	0.152	0.317	0.006	0.012	
L	0.406	1.270	0.016	0.050	
α	0°	8°			

Table 8.5 28 pin SOIC package dimensions

- 1 Overall length and width dimensions do not include mold flash or protrusions. Mold flash or protrusions shall not exceed 0.006 inches per side.
- 2 Formed leads shall be planar with respect to one another within 0.004 inches at seating plane.
- 3 F is to allow for positive dambar protrusion.

#### 8.4 Pinout

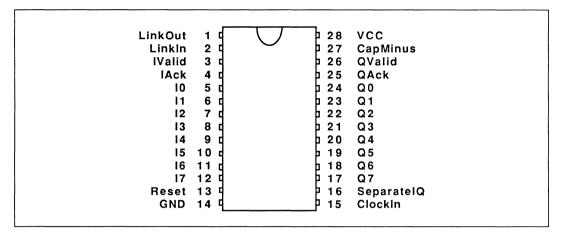


Figure 8.4 IMS C011 Mode 1 pinout

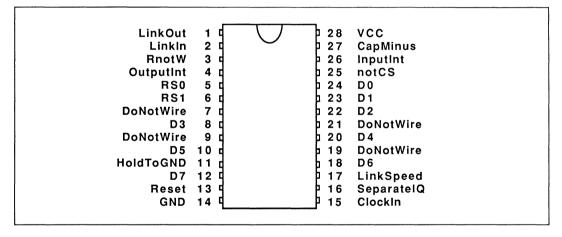


Figure 8.5 IMS C011 Mode 2 pinout

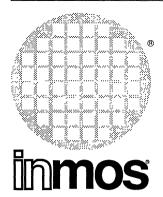
# 9 Ordering

This section indicates the designation of package selections for the IMS C011. Speed of **ClockIn** is 5 MHz for all parts.

For availability contact local INMOS sales office or authorised distributor.

Table 9.1 IMS C011 ordering details

INMOS designation	Package
IMS C011-P20S	28 pin plastic dual-in-line
IMS C011-S20S	28 pin ceramic sidebraze
IMS C011-S20M	28 pin ceramic sidebraze MIL Spec



# IMS C012 link adaptor

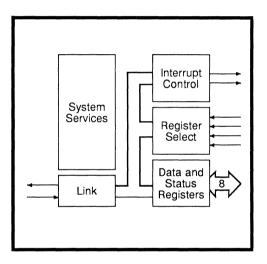
# **Engineering Data**

#### **FEATURES**

Standard INMOS link protocol
10 or 20 Mbits/sec operating speed
Communicates with INMOS transputers
Converts between serial link and parallel bus
Tristate bidirectional bus interface
Memory mapped registers
Interrupt capability
Single +5V ±5% power supply
TTL and CMOS compatibility
120mW power dissipation
Standard 24 pin 0.3" plastic package

#### **APPLICATIONS**

Connecting microprocessors to transputers High speed links between microprocessors Inter-family microprocessor interfacing



42 1413 02 May 1989

#### 1 Introduction

The INMOS communication link is a high speed system interconnect which provides full duplex communication between members of the INMOS transputer family, according to the INMOS serial link protocol. The IMS C012, a member of this family, provides for full duplex transputer link communication with standard microprocessor and sub-system architectures, by converting bi-directional serial link data into parallel data streams.

All INMOS products which use communication links, regardless of device type, support a standard communications frequency of 10 Mbits/sec; most products also support 20 Mbits/sec. Products of different type or performance can, therefore, be interconnected directly and future systems will be able to communicate directly with those of today. The IMS C012 link will run at either the standard speed of 10 Mbits/sec or at the higher speed of 20 Mbits/sec. Data reception is asynchronous, allowing communication to be independent of clock phase.

The IMS C012 provides an interface between an INMOS serial link and a microprocessor system bus. Status and data registers for both input and output ports can be accessed across the byte-wide bi-directional interface. Two interrupt outputs are provided, one to indicate input data available and one for output buffer empty.

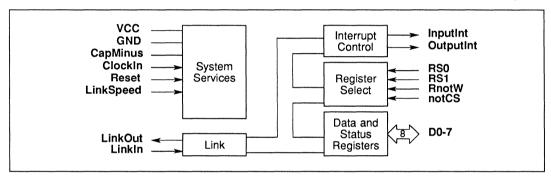


Figure 1.1 IMS C012 block diagram

# 2 Pin designations

Table 2.1 IMS C012 services and link

Pin	In/Out	Function
VCC, GND CapMinus ClockIn Reset	in in	Power supply and return External capacitor for internal clock power supply Input clock System reset
LinkIn LinkOut	in out	Serial data input channel Serial data output channel

Table 2.2 IMS C012 parallel interface

Pin	In/Out	Function
D0-7	in/out	Bi-directional data bus
notCS	in	Chip select
RS0-1	in	Register select
RnotW	in	Read/write control signal
Inputint	out	Interrupt on link receive buffer full
OutputInt	out	Interrupt on link transmit buffer empty
LinkSpeed	in	Select link speed as 10 or 20 Mbits/sec
HoldToGND		Must be connected to GND

Signal names are prefixed by **not** if they are active low, otherwise they are active high. Pinout details for various packages are given on page 548.

#### 3 System services

System services include all the necessary logic to start up and maintain the IMS C012.

#### 3.1 Power

Power is supplied to the device via the VCC and GND pins. The supply must be decoupled close to the chip by at least one 100 nF low inductance (e.g. ceramic) capacitor between VCC and GND. Four layer boards are recommended; if two layer boards are used, extra care should be taken in decoupling.

AC noise between VCC and GND must be kept below 200 mV peak to peak at all frequencies above 100 KHz. AC noise between VCC and the ground reference of load capacitances must be kept below 200 mV peak to peak at all frequencies above 30 MHz. Input voltages must not exceed specification with respect to VCC and GND, even during power-up and power-down ramping, otherwise *latchup* can occur. CMOS devices can be permanently damaged by excessive periods of latchup.

#### 3.2 CapMinus

The internally derived power supply for internal clocks requires an external low leakage, low inductance  $1\mu$ F capacitor to be connected between VCC and CapMinus. A ceramic capacitor is preferred, with an impedance less than 3 Ohms between 100 KHz and 10 MHz. If a polarised capacitor is used the negative terminal should be connected to CapMinus. Total PCB track length should be less than 50 mm. The positive connection of the capacitor must be connected directly to VCC. Connections must not otherwise touch power supplies or other noise sources.

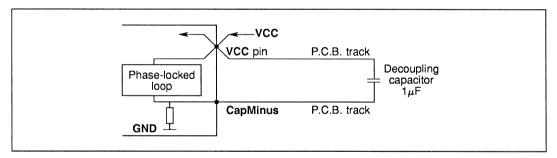


Figure 3.1 Recommended PLL decoupling

#### 3.3 Clockin

Transputer family components use a standard clock frequency, supplied by the user on the **ClockIn** input. The nominal frequency of this clock for all transputer family components is 5 MHz, regardless of device type, transputer word length or processor cycle time. High frequency internal clocks are derived from **ClockIn**, simplifying system design and avoiding problems of distributing high speed clocks externally.

A number of transputer family devices may be connected to a common clock, or may have individual clocks providing each one meets the specified stability criteria. In a multi-clock system the relative phasing of **ClockIn** clocks is not important, due to the asynchronous nature of the links. Mark/space ratio is unimportant provided the specified limits of **ClockIn** pulse widths are met.

Oscillator stability is important. ClockIn must be derived from a crystal oscillator; RC oscillators are not sufficiently stable. ClockIn must not be distributed through a long chain of buffers. Clock edges must be monotonic and remain within the specified voltage and time limits.

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTE
TDCLDCH	ClockIn pulse width low	40			ns	1
TDCHDCL	ClockIn pulse width high	40			ns	1
TDCLDCL	ClockIn period	1	200	400	ns	1,2,4
TDCerror	ClockIn timing error			±0.5	ns	1,3
TDC1DC2	Difference in ClockIn for 2 linked devices	1		400	ppm	1,4
TDCr	ClockIn rise time	1		10	ns	1,5
TDCf	ClockIn fall time			8	ns	1,5

Table 3.1 Input clock

- 1 These paramters are not tested.
- 2 Measured between corresponding points on consecutive falling edges.
- 3 Variation of individual falling edges from their nominal times.
- 4 This value allows the use of 200ppm crystal oscillators for two devices connected together by a link.
- 5 Clock transitions must be monotonic within the range VIH to VIL (table 6.3).

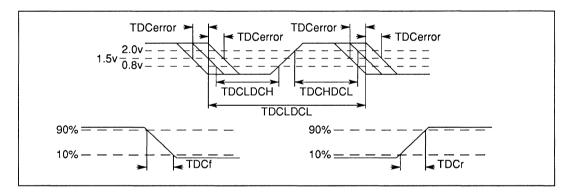


Figure 3.2 ClockIn timing

#### 3.4 Reset

The **Reset** pin can go high with **VCC**, but must at no time exceed the maximum specified voltage for **VIH**. After **VCC** is valid **ClockIn** should be running for a minimum period **TDCVRL** before the end of **Reset**. **LinkIn** must be held low during **Reset**.

Reset initialises the IMS C012 to the following state: LinkOut is held low; the interrupt outputs InputInt and OutputInt are held low; interrupts are disabled; D0-7 are high impedance.

Table	3.2	Reset
-------	-----	-------

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTE
TPVRH	Power valid before Reset	10			ms	
TRHRL	Reset pulse width high	8			ClockIn	1
TDCVRL	Clockln running before Reset end	10			ms	2

- 1 Full periods of Clockin TDCLDCL required.
- 2 At power-on reset.

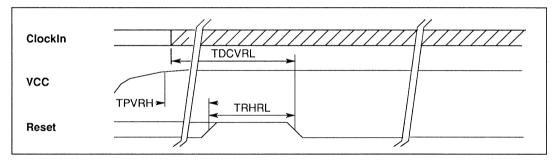


Figure 3.3 Reset Timing

#### 4 Links

INMOS bi-directional serial links provide synchronized communication between INMOS products and with the outside world. Each link comprises an input channel and output channel. A link between two devices is implemented by connecting a link interface on one device to a link interface on the other device. Every byte of data sent on a link is acknowledged on the input of the same link, thus each signal line carries both data and control information.

The quiescent state of a link output is low. Each data byte is transmitted as a high start bit followed by a one bit followed by eight data bits followed by a low stop bit. The least significant bit of data is transmitted first. After transmitting a data byte the sender waits for the acknowledge, which consists of a high start bit followed by a zero bit. The acknowledge signifies both that a process was able to receive the acknowledged data byte and that the receiving link is able to receive another byte.

Links are not synchronised with **ClockIn** and are insensitive to its phase. Thus links from independently clocked systems may communicate, providing only that the clocks are nominally identical and within specification.

Links are TTL compatible and intended to be used in electrically quiet environments, between devices on a single printed circuit board or between two boards via a backplane. Direct connection may be made between devices separated by a distance of less than 300 millimetres. For longer distances a matched 100 ohm transmission line should be used with series matching resistors **RM**. When this is done the line delay should be less than 0.4 bit time to ensure that the reflection returns before the next data bit is sent.

Buffers may be used for very long transmissions. If so, their overall propagation delay should be stable within the skew tolerance of the link, although the absolute value of the delay is immaterial.

The IMS C012 link supports the standard INMOS communication speed of 10 Mbits/sec. In addition it can be used at 20 Mbits/sec. Link speed is selected by **LinkSpeed**; when the **LinkSpeed** pin is low, the link operates at the standard 10 Mbits/sec; when high it operates at 20 Mbits/sec.

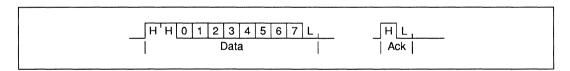


Figure 4.1 IMS C012 link data and acknowledge packets

SYMBOL	PARAMETER		MIN	NOM	MAX	UNITS	NOTE
TJQr	LinkOut rise time				20	ns	1
TJQf	LinkOut fall time				10	ns	1
TJDr	LinkIn rise time				20	ns	1
TJDf	LinkIn fall time				20	ns	1
TJQJD	Buffered edge delay		0			ns	
TJBskew	Variation in TJQJD 2	0 Mbits/s			3	ns	2
	1	0 Mbits/s			10	ns	2
CLIZ	LinkIn capacitance @	f=1MHz			7	pF	1
CLL	LinkOut load capacitance	i			50	pF	
RM	Series resistor for 100 $\Omega$ transmissi	on line		56		ohms	

Table 4.1 Link

- 1 These paramters are sampled, but are not 100% tested.
- 2 This is the variation in the total delay through buffers, transmission lines, differential receivers etc., caused by such things as short term variation in supply voltages and differences in delays for rising and falling edges.

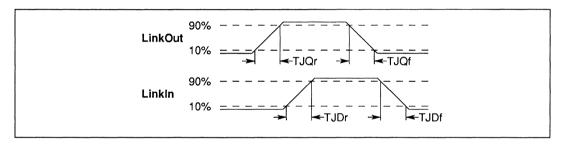


Figure 4.2 IMS C012 link timing

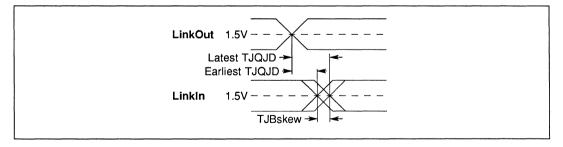


Figure 4.3 IMS C012 buffered link timing

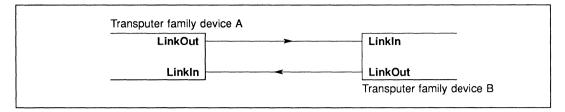


Figure 4.4 IMS C012 Links directly connected

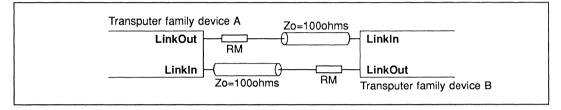


Figure 4.5 IMS C012 Links connected by transmission line

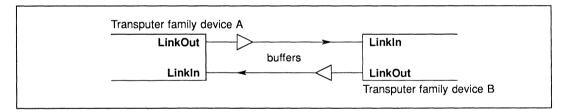


Figure 4.6 IMS C012 Links connected by buffers

#### 5 Parallel interface

The IMS C012 provides an interface between a link and a microprocessor style bus. Operation of the link adaptor is controlled through the parallel interface bus lines **D0-7** by reading and writing various registers in the link adaptor. Registers are selected by **RS0-1** and **RnotW**, and the chip enabled with **notCS**.

For convenience of description, the device connected to the parallel side of the link adaptor is presumed to be a microprocessor, although this will not always be the case.

#### 5.1 D0-7

Data is communicated between a microprocessor bus and the link adaptor via the bidirectional bus lines **D0-7**. The bus is high impedance unless the link adaptor chip is selected and the **RnotW** line is high. The bus is used by the microprocessor to access status and data registers.

#### 5.2 notCS

The link adaptor chip is selected when **notCS** is low. Register selectors **RS0-1** and **RnotW** must be valid before **notCS** goes low; **D0-7** must also be valid if writing to the chip (**RnotW** low). Data is read by the link adaptor on the rising edge of **notCS**.

#### 5.3 RnotW

**RnotW**, in conjunction with **notCS**, selects the link adaptor registers for read or write mode. When **RnotW** is high, the contents of an addressed register appear on the data bus **D0-7**; when **RnotW** is low the data on **D0-7** is written into the addressed register. The state of **RnotW** is latched into the link adaptor by **notCS** going low; it may be changed before **notCS** returns high, within the timing restrictions given.

#### 5.4 RS0-1

One of four registers is selected by RS0-1. A register is addressed by setting up RS0-1 and then taking notCS low; the state of RnotW when notCS goes low determines whether the register will be read or written. The state of RS0-1 is latched into the link adaptor by notCS going low; it may be changed before notCS returns high, within the timing restrictions given. The register set comprises a read-only data input register, a write-only data output register and a read/write status register for each.

RS1	RS0	RnotW	Register
0	0	1	Read data
0	0	0	Invalid
0	1	1	Invalid
0	1	0	Write data
1	0	1	Read input status
1	0	0	Write input status
1	1	1	Read output status
1	1	0	Write output status

Table 5.1 IMS C012 register selection

#### 5.4.1 Input Data Register

This register holds the last data packet received from the serial link. It never contains acknowledge packets. It contains valid data only whilst the *data present* flag is set in the input status register. It cannot be assumed to contain valid data after it has been read; a double read may or may not return valid data on the second read. If *data present* is valid on a subsequent read it indicates new data is in the buffer. Writing to this register will have no effect.

Table 5.2 IMS C012 parallel interface conti	Table	5.2	IMS	C012	parallel	interface	contro
---	-------	-----	-----	------	----------	-----------	--------

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTE
TRSVCSL	Register select setup	5			ns	
TCSLRSX	Register select hold	5			ns	
TRWVCSL	Read/write strobe setup	5			ns	
TCSLRWX	Read/write strobe hold	5	ĺ		ns	
TCSLCSH	Chip select active	50	ì		ns	
TCSHCSL	Delay before re-assertion of chip select	50	ļ		ns	

Table 5.3 IMS C012 parallel interface read

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTE
TLdVIIH	Start of link data to InputInt high			13	bits	1
TCSLIIL	Chip select to InputInt low			30	ns	
TCSLDrX	Chip select to bus active	5			ns	ŀ
TCSLDrV	Chip select to data valid			40	ns	
TCSHDrZ	Chip select high to bus tristate			25	ns	
TCSHDrX	Data hold after chip select high	5		1	ns	
TCSHLaV	Chip de-select to start of Ack	0.8		2	bits	1,2

- 1 Unit of measurement is one link data bit time; at 10 Mbits/s data link speed, one bit time is nominally 100 ns.
- 2 Maximum time assumes there is no data packet already on the link. Maximum time with data on the link is extended by 11 bits.

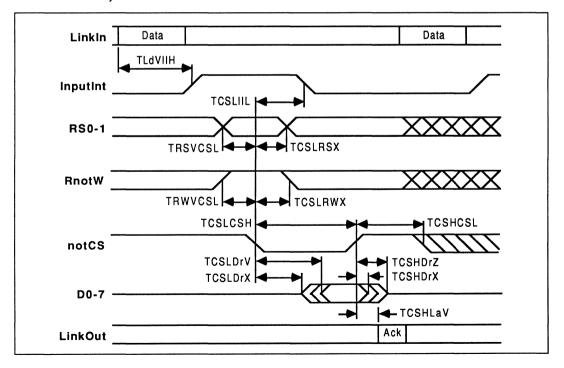


Figure 5.1 IMS C012 read parallel data from link adaptor

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTE
TCSHDwV	Data setup	15			ns	
TCSHDwX	Data hold	5			ns	
TCSLOIL	Chip select to OutputInt low			30	ns	
TCSHLdV	Chip select high to start of link data	0.8		2	bits	1,2
TLaVOIH	Start of link Ack to OutputInt high			3	bits	1,3
TLdVOIH	Start of link data to OutputInt high			13	bits	1,3

Table 5.4 IMS C012 parallel interface write

- 1 Unit of measurement is one link data bit time; at 10 Mbits/s data link speed, one bit time is nominally 100 ns.
- 2 Maximum time assumes there is no acknowledge packet already on the link. Maximum time with acknowledge on the link is extended by 2 bits.
- 3 Both data transmission and the returned acknowledge must be completed before OutputInt can go high.

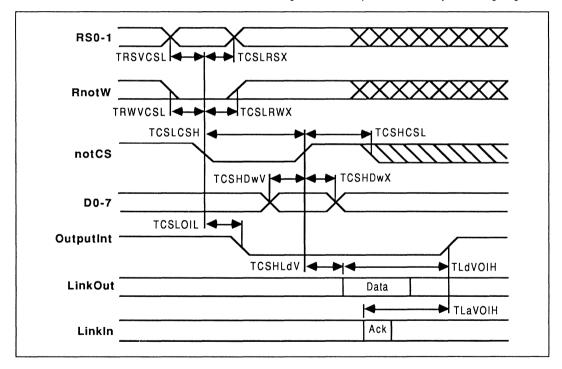


Figure 5.2 IMS C012 write parallel data to link adaptor

5 Parallel interface 541

#### 5.4.2 Input Status Register

This register contains the *data present* flag and the *interrupt enable* control bit for **inputint**. The *data present* flag is set to indicate that data in the data input buffer is valid. It is reset low only when the data input buffer is read, or by **Reset**. When writing to this register, the *data present* bit must be written as zero.

The *interrupt enable* bit can be set and reset by writing to the status register with this bit high or low respectively. When the *interrupt enable* and *data present* flags are both high, the **InputInt** output will be high (page 541). Resetting *interrupt enable* will take **InputInt** low; setting it again before reading the data input register will set **InputInt** high again. The *interrupt enable* bit can be read to determine its status.

When writing to this register, bits 2-7 must be written as zero; this ensures that they will be zero when the register is read. Failure to write zeroes to these bits may result in undefined data being returned by these bits during a status register read.

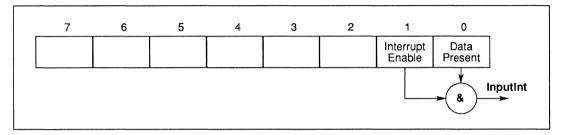


Figure 5.3 IMS C012 input status register

#### 5.4.3 Output Data Register

Data written to this link adaptor register is transmitted out of the serial link as a data packet. Data should only be written to this register when the *output ready* bit in the output status register is high, otherwise data already being transmitted may be corrupted. Reading this register will result in undefined data being read.

#### 5.4.4 Output Status Register

This register contains the *output ready* flag and the *interrupt enable* control bit for **OutputInt**. The *output ready* flag is set to indicate that the data output buffer is empty. It is reset low only when data is written to the data output buffer; it is set high by **Reset**. When writing to this register, the *output ready* bit must be written as zero.

The *interrupt enable* bit can be set and reset by writing to the status register with this bit high or low respectively. When the *interrupt enable* and *output ready* flags are both high, the **OutputInt** output will be high (page 542). Resetting *interrupt enable* will take **OutputInt** low; setting it again whilst the data output register is empty will set **OutputInt** high again. The *interrupt enable* bit can be read to determine it's status.

When writing to this register, bits 2-7 must be written as zero; this ensures that they will be zero when the register is read. Failure to write zeroes to these bits may result in undefined data being returned by these bits during a status register read.

#### 5.5 InputInt

The **InputInt** output is set high to indicate that a data packet has been received from the serial link. It is inhibited from going high when the *interrupt enable* bit in the input status register is low (page 541). **InputInt** is reset low when data is read from the input data register (page 538) and by **Reset** (page 534).

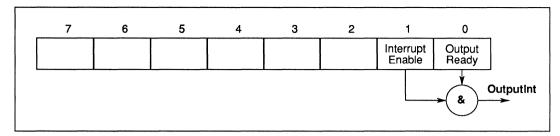


Figure 5.4 IMS C012 output status register

#### 5.6 OutputInt

The **OutputInt** output is set high to indicate that the link is free to receive data from the microprocessor for transmission as a data packet out of the serial link. It is inhibited from going high when the *interrupt enable* bit in the output status register is low (page 541). **OutputInt** is reset low when data is written to the data output register (page 541); it is set low by **Reset** (page 534).

#### 5.7 Data read

A data packet received on the input link sets the *data present* flag in the input status register. If the *interrupt enable* bit in the status register is set, the *InputInt* output pin will be set high. The microprocessor will either respond to the interrupt (if the *interrupt enable* bit is set) or will periodically read the input status register until the *data present* bit is high.

When data is available from the link, the microprocessor reads the data packet from the data input register. This will reset the *data present* flag and cause the link adaptor to transmit an acknowledge packet out of the serial link output. **InputInt** is automatically reset by reading the data input register; it is not necessary to read or write the input status register.

#### 5.8 Data write

When the data output buffer is empty the *output ready* flag in the output status register is set high. If the *interrupt enable* bit in the status register is set, the **OutputInt** output pin will also be set high. The microprocessor will either respond to the interrupt (if the *interrupt enable* bit is set) or will periodically read the output status register until the *output ready* bit is high.

When the *output ready* flag is high, the microprocessor can write data to the data output buffer. This will result in the link adaptor resetting the *output ready* flag and commencing transmission of the data packet out of the serial link. The *output ready* status bit will remain low until an acknowledge packet is received by the input link. This will set the *output ready* flag high; if the *interrupt enable* bit is set, **OutputInt** will also be set high.

# 6 Electrical specifications

#### 6.1 DC electrical characteristics

Table 6.1 Absolute maximum ratings

SYMBOL	PARAMETER	MIN	MAX	UNITS	NOTE
VCC	DC supply voltage	0	7.0	V	1,2,3
VI, VO	Voltage on input and output pins	-0.5	VCC+0.5	V	1,2,3
11	Input current		±25	mA	4
OSCT	Output short circuit time (one pin)		1	s	2
TS	Storage temperature	-65	150	°C	2
TA	Ambient temperature under bias	-55	125	°C	2
PDmax	Maximum allowable dissipation		600	mW	

#### Notes

- 1 All voltages are with respect to GND.
- 2 This is a stress rating only and functional operation of the device at these or any other conditions beyond those indicated in the operating sections of this specification is not implied. Stresses greater than those listed may cause permanent damage to the device. Exposure to absolute maximum rating conditions for extended periods may affect reliability.
- 3 This device contains circuitry to protect the inputs against damage caused by high static voltages or electrical fields. However, it is advised that normal precautions be taken to avoid application of any voltage higher than the absolute maximum rated voltages to this high impedance circuit. Unused inputs should be tied to an appropriate logic level such as VCC or GND.
- 4 The input current applies to any input or output pin and applies when the voltage on the pin is between GND and VCC.

Table 6.2 Operating conditions

SYMBOL	PARAMETER	MIN	MAX	UNITS	NOTE
VCC	DC supply voltage	4.75	5.25	٧	1
VI, VO	Input or output voltage	0	VCC	V	1,2
CL	Load capacitance on any pin		60	pF	
TA	Operating temperature range	0	70	۰C	3

- 1 All voltages are with respect to GND.
- 2 Excursions beyond the supplies are permitted but not recommended; see DC characteristics.
- 3 Air flow rate 400 linear ft/min transverse air flow.

SYMBOL	PARAME	TER	MIN	MAX	UNITS	NOTE
VIH	High level input voltage		2.0	VCC+0.5	V	1,2
VIL	Low level input voltage		-0.5	0.8	V	1,2
l II	Input current	@ GND <vi<vcc< td=""><td></td><td>±10</td><td>μA</td><td>1,2,7</td></vi<vcc<>		±10	μA	1,2,7
	·			±200	$\mu$ A	1,2,8
VOH	Output high voltage	@ IOH=2mA	VCC-1		V	1,2
VOL	Output low voltage	@ IOL=4mA		0.4	V	1,2
IOS	Output short circuit current	@ GND <vo<vcc< td=""><td>36</td><td>65</td><td>mA</td><td>1,2,3,6</td></vo<vcc<>	36	65	mA	1,2,3,6
			65	100	mA	1,2,4,6
IOZ	Tristate output current	@ GND <vo<vcc< td=""><td></td><td>±10</td><td><math>\mu A</math></td><td>1,2</td></vo<vcc<>		±10	$\mu A$	1,2
PD	Power dissipation			120	mW	2,5
CIN	Input capacitance	@ f=1MHz		7	pF	6
COZ	Output capacitance	@ f=1MHz		10	pF	6

Table 6.3 DC characteristics

- 1 All voltages are with respect to GND.
- 2 Parameters for IMS C012-S measured at 4.75V<VCC<5.25V and 0°C<TA<70°C. Input clock frequency = 5 MHz.</p>
- 3 Current sourced from non-link outputs.
- 4 Current sourced from link outputs.
- 5 Power dissipation varies with output loading.
- 6 This parameter is sampled and not 100% tested.
- 7 For inputs other than those in Note 8.
- 8 For pins 2, 3, 5, 6, 7, 9, 11, 13, 14, 21.

#### 6.2 Equivalent circuits

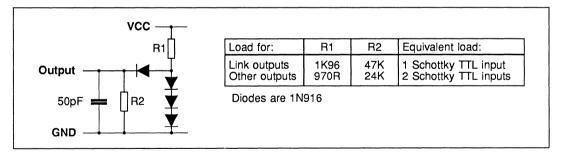


Figure 6.1 Load circuit for AC measurements

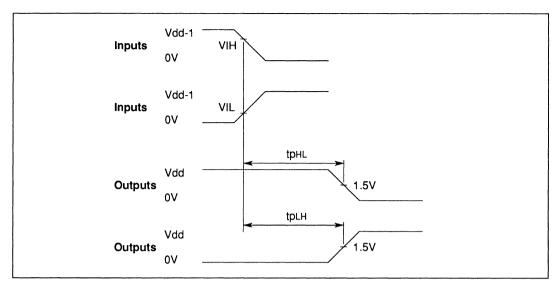


Figure 6.2 AC measurements timing waveforms

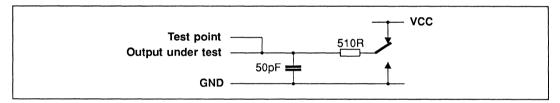


Figure 6.3 Tristate load circuit for AC measurements

## 6.3 AC timing characteristics

Table 6.4 Input, output edges

SYMBOL	PARAMETER	MIN	MAX	UNITS	NOTE
TDr	Input rising edges	2	20	ns	1,2
TDf	Input falling edges	2	20	ns	1,2
TQr	Output rising edges		25	ns	1
TQf	Output falling edges		15	ns	1
CSLaHZ	Chip select high to tristate		25	ns	
CSLaLZ	Chip select low to tristate		25	ns	

- 1 Non-link pins; see section on links.
- 2 All inputs except Clockin; see section on Clockin.

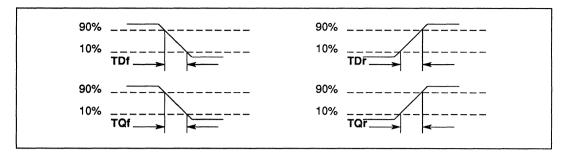


Figure 6.4 IMS C012 input and output edge timing

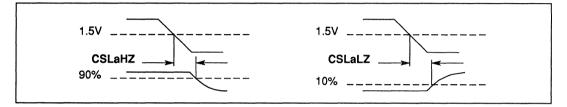


Figure 6.5 IMS C012 tristate timing relative to notCS

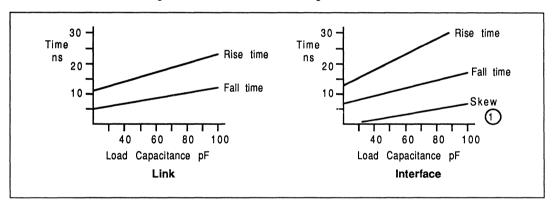


Figure 6.6 Typical rise/fall times

1 Skew is measured between notCS with a standard load (2 Schottky TTL inputs and 30pF) and notCS with a load of 2 Schottky TTL inputs and varying capacitance.

#### 6.4 Power rating

Internal power dissipation  $P_{INT}$  of transputer and peripheral chips depends on VCC, as shown in figure 6.7.  $P_{INT}$  is substantially independent of temperature.

Total power dissipation  $P_D$  of the chip is

$$P_D = P_{INT} + P_{IO}$$

where  $P_{IO}$  is the power dissipation in the input and output pins; this is application dependent.

Internal working temperature  $T_J$  of the chip is

$$T_J = T_A + \theta J_A * P_D$$

where  $T_A$  is the external ambient temperature in °C and  $\theta J_A$  is the junction-to-ambient thermal resistance in °C/W.  $\theta J_A$  for each package is given in the Packaging Specifications section.

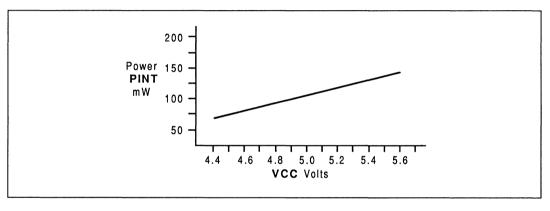


Figure 6.7 IMS C012 internal power dissipation vs VCC

# 7 Package specifications

# 7.1 24 pin plastic dual-in-line package

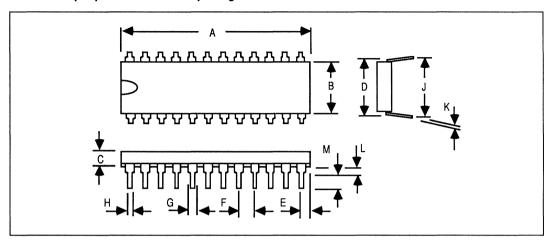


Figure 7.1 24 pin plastic dual-in-line package dimensions

Table 7.1 24 pin plastic dual-in-line packagedimensions

	Millimetres		Inches		
DIM	NOM	TOL	NOM	TOL	Notes
Α	31.242	+0.508	1.230	+0.020	
		-0.254		-0.010	
В	6.604	±0.127	0.260	±0.005	
С	3.302	±0.381	0.130	±0.015	
D	7.620	±0.127	0.300	±0.005	
E	1.651	±0.127	0.065	$\pm 0.005$	
F	2.540	±0.127	0.100	$\pm 0.005$	
G	1.524	±0.127	0.060	±0.005	
Н	0.457	±0.127	0.018	$\pm 0.005$	
J	8.382	±0.508	0.330	±0.020	
K	0.254	±0.025	0.010	±0.001	
L	0.508	±0.127	0.020	$\pm 0.005$	
M	3.048		0.120		Minimum

Package weight is approximately 2 grams

Table 7.2 24 pin plastic dual-in-line package junction to ambient thermal resistance

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTE
θJA	At 400 linear ft/min transverse air flow		115		°C/W	

#### 7.2 Pinout

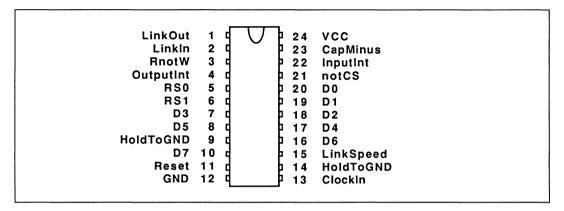


Figure 7.2 IMS C012 pinout

# 8 Ordering

This section indicates the designation of package selections for the IMS C012. Speed of ClockIn is 5 MHz for all parts.

For availability contact local INMOS sales office or authorised distributor.

Table 8.1 IMS C012 ordering details

INMOS designation	Package		
IMS C012-P20S	24 pin plastic dual-in-line		



# quality and reliability

# A Quality and Reliability

The INMOS quality programme is set up to be attentive to every phase of the semiconductor product life cycle. This includes specific programmes in each of the following areas:

- Total Quality Control (TQC)
- · Quality and Reliability in Design
- Document Control
- New Product Qualification
- Product Monitoring Programme
- Production Testing and Quality Monitoring Procedure

#### A.1 Total quality control (TQC) and reliability programme

Our objective to continuously build improved quality and reliability into every INMOS part has resulted in a comprehensive Quality/Reliability Programme of which we are proud. This programme demonstrates INMOS' serious commitment to supporting the quality and reliability needs of the electronics marketplace.

INMOS is systematically shifting away from a traditional screening approach to quality control and towards one of building in Experimental Design quality through Statistical Process Control (SPC). This new direction was initiated with a vigorous programme of education and scientific method training.

In the first year of the programme approximately 80 INMOS employees worldwide received thorough SPC training. This training has been extended to cover advanced SPC and experimental design. Some of the courses taught are listed below:

- Experimental Design Techniques
- Statistical Process Control Methods
- · Quality Concepts
- Problem Solving Techniques
- Statistical Software Analysis Techniques

Today INMOS utilizes experimental design techniques and process control/monitoring throughout its development and manufacturing cycles. The following TQC tools are currently supported by extensive databases and analysis software.

- 1. Pareto charts
- 6. Correlation Plots
- 2. Cause/Effect Diagrams
- 7. Control Charts
- 3. Process Flow Charts
- 8. Experimental Design
- 4. Run Charts
- 9. Process Capability Studies
- 5. Histograms

#### A.2 Quality and reliability in design

The INMOS quality programme begins with the design of new INMOS products. The following procedures are examples from the INMOS programme to design quality and reliability into every product.

Innovative design techniques are employed to achieve product performance using, whenever possible, state of the art techniques. For example, INMOS uses 300 nm gate oxides on its high performance graphics, SRAM and MICRO products to obtain the reliability inherent in the thicker gate oxide. In addition, circuit design engineers work hand in hand with process engineers to optimise the design for the process and the

process for the product family. The result is a highly reliable design implemented in a process technology achievable within manufacturing.

INMOS products are designed to have parametric margins beyond the product target specifications. The design performance is verified using simulations of circuit performance over voltage and temperature values beyond those of specified product operation, including verification beyond the military performance range. In addition, the device models are chosen to ensure tolerance to wide variations in process parameters beyond those expected in manufacture.

The design process includes consideration of quality issues such as signal levels available for sensing, reduction of internal noise levels, stored data integrity and testability of all device functions. Electro-static damage protection techniques are included in the design with input protection goals of 2K volts for MIL-STD-883 testing methods. Specific customer requirements can be met by matching their detailed specifications against INMOS designed in margins.

The completion of the design includes the use of INMOS computer aided design software to fully check and verify the design and layout. This improves quality as well as ensuring the timely introduction of new products.

#### A.3 Document control

The Document Control Department maintains control over all manufacturing specifications, lot travellers, procurement specifications and drawings, reticle tapes and test programmes. New specifications and changes are subject to approval by the Engineering and Manufacturing managers or their delegates. Change is rigorously controlled through an Engineering Change Notice procedure, and QA department managers screen and approve all such changes.

An extensive archiving system ensures that the history of any Change Notice is readily available.

Document Control also has responsibility for controlling in-line documentation in all manufacturing areas which includes distribution of specifications, control of changes and liaison with production control and manufacturing in introducing changed procedures into the line.

Extensive use is made of computer systems to control documentation on an international basis.

#### A.4 New product qualification

INMOS performs a thorough internal product qualification prior to the delivery of any new product, other than engineering samples of prototypes to customers.

Care is taken to select a representative sample from the final prototype material. This typically consists of three different production lots. Testing is then done to assure the initial product reliability levels are achieved. Product qualifications are done in accordance with MIL-STD-883, methods 5004 and 5005, or CECC/BS9000.

The initial INMOS qualification data, and the ongoing monitor data can be very useful in the user qualification decision process. INMOS also has a very successful history of performing customer qualification testing in-house and performing joint qualification programmes with customers. INMOS remains committed to joint customer/vendor programmes.

#### A.5 Product monitoring programme

At the levels of quality and reliability performance required today (low PPM and FIT levels), it is essential that a large statistically significant, current product database be maintained. One of the programmes that INMOS uses to accomplish this is the Product Monitoring Programme (PMP).

The PMP is a comprehensive ongoing programme of reliability testing. A small sample is pulled from production lots of a particular part type. This population is then used to create the specific samples to put on the various operating and environmental tests. Tests run in this programme include extended temperature

operating life, THB and temperature cycle. Efforts are continuing to identify and correlate more accelerated tests to be used in the PMP.

#### A.6 Production testing and quality monitoring procedure

#### A.6.1 Reliability testing

INMOS' primary reliability test method is to bias devices at their maximum rated operating power supply level in a 140° C ambient temperature. A scheme of time varying input signals is used to simulate the complete functional operation of the device. The failure rate is then computed from the results of the operating life test using Arrhenius modelling for each specific failure mechanism known. The failure rate is reported at a temperature that is a typical worst case application environment and is expressed in units of FITs where 1 FIT = 1 Fail in 10E9 device hours, (100 FIT = 0.01%/1000 Hrs). The current database enables the failure rate to be valid over various environmental conditions.

The failure rate goal for INMOS products is 100 FITs or less at product introduction with a 50 FIT level to be attained within one year.

For plastic packaged product, additional testing methods and reliability indices become important. Humidity testing is used to evaluate the relative hermeticity of the package, and thermal cycling tests are used principally to evaluate the durability of the assembly (e.g. die/bond attach).

The Humidity Test comprises of temperature, humidity, bias (THB) at 85°C, 85% Relative Humidity, and a 5V static bias configuration selected to maintain the component in a state of minimum power dissipation and enhance the formation of galvanic corrosion. INMOS reliability goals have always been to meet or better the current 'industry standards' and a target of less than 1% failures through 1000 hours of THB at 90% confidence has been set.

The Thermal Cycling tests are performed from  $-65^{\circ}$ C to  $+150^{\circ}$ C for 500–1000 cycles, with no bias applied. Thermal Shock tests using a liquid to liquid (Freon) method are cycled between  $-55^{\circ}$ C and  $+125^{\circ}$ C. The INMOS Reliability qualification and monitoring goal for the above tests is less than 1% failures at 90% confidence.

#### A.6.2 Production testing

Electrical testing at INMOS begins while the devices are still in wafer form before being divided into individual die. While in this form, two different types of electrical test are performed.

The Parametric Probe test is to verify that the individual component parameters are within their design limits. This is accomplished by testing special components on the wafer. The results of these tests provide feedback to our wafer fab manufacturing facilities which allows them to ensure that the components used in the actual devices perform within their design limits. This testing is performed on all lots which are processed, and any substandard wafers discarded. These components are placed in the scribe streets of the wafer so they are destroyed in the dicing operation when they are not of any further use. By placing them there, valuable chip real estate is saved, thereby holding down cost while still providing the necessary data.

The Electrical Probe test performed on all wafers is the test of each individual circuit or chip on every wafer. The defective dice are identified so they may be later discarded after the wafer has been separated into individual die. This test fully exercises the circuits for all AC and DC datasheet parameters in addition to verifying functionality.

After the dice have been assembled into packages they are again tested in our Final Test operation. In a mature product the typical flow is:

- Preburn-in test
- Burn-in at 140°C
- · Final test

- PDA (Percent Defect Allowed)
- Device Symbolisation
- QA Final Acceptance

The temperature setting used for hot testing is selected so that the junction temperature is the same as it would be after thermal stabilisation occurred in the specified environment. This is calculated using the hot temperature power dissipation along with the thermal resistance of the package used. All INMOS product is electrically tested and burned-in prior to shipment. Historically, the industry has selected burn-in times using the MIL Standards as a guide (when the market would support the cost) or on a 'best guess' basis dominated by cost considerations. Whereas INMOS invoke a burn-in reduction exercise to ensure the reduced time has no reliability impact.

#### A.6.3 Quality monitoring procedure

In the Outgoing Quality Monitoring programme, random samples are pulled from lots, that have been successfully tested to data sheet criteria. Rejected lots are 100% retested and more importantly, failures are analysed and corrective actions identified to prevent the recurrence of specific problems.

The extensive series of electrical tests with the associated Burn-in PDA limits and Quality Assurance tests ensure we will be able to continue to improve our high quality and reliability standards.

## INMOS MIL-STD-883C/MIL-I-45208 MATERIAL PROCUREMENT & PRODUCT FLOW

**INMOS INMOS OFFSHORE NEWPORT** COL. SPGS. **ASSEMBLY** U.K. USA [6] [5] [5] 600 mil Package Parts, All MICRO & Ğ17x Parts [6] 300 mil DIP, LCC & FLAT PACK

Wafer	Fab	Matil	Procurement	[2]

Wafer Fab Process [2]

Wafer Level Elect. Test [2]

Internal Visual QA Sample [2]

Assembly Mat'l Procurement

Assembly Mat'l QA Inspect

Assembly Process [1]

**Electrical Testing** 

Burn-in

[Solder Dip] [3]

QCI Group A,B

QCI Group C,D

Certificate of Conformance

[CSI, GSI, PVT] [4]

#### Notes:

- [1] Anam, Korea or GTE, Taiwan
- [2] Newport Fab. Product: All NMOS, CMOS SRAM All Transputer All G17x (CLUT)
- [3] Hot Solder Dip as reg'd at Colo. Spgs. Subcontractor
- [4] As required by Customer
- SRAM Parts

$\nabla$	Raw Material Procurement
0	Manufacturing Process
	QA Gate



# index

### B Index

! 13	link IMS T801 171
" 19	link IMS T805 105
() 18	Address
<b>* 18</b> , 114, 117, 180, 183, 250, 253, 321, 388,	bus IMS T222 423
443	bus IMS T414 359
<b>+ 18</b> , 114, 117, 180, 183, 250, 253, 321, 323,	bus IMS T425 290
388, 390, 443, 445	bus IMS T800 221
<b>- 18</b> , 114, 180, 250, 321, 323, 388, 390, 443,	bus IMS T801 160
445	bus IMS T805 <b>80</b>
<b>18</b> , 114, 117, 180, 183, 250, 253, 321, 388,	byte IMS T222 420
443	byte IMS T414 355
/\ 18, 114, 180, 250, 321, 388, 443	byte IMS T425 286
:= 13	byte IMS T800 217
<b>18</b> , 114, 117, 180, 183, 250, 253, 321, 323,	byte IMS T801 157
388, 390, 443, 445	byte IMS T805 77
<< 18, 114, 180, 250, 321, 388, 443	mark IMS M212 465
<= <b>18</b> , 114, 117, 180, 183, 250, 253, 321, 323,	refresh IMS T414 373
388, 390, 443, 445	refresh IMS T425 305
<b>18</b> , 114, 117, 180, 183, 250, 253, 321, 323,	refresh IMS T800 235
388, 390, 443, 445	refresh IMS T805 91
<b>= 18</b> , 114, 117, 180, 183, 250, 253, 321, 323,	space IMS T222 400, 403
388, 390, 443, 445	space IMS T225 454
<b>18</b> , 114, 117, 180, 183, 250, 253, 321, 323,	space IMS T414 335, 337
388, 390, 443, 445	space IMS T425 262, 265
>< 18, 114, 180, 250, 321, 388, 443 >= 18, 114, 117, 180, 183, 250, 253, 321, 323	space IMS T800 191, 193
>= <b>18</b> , 114, 117, 180, 183, 250, 253, 321, 323,	space IMS T801 129, 132
388, 390, 443, 445	space IMS T805 49, 52
>> <b>18</b> , 114, 180, 250, 321, 388, 443 <b>? 13</b>	<b>AFTER 18, 19</b> , 40, 115, 181, 251, 322, 389, 444
<b>18</b> , 114, 180, 250, 321, 388, 443	<b>ALT 15</b> , 16, 22, 29, 115, 119, 181, 184, 251,
<b>18</b> , 114, 180, 250, 321, 388, 443	255, 322, 325, 389, 391, 444, 446
ABS 117, 183, 253	Alternation construction 14, 15, 16, 29, 40  Analyse
Absolute maximum ratings	IMS T222 418, 419
IMS C004 494	IMS T414 <b>353</b> , 354
IMS C004 494 IMS C011 519	•
11013 0011 319	
IMS C012 543	IMS T425 <b>284</b> , 285
IMS C012 543	IMS T800 <b>215</b> , 216
IMS T222 437	IMS T800 <b>215</b> , 216 IMS T801 <b>155</b> , 156
IMS T222 437 IMS T414 382	IMS T800 <b>215</b> , 216 IMS T801 <b>155</b> , 156 IMS T805 <b>75</b> , 76
IMS T222 437 IMS T414 382 IMS T425 315	IMS T800 <b>215</b> , 216 IMS T801 <b>155</b> , 156 IMS T805 <b>75</b> , 76 <b>AND 18</b> , 114, 180, 250, 321, 388, 443
IMS T222 437 IMS T414 382 IMS T425 315 IMS T800 244	IMS T800 <b>215</b> , 216 IMS T801 <b>155</b> , 156 IMS T805 <b>75</b> , 76 <b>AND 18</b> , 114, 180, 250, 321, 388, 443 ANSI-IEEE 754-1985 17
IMS T222 437 IMS T414 382 IMS T425 315 IMS T800 244 IMS T801 174	IMS T800 <b>215</b> , 216 IMS T801 <b>155</b> , 156 IMS T805 <b>75</b> , 76 <b>AND 18</b> , 114, 180, 250, 321, 388, 443 ANSI-IEEE 754-1985 17 IMS T800 190, 209
IMS T222 437 IMS T414 382 IMS T425 315 IMS T800 244 IMS T801 174 IMS T805 108	IMS T800 <b>215</b> , 216 IMS T801 <b>155</b> , 156 IMS T805 <b>75</b> , 76 <b>AND 18</b> , 114, 180, 250, 321, 388, 443 ANSI-IEEE 754-1985 17 IMS T800 190, 209 IMS T801 128, 149
IMS T222 437 IMS T414 382 IMS T425 315 IMS T800 244 IMS T801 174 IMS T805 108 Access	IMS T800 <b>215</b> , 216 IMS T801 <b>155</b> , 156 IMS T805 <b>75</b> , 76 <b>AND 18</b> , 114, 180, 250, 321, 388, 443 ANSI-IEEE 754-1985 17 IMS T800 190, 209 IMS T801 128, 149 IMS T805 48, 69
IMS T222 437 IMS T414 382 IMS T425 315 IMS T800 244 IMS T801 174 IMS T805 108 Access byte-wide IMS C011 504	IMS T800 <b>215</b> , 216 IMS T801 <b>155</b> , 156 IMS T805 <b>75</b> , 76 <b>AND 18</b> , 114, 180, 250, 321, 388, 443 ANSI-IEEE 754-1985 17 IMS T800 190, 209 IMS T801 128, 149 IMS T805 48, 69 Application
IMS T222 437 IMS T414 382 IMS T425 315 IMS T800 244 IMS T801 174 IMS T805 108 Access byte-wide IMS C011 504 byte-wide IMS C012 530	IMS T800 215, 216 IMS T801 155, 156 IMS T805 75, 76  AND 18, 114, 180, 250, 321, 388, 443  ANSI-IEEE 754-1985 17 IMS T800 190, 209 IMS T801 128, 149 IMS T805 48, 69  Application bidirectional exchange IMS C004 490
IMS T222 437 IMS T414 382 IMS T425 315 IMS T800 244 IMS T801 174 IMS T805 108 Access byte-wide IMS C011 504 byte-wide IMS C012 530 Acknowledge	IMS T800 215, 216 IMS T801 155, 156 IMS T805 75, 76  AND 18, 114, 180, 250, 321, 388, 443  ANSI-IEEE 754-1985 17 IMS T800 190, 209 IMS T801 128, 149 IMS T805 48, 69  Application bidirectional exchange IMS C004 490 bus systems IMS C004 490
IMS T222 437 IMS T414 382 IMS T425 315 IMS T800 244 IMS T801 174 IMS T805 108 Access byte-wide IMS C011 504 byte-wide IMS C012 530 Acknowledge link 39	IMS T800 215, 216 IMS T801 155, 156 IMS T805 75, 76  AND 18, 114, 180, 250, 321, 388, 443  ANSI-IEEE 754-1985 17 IMS T800 190, 209 IMS T801 128, 149 IMS T805 48, 69  Application bidirectional exchange IMS C004 490 bus systems IMS C004 490 drawing coloured text 43
IMS T222 437 IMS T414 382 IMS T425 315 IMS T800 244 IMS T801 174 IMS T805 108 Access byte-wide IMS C011 504 byte-wide IMS C012 530 Acknowledge link 39 link IMS C004 485	IMS T800 215, 216 IMS T801 155, 156 IMS T805 75, 76 AND 18, 114, 180, 250, 321, 388, 443 ANSI-IEEE 754-1985 17 IMS T800 190, 209 IMS T801 128, 149 IMS T805 48, 69 Application bidirectional exchange IMS C004 490 bus systems IMS C004 490 drawing coloured text 43 enhanced controller IMS M212 470
IMS T222 437 IMS T414 382 IMS T425 315 IMS T800 244 IMS T801 174 IMS T805 108 Access byte-wide IMS C011 504 byte-wide IMS C012 530 Acknowledge link 39 link IMS C004 485 link IMS C011 509	IMS T800 215, 216 IMS T801 155, 156 IMS T805 75, 76 AND 18, 114, 180, 250, 321, 388, 443 ANSI-IEEE 754-1985 17 IMS T800 190, 209 IMS T801 128, 149 IMS T805 48, 69 Application bidirectional exchange IMS C004 490 bus systems IMS C004 490 drawing coloured text 43 enhanced controller IMS M212 470 IMS T222 426
IMS T222 437 IMS T414 382 IMS T425 315 IMS T800 244 IMS T801 174 IMS T805 108 Access byte-wide IMS C011 504 byte-wide IMS C012 530 Acknowledge link 39 link IMS C004 485 link IMS C011 509 link IMS C012 535	IMS T800 215, 216 IMS T801 155, 156 IMS T805 75, 76 AND 18, 114, 180, 250, 321, 388, 443 ANSI-IEEE 754-1985 17 IMS T800 190, 209 IMS T801 128, 149 IMS T805 48, 69 Application bidirectional exchange IMS C004 490 bus systems IMS C004 490 drawing coloured text 43 enhanced controller IMS M212 470
IMS T222 437 IMS T414 382 IMS T425 315 IMS T800 244 IMS T801 174 IMS T805 108 Access byte-wide IMS C011 504 byte-wide IMS C012 530 Acknowledge link 39 link IMS C004 485 link IMS C011 509 link IMS C012 535 link IMS C12 434	IMS T800 215, 216 IMS T801 155, 156 IMS T805 75, 76 AND 18, 114, 180, 250, 321, 388, 443 ANSI-IEEE 754-1985 17 IMS T800 190, 209 IMS T801 128, 149 IMS T805 48, 69 Application bidirectional exchange IMS C004 490 bus systems IMS C004 490 drawing coloured text 43 enhanced controller IMS M212 470 IMS T222 426 IMS T414 365
IMS T222 437 IMS T414 382 IMS T425 315 IMS T800 244 IMS T801 174 IMS T805 108 Access byte-wide IMS C011 504 byte-wide IMS C012 530 Acknowledge link 39 link IMS C004 485 link IMS C011 509 link IMS C012 535 link IMS T222 434 link IMS T414 379	IMS T800 215, 216 IMS T801 155, 156 IMS T805 75, 76  AND 18, 114, 180, 250, 321, 388, 443  ANSI-IEEE 754-1985 17 IMS T800 190, 209 IMS T801 128, 149 IMS T805 48, 69  Application bidirectional exchange IMS C004 490 bus systems IMS C004 490 drawing coloured text 43 enhanced controller IMS M212 470 IMS T222 426 IMS T414 365 IMS T425 296
IMS T222 437 IMS T414 382 IMS T425 315 IMS T800 244 IMS T801 174 IMS T805 108 Access byte-wide IMS C011 504 byte-wide IMS C012 530 Acknowledge link 39 link IMS C004 485 link IMS C011 509 link IMS C012 535 link IMS T222 434 link IMS T414 379	IMS T800 215, 216 IMS T801 155, 156 IMS T805 75, 76  AND 18, 114, 180, 250, 321, 388, 443  ANSI-IEEE 754-1985 17 IMS T800 190, 209 IMS T801 128, 149 IMS T805 48, 69  Application bidirectional exchange IMS C004 490 bus systems IMS C004 490 drawing coloured text 43 enhanced controller IMS M212 470 IMS T222 426 IMS T414 365 IMS T425 296 IMS T800 227

multiple control IMS C004 490	IMS T425 262, 270
winchester controller IMS M212 469, 470	IMS T800 190, 198
Architecture 28, 29	IMS T801 128, 137
internal 31	IMS T805 48, 57
rationale 9	performance 118, 184, 254, 324
Arithmetic	two-dimensional 43
multiple length 116, 182, 252, 322, 389, 444	BOOL 17
operation IMS T222 403 operation IMS T414 337	Boolean
operation IMS T425 265	expression 119, 184, 255, 325, 391, 446 operator 114, 180, 250, 321, 388, 443
operation IMS T800 193	BootFromRom
operation IMS T801 132	IMS T222 416, 418
operation IMS T805 52	IMS T414 <b>351</b> , 353
operator 114, 117, 180, 183, 250, 253, 321,	IMS T425 <b>282</b> , 284
388, 443	IMS T800 <b>213</b> , 215
Array 17	IMS T801 153, 155
assignment 119, 184, 255, 325, 391, 446	IMS T805 <b>73</b> , 75
byte 19	Bootstrap 24, 25
of disk controllers IMS M212 471	address IMS T222 418
of processes 16	address IMS T414 353
of transputers IMS M212 471	address IMS T425 284
type 17	address IMS T800 215
variable 114, 180, 250, 321, 388, 443	address IMS T801 155
ASCII 19	address IMS T805 75
ASHIFTLEFT 116, 182, 252 ASHIFTRIGHT 116, 182, 252	code IMS T222 420 code IMS T414 355
Assignment 13, 29, 114, 117, 180, 183, 250,	code IMS T414 333
253, 321, 388, 443	code IMS T800 217
array 119, 184, 255, 325, 391, 446	code IMS T801 <b>157</b>
process 12, <b>13</b>	code IMS T805 77
•	IMS M212 467
Bandwidth	IMS T222 416, 418
memory 43	IMS T414 <b>351</b> , 353, 376
memory IMS T222 405	IMS T425 282, 284, 308
memory IMS T414 339	IMS T800 213, 215, 238
memory IMS T425 267	IMS T801 153, 155
memory IMS T800 195 memory IMS T801 134	IMS T805 <b>73</b> , <b>75</b> , 94 program IMS T222 432
memory IMS T805 54	program IMS T801 167
Barrel shifter 42	Bootstrapping
Behaviour	IMS T425 <b>286</b>
logical 7, 20, 22	IMS T805 77
physical 7	Brackets 18
Benchmark	Break point
LINPACK 42	IMS T425 <b>286</b>
speed 41	IMS T805 <b>77</b>
Whetstone 41	Buffer
Bit	input IMS C011 517
counting performance 118, 184, 254, 324	input IMS C012 541
data <b>39</b> operator 114, 180, 250, 321, 388, 443	link 24 output IMS C011 517
reversal performance 118, 184, 254, 324	output IMS C011 517
start 39	Bus 31
stop 39	IMS C011 514
Bit-bit 43	IMS C012 538
BITCOUNT 118, 184, 254, 324	Byte
BITREVNBIT 118, 184, 254, 324	access IMS C011 504
BITREVWORD 118, 184, 254, 324	access IMS C012 530
Block move 43	access IMS T222 423, 428
conditional 43	access IMS T801 160

address IMS T222 420, 423 address IMS T414 355 address IMS T425 286 address IMS T800 217 address IMS T801 157, 160 address IMS T805 77  BYTE 17	internal IMS T800 198 internal IMS T801 137 internal IMS T805 57 link 23 link IMS C004 485 link IMS C011 509 link IMS C012 535
C 23	link IMS T222 434 link IMS T414 379
Capacitive load 9	link IMS T425 312
CapMinus	link IMS T800 241
IMS C004 482	link IMS T801 171
IMS C011 506 IMS C012 532	link IMS T805 105 memory 23
IMS T222 415	occam 23
IMS T414 349	output 29
IMS T425 280	process 29
IMS T800 211	Characteristics
IMS T801 151 IMS T805 71	AC timing IMS C004 496 AC timing IMS C011 521
CapPlus	AC timing IMS C012 545
IMS C004 482	AC timing IMS T222 439
IMS T222 415	AC timing IMS T414 384
IMS T414 349 IMS T425 280	AC timing IMS T425 317 AC timing IMS T800 246
IMS T800 211	AC timing IMS T801 176
IMS T801 151	AC timing IMS T805 110
IMS T805 71	DC electrical IMS C004 494, 495
CASE 16 CHAN OF 17	DC electrical IMS C011 519, 520 DC electrical IMS C012 543, 544
protocol 114, 180, 250, 321, 388, 443	DC electrical IMS T222 437, 438
Channel 8, 12, 13, 15, 17, 20, 23, 29, 114, 180,	DC electrical IMS T414 382, 383
250, 321, 388, 443	DC electrical IMS T425 315, 316
communication 36, 38 disk hardware IMS M212 466	DC electrical IMS T800 244, 245 DC electrical IMS T801 174, 175
empty 36	DC electrical IMS T805 108, 109
event IMS T222 433	CLIP2D 43, 118, 184, 254, 324
event IMS T414 378	Clock 17, 25
event IMS T425 310 event IMS T800 240	input 24, 25 input, internal IMS C004 482
event IMS T801 169	input, internal IMS C004 482 input, internal IMS C011 506
event IMS T805 103	input, internal IMS C012 532
external 36	input, internal IMS T222 415
external IMS T222 408 external IMS T414 342	input, internal IMS T414 349 input, internal IMS T425 280
external IMS T425 270	input, internal IMS T425 <b>280</b> input, internal IMS T800 <b>211</b>
external IMS T800 198	input, internal IMS T801 151
external IMS T801 137	input, internal IMS T805 71
external IMS T805 57 IMS T222 407	internal 24 link 25
IMS T414 341	link IMS C004 485
IMS T425 269	link IMS C011 509
IMS T800 197	link IMS C012 535
IMS T801 136	link IMS T222 434
IMS T805 56 input 29	link IMS T414 379 link IMS T425 312
internal 36	link IMS T800 241
internal IMS T222 408	link IMS T801 171
internal IMS T414 342	link IMS T805 105
internal IMS T425 270	multiple IMS C004 482

multiple IMS C011 506	IMS T425 267
multiple IMS C012 532 multiple IMS T222 415	IMS T800 195 IMS T801 134
multiple IMS T414 349	IMS T805 54
multiple IMS T425 280	Colour
multiple IMS T800 211	display 43
multiple IMS T801 151	graphics 34
multiple IMS T805 71	text example 43
phase 11	Communication 8, 9, <b>10</b> , 28, 31, <b>36</b>
processor 40	bandwidth <b>9</b>
processor IMS T222 408	channel 12, 36, 38
processor IMS T414 342	construction 12, 15
processor IMS T425 270	contention 9
processor IMS T800 198 processor IMS T801 137	external 38 frequency 11, 24
processor IMS T805 57	IMS T222 407
stability IMS C004 482	IMS T414 341
stability IMS C011 506	IMS T425 269
stability IMS C012 532	IMS T800 197
stability IMS T222 415	IMS T801 136
stability IMS T414 349	IMS T805 56
stability IMS T425 280	interface 10
stability IMS T800 211	internal 36
stability IMS T801 151	language 23
stability IMS T805 71	link 9, 39
timer 19 timer IMS T222 <b>408</b>	parallel IMS C011 504 parallel IMS C012 530
timer IMS T414 342	process IMS T222 407
timer IMS T425 <b>270</b>	process IMS T414 <b>341</b>
timer IMS T800 198	process IMS T425 <b>269</b>
timer IMS T801 137	process IMS T800 197
timer IMS T805 57	process IMS T801 136
transputer 11	process IMS T805 56
Clockin	speed 30
IMS C004 482	Comparison operator 114, 117, 180, 183, 250
IMS C011 506	253, 321, 388, 443
IMS C012 532 IMS T222 415	Compatibility IMS T425 <b>263</b>
IMS T414 349	IMS T805 49
IMS T425 <b>280</b>	Concept 29
IMS T800 211	Concurrency 7, 12, 28
MS T801 151	IMS T222 406
IMS T805 71	IMS T414 340
period IMS T222 422	IMS T425 <b>268</b>
period IMS T414 357	IMS T800 196
period IMS T425 288	IMS T801 135
period IMS T800 219	IMS T805 55
period IMS T801 162 period IMS T805 82	internal 12
period IMS T805 82 skew IMS C011 507	support 34 Concurrent
Code	FPU/CPU operation 41
function/operation IMS T222 409	process 12, 14, 15, 22, 29
function/operation IMS T414 343	systems 12
function/operation IMS T425 272	Conditional construction 14, 15
function/operation IMS T800 200	Configuration
function/operation IMS T801 139	coding IMS T414 371
function/operation IMS T805 59	coding IMS T425 302
Coding efficiency	coding IMS T800 233
IMS T222 405	coding IMS T805 101
IMS T414 339	memory IMS T222 418

memory IMS T414 353, 357, 366	Conversion
memory IMS T425 284, 288, 297	INT, REAL 117, 183, 253
memory IMS T800 215, 219, 228	<b>REAL, INT</b> 117, 183, 253
memory IMS T801 155	CPU <b>31</b>
memory IMS T805 75, 79, <b>81, 96</b>	concurrent operation 41
memory, external IMS T414 366, <b>368</b> , 369,	register 31, 32
370, 372	CRC
memory, external IMS T425 297, <b>299</b> , 300,	IMS M212 465, 467
301, 303	IMS T425 262
memory, external IMS T800 228, <b>230</b> , 231,	IMS T800 191
232, 234	IMS T801 129
memory, external IMS T805 98, 99, 100, 102	IMS T805 49
memory, internal IMS T414 366	performance 118, 184, 254, 324
memory, internal IMS T425 297	CRCBYTE 118, 184, 254, 324
memory, internal IMS T800 228	CRCWORD 118, 184, 254, 324
memory, internal IMS T805 96	Cyclic redundancy
program 20	IMS M212 465, 467
Connection Link IMS COOA 485	IMS T425 262
link IMS C004 485	IMS T800 191
link IMS C011 509	IMS T801 129
link IMS C012 535	IMS T805 49
link IMS T222 434 link IMS T414 379	performance 118, 184, 254, 324
link IMS T425 312	D0-7
link IMS T800 241	IMS C011 508, <b>514</b>
link IMS T801 171	IMS C012 534, <b>538</b>
link IMS T805 105	DABS 117, 183, 253
Constant 115, 181, 251, 322, 389, 444	Data
subscript 114, 180, 250, 321, 388, 443	bit <b>39</b>
value <b>33</b>	bus IMS T222 422, <b>423</b>
Construction 14, 29, 115, 181, 251, 322, 389,	bus IMS T414 359
444	bus IMS T425 290
alternation 14, 15, 16, 29, 40	bus IMS T800 221
communication 15	bus IMS T801 159, 160
conditional 14, 15	bus IMS T805 80
parallel 12, <b>14</b> , <b>20</b> , 29, 35	link <b>39</b>
parallel IMS T222 407	link IMS C004 485
parallel IMS T414 341	link IMS C011 509
parallel IMS T425 269	link IMS C012 535
parallel IMS T800 197	link IMS T222 434
parallel IMS T801 136	link IMS T414 379
parallel IMS T805 56	link IMS T425 312
performance 118, 184, 254, 325, 391, 446	link IMS T800 241
repetition 16	link IMS T801 171
replication 16	link IMS T805 105
selection 16	rate 24, 39
sequential 12, 14, 15, 29	rate IMS T222 400
Context switch	rate IMS T225 454
IMS T425 286	rate IMS T414 335 rate IMS T425 262
IMS T805 77	rate IMS T425 262 rate IMS T800 191
Control byte IMS T222 418	rate IMS 1800 191
byte IMS T222 418 byte IMS T414 351, 353	rate IMS 1805 49
byte IMS T414 331, 333 byte IMS T425 284	rate link IMS T222 434
byte IMS T800 215	rate link IMS T414 379
byte IMS 7800 213	rate link IMS T425 312
byte IMS 7805 <b>75</b>	rate link IMS T800 241
link IMS C004 484	rate link IMS T801 171
logic IMS M212 466	rate link IMS T805 105
	read IMS C011 <b>518</b>

read IMS C012 542	IMS T414 358
separation IMS M212 465	IMS T425 289
serial 39	IMS T800 220
structure 33	IMS T801 159
structure IMS T222 404	IMS T805 79
structure IMS T414 338	DisableIntRAM
structure IMS T425 266	IMS T222 420
structure IMS T800 194	IMS T414 355
structure IMS T801 133 structure IMS T805 53	IMS T425 286 IMS T800 217
transfer 15	IMS T805 77
value 32	Disk
value IMS T222 404	command IMS M212 467, 468
value IMS T414 338	compression/decompression IMS M212 471
value IMS T425 266	controller IMS M212 464, 465
value IMS T800 194	cylinder IMS M212 467
value IMS T801 133	drive selection IMS M212 464
value IMS T805 53 write IMS C011 518	encryption/decryption IMS M212 471 floppy IMS M212 464, <b>467</b>
write IMS C012 542	format IMS M212 467
Data Present	head IMS M212 467
IMS C011 514, 517, 518	head position IMS M212 464
IMS C012 538, 541, 542	interleave IMS M212 468
Declaration 17, 114, 180, 250, 321, 388, 443	management IMS M212 471
Decoupling	parameter IMS M212 467
IMS C004 482 IMS C011 506	port IMS M212 465 programming interface IMS M212 467
IMS C011 300	SA400/450 IMS M212 465, 467
IMS T222 415	sector IMS M212 467
IMS T414 <b>349</b>	ST506/412 IMS M212 465, 467
IMS T425 280	status IMS M212 464
IMS T800 211	winchester IMS M212 464, <b>467</b>
IMS T801 151	DMA at report IMS T222 421 422
IMS T805 <b>71</b> Delay	at reset IMS T222 431, 432 at reset IMS T414 377
input 19	at reset IMS T425 309
timer 19	at reset IMS T800 239
Deschedule <b>35</b> , 37, 38	at reset IMS T801 167
IMS T222 <b>406</b> , 407, 408	at reset IMS T805 95
IMS T414 <b>340</b> , 341, 342	IMS T222 423, 424, <b>431</b>
IMS T425 <b>268</b> , 269, 270	IMS T414 358, <b>376</b>
IMS T800 <b>196</b> , 197, 198 IMS T801 <b>135</b> , 136, 137	IMS T425 289, <b>308</b> IMS T800 220, <b>238</b>
IMS T805 <b>55</b> , 56, 57	IMS T801 159, <b>161</b> , <b>167</b>
point IMS T222 407, 410, 418	IMS T805 79, <b>94</b>
point IMS T414 341, 344, 353	operation IMS T222 431
point IMS T425 269, 273, 284	operation IMS T414 377
point IMS T800 197, 201, 215	operation IMS T425 309
point IMS T801 136, 140, 155	operation IMS T800 239
point IMS T805 <b>56</b> , <b>60</b> , 75 Device 29	operation IMS T801 167 operation IMS T805 95
Direct function 33	DRAW2D 43, 118, 184, 254, 324
IMS T222 404	DSQRT 117, 183, 253
IMS T414 338	
IMS T425 <b>266</b>	ECC
IMS T800 194	IMS M212 465, 467
IMS T801 133	Efficiency 34
IMS T805 53 Direct memory access	Electrical AC timing characteristics IMS C004 496
IMS T222 423	AC timing characteristics IMS C011 521
	5

AC timing characteristics IMS C012 545	power up IMS T222 419
AC timing characteristics IMS T222 439	power up IMS T414 354
AC timing characteristics IMS T414 384	power up IMS T425 285
AC timing characteristics IMS T425 317	power up IMS T800 216
AC timing characteristics IMS T800 246	power up IMS T801 156
AC timing characteristics IMS T801 176	power up IMS T805 76
AC timing characteristics IMS T805 110	Error 25
DC characteristics IMS C004 494, 495	analysis 21
DC characteristics IMS C011 519, 520	analysis IMS T222 419
DC characteristics IMS C012 543, 544	analysis IMS T414 354
DC characteristics IMS T222 437, 438	analysis IMS T425 285
DC characteristics IMS T414 382, 383	analysis IMS T800 216
DC characteristics IMS T425 315, 316	analysis IMS T801 156
DC characteristics IMS T800 244, 245	analysis IMS T805 76
DC characteristics IMS T801 174, 175	circuit IMS T222 419
DC characteristics IMS T805 108, 109	circuit IMS T414 354
operating conditions IMS C004 494	circuit IMS T425 285
operating conditions IMS C011 519	circuit IMS T800 216
operating conditions IMS C012 543	circuit IMS T801 156
operating conditions IMS T222 437	circuit IMS T805 76
operating conditions IMS T414 382	correcting code 34
operating conditions IMS T425 315	correcting code IMS M212 465, 467
operating conditions IMS T800 244	expression check 115, 181, 251, 322, 389,
operating conditions IMS T801 174	444
operating conditions IMS T805 108	floating point IMS T800 210
specification 24	floating point IMS T801 150
EMI	floating point IMS T805 70
IMS T222 <b>422</b>	handling 21
IMS T414 <b>357</b>	IMS M212 467
IMS T425 <b>288</b>	languages 21
	•
IMS T800 219	reset IMS T222 419
IMS T801 159	reset IMS T414 354
IMS T805 79	reset IMS T425 285
EnableJ0Break	reset IMS T800 216
IMS T425 284	reset IMS T801 156
_IMS T805 75	_ reset IMS T805 76
Equivalent circuit	Errorin
IMS C004 495	IMS T425 <b>285</b>
IMS C011 520	IMS T800 216
IMS C012 544	_ IMS_T805 <b>76</b>
IMS T222 438	ErrorOut
IMS T414 383	IMS T801 156
IMS T425 316	Evaluation
IMS T800 245	expression IMS T222 403, 405
IMS T801 175	expression IMS T414 <b>337</b> , 339
IMS T805 109	expression IMS T425 <b>265</b> , 267
Erastosthenes 119, 184, 255, 325, 391, 446	expression IMS T800 193, 195
Error 25	expression IMS T801 132, 134
IMS T222 418, <b>419</b>	expression IMS T805 52, 54
IMS T414 353, <b>354</b>	stack 31, <b>32</b> , 36
IMS T425 284, <b>285</b>	stack IMS T222 403, 407, 408
IMS T800 215, <b>216</b>	stack IMS T414 337, 341, 342
IMS T805 75, <b>76</b>	stack IMS T425 265, 269, 270
Error 25	stack IMS T800 193, 197, 198
IMS T222 419	stack IMS T801 132, 136, 137
IMS T414 <b>354</b>	stack IMS T805 <b>52</b> , 56, 57
IMS T425 <b>285</b>	Event 16, 26
IMS T800 216	IMS T222 433
IMS T801 155, <b>156</b>	IMS T414 378
IMS 1805 1805 1805	IMS T414 370

IMS T800 240	Floating point 28, 40
IMS T801 169	address 41
IMS T805 103	co-processor 42
EventAck	comparison 42
IMS T222 433	concurrency IMS T800 209
IMS T414 378	concurrency IMS T801 149
IMS T425 310	concurrency IMS T805 69
IMS T800 240	concurrent operation 41
IMS T801 169	datapath 42
IMS T805 103	design 40, 42
EventReq	division 42
IMS T222 419, <b>433</b> IMS T414 354, <b>378</b>	double length IMS T800 209 double length IMS T801 149
IMS T425 285, <b>310</b>	double length IMS T805 69
IMS T800 216, <b>240</b>	error IMS T800 <b>210</b>
IMS T801 156, <b>169</b>	error IMS T801 <b>150</b>
IMS T805 76, <b>103</b>	error IMS T805 <b>70</b>
EventWaiting	functions 117, 183, 253
IMS T425 310	instruction 34, 40
IMS T801 169	microcode 42
IMS T805 103	multiplication 41, 42
Example	normalise IMS T800 209
drawing coloured text 43	normalise IMS T801 149
instruction set IMS T222 409	normalise IMS T805 69
instruction set IMS T414 343	operand 41
instruction set IMS T425 272	performance 113, <b>117</b> , 179, <b>183</b> , 249, <b>253</b> ,
instruction set IMS T800 200	323, 390, 445
instruction set IMS T801 139	processor 31, 40
instruction set IMS T805 59	processor IMS T800 190, <b>209</b>
Execution	processor IMS T801 128, 149
instruction IMS T222 404	processor IMS T805 48, <b>69</b>
instruction IMS T414 338	register 31
instruction IMS T425 266	rounding IMS T800 209
instruction IMS T800 194	rounding IMS T801 149
instruction IMS T801 133	rounding IMS T805 69
instruction IMS T805 53	selector sequence IMS T800 200, 209
Expression 12, <b>18</b> , 29, 114, 115, 117, 180, 181, 183, 250, 251, 253, 321, 322, 388, 389,	selector sequence IMS T801 139, 149 selector sequence IMS T805 59, 69
443, 444	single length IMS T800 209
evaluation IMS T222 <b>403</b> , 405	single length IMS T801 149
evaluation IMS T414 <b>337</b> , 339	single length IMS T805 69
evaluation IMS T425 <b>265</b> , 267	stack IMS T800 209
evaluation IMS T800 193, 195	stack IMS T801 149
evaluation IMS T801 <b>132</b> , 134	stack IMS T805 69
evaluation IMS T805 52, 54	Floating point numbers 17
subscript 114, 180, 250, 321, 388, 443	FM
External	IMS M212 465
memory interface IMS T222 422	FOR 16
memory interface IMS T414 357	Fortran 23
memory interface IMS T425 288	FPU (see Floating point) 31, 69, 149, 209
memory interface IMS T800 219	FP'Error
memory interface IMS T801 159	IMS T800 210
memory interface IMS T805 79	IMS T801 150
memory performance 118, 184, 254, 324,	IMS T805 <b>70</b> FRACMUL 116, 182, 252
<b>391, 446</b> registers 19	Frequency
rogistors 19	changer IMS C011 504
Factorial 18	Clockin IMS C004 482
FALSE 18	Clockin IMS C011 506
Flash multiplier 42	Clockin IMS C012 532

Clockin IMS T222 415 Clockin IMS T414 349 Clockin IMS T425 280 Clockin IMS T800 211 Clockin IMS T801 151 Clockin IMS T805 71 link 24	IMS T800 215, 216 IMS T801 155, 156 IMS T805 75, 76 HaltOnError IMS T222 418 IMS T414 353 IMS T425 284
modulation IMS M212 <b>465</b> Function <b>18</b> , 115, 181, 251, 322, 389, 444	IMS T800 215 IMS T805 75
code 32 code IMS T222 404, 409	HaltOnError IMS T222 <b>419</b>
code IMS T414 338, 343	IMS T414 354
code IMS T425 266, 272 code IMS T800 194, 200	IMS T425 <b>285</b> IMS T800 <b>216</b>
code IMS T801 133, 139	IMS T801 155, <b>156</b>
code IMS T805 53, 59 direct 33	IMS T805 <b>76</b> Handshake 12
direct IMS T222 404	event IMS T222 433
direct IMS T414 338	event IMS T414 378
direct IMS T425 266 direct IMS T800 194	event IMS T425 310 event IMS T800 240
direct IMS T801 133	event IMS T801 169
direct IMS T805 53	event IMS T805 103
indirect 34 indirect IMS T222 405	parallel IMS C011 504, 512, 513 Hardware <b>9</b>
indirect IMS T414 339	channel IMS M212 466
indirect IMS T425 267 indirect IMS T800 195	IMS M212 465, 468 Harness 12, 23
indirect IMS T801 134	1411000 12, 20
indirect IMS T805 54	10-7
prefix <b>33</b> , 34 prefix IMS T222 <b>404</b>	IMS C011 512 IAck
prefix IMS T414 338	IMS C011 508, 512
prefix IMS T425 266 prefix IMS T800 194	<b>IF 15</b> , 16, 29, 115, 119, 181, 184, 251, 255
prefix IMS T800 194	322, 325, 389, 391, 444, 446 Implementation
prefix IMS T805 53	hard-wired 9
<b>FUNCTION 18</b> , 114, 117, 180, 183, 250, 253, 321, 388, 443	hardware 9 link 10
321, 333, 443	occam 9
GND 24	program 12
IMS C004 482 IMS C011 506	IMS B005 470 IMS C004 <b>480</b>
IMS C012 532	IMS C011 504
IMS T222 415 IMS T414 349	IMS C012 530 IMS M212 464
IMS T425 280	IMS T222 400, 454
IMS T800 211	IMS T225 454
IMS T801 151 IMS T805 71	IMS T414 335 IMS T425 262
Graphics 43	IMS T800 190
support IMS T425 262	IMS T805 48
support IMS T800 190 support IMS T801 128	Indirect function 34 IMS T222 405
support IMS T805 48	IMS T414 339
Halt 21	IMS T425 <b>267</b> IMS T800 <b>195</b>
IMS T222 418, 419	IMS T800 193
IMS T414 353, 354	IMS T805 54
IMS T425 284, 285	

Indirection code	debugging IMS T805 61
instruction IMS T800 209	descheduling IMS T222 410
instruction IMS T801 149	descheduling IMS T414 344
instruction IMS T805 69	descheduling IMS T425 273
Input 12, 13, 19, 25, 29, 114, 117, 180, 183,	descheduling IMS T800 201
250, 253, 321, 388, 443	descheduling IMS T801 140
buffer IMS C011 517	descheduling IMS T805 60
buffer IMS C012 541	description 32, 33, 34, 35, 36, 40
channel 29	description IMS T222 403, <b>404</b> , 405, 407, 408
	description IMS T414 337, <b>338</b> , 339, 341, 342
clock 24, 25	
clock IMS C004 482	description IMS T425 265, <b>266</b> , 267, 269, 270
clock IMS C011 506	description IMS T800 193, <b>194</b> , 195, 197,
clock IMS C012 532	198, 200, <b>209</b> , <b>210</b>
clock IMS T222 415	description IMS T801 132, 133, 134, 136,
clock IMS T414 349	137, 139, <b>149</b> , <b>150</b>
clock IMS T425 280	description IMS T805 52, <b>53</b> , 54, 56, 57, 59,
clock IMS T800 211	69, 70
clock IMS T801 151	error IMS T222 410
clock IMS T805 71	error IMS T414 344
link IMS C004 485	error IMS T425 274
link IMS C011 509	error IMS T800 202
link IMS C012 535	error IMS T801 141
link IMS T222 418, <b>434</b>	error IMS T805 61
link IMS T414 353, <b>379</b>	execution IMS T222 404
link IMS T425 284, <b>312</b>	execution IMS T414 338
link IMS T800 215, <b>241</b>	execution IMS T425 266
link IMS T801 155, <b>171</b>	execution IMS T800 194
link IMS T805 75, <b>105</b>	execution IMS T801 133
pins 24	execution IMS T805 53
port IMS C011 512	floating point 40
•	floating point error IMS T800 202
process 12, 13, 15	
process IMS T222 408	floating point error IMS T801 141
process IMS T414 342	floating point error IMS T805 61
process IMS T425 270	format IMS T222 404
process IMS T800 198	format IMS T414 338
process IMS T801 137	format IMS T425 266
process IMS T805 57	format IMS T800 194
register IMS C011 514, 517	format IMS T801 133
register IMS C012 538, 541	format IMS T805 53
. • • • • • • • • • • • • • • • • • • •	
timer 115, 181, 251, 322, 389, 444	IMS T222 403
voltage 24	IMS T414 337
InputInt	IMS T425 265
IMS C011 508, <b>517</b> , 518	IMS T800 193
IMS C012 534, <b>541</b> , 542	IMS T801 132
Instruction	IMS T805 52
arithmetic IMS T222 405	indirection code IMS T800 209
arithmetic IMS T414 339	indirection code IMS T801 149
arithmetic IMS T425 267	indirection code IMS T805 69
arithmetic IMS T800 195	logical IMS T222 405
arithmetic IMS T801 134	logical IMS T414 339
arithmetic IMS T805 54	logical IMS T425 267
comparison IMS T222 405	logical IMS T800 195
comparison IMS T414 339	logical IMS T801 134
comparison IMS T425 267	logical IMS T805 54
comparison IMS T800 195	memory relative IMS T222 404
comparison IMS T801 134	memory relative IMS T414 338
comparison IMS T805 54	memory relative IMS T425 266
debugging IMS T225 457	memory relative IMS T800 194
debugging IMS T425 274	memory relative IMS T801 133
debugging IMS T801 141	memory relative IMS T805 53

operation 33	memory 10, 24
pointer 35	memory IMS T222 403, 418, <b>422</b>
pointer IMS T222 407	memory IMS T414 337, 353, <b>357</b>
pointer IMS T414 341	memory IMS T425 265, 284, <b>288</b>
pointer IMS T425 269	memory IMS T800 193, 215, <b>219</b>
pointer IMS T800 197	memory IMS T801 132, 155, <b>159</b>
pointer IMS T801 136	memory IMS T805 52, 75, <b>79</b>
pointer IMS T805 56	parallel IMS C011 514
prefetch 34	parallel IMS C012 538
single byte IMS T222 404	peripheral IMS M212 465
single byte IMS T414 338	SCSI IMS M212 472
single byte IMS T425 266	serial data IMS M212 464
single byte IMS T800 194	Interrupt 12, 16, 31
single byte IMS T801 133 single byte IMS T805 53	IMS C011 504, 508 IMS C012 530
	latency IMS T222 407
workspace IMS T222 407 workspace IMS T414 341	latency IMS T414 341
workspace IMS T425 269	latency IMS T425 269
workspace IMS T800 197	latency IMS T800 197
workspace IMS T801 136	latency IMS T801 136
workspace IMS T805 56	latency IMS T805 56
Instruction set 12, 32, 411	latency performance 119, 185, 255, 325, 392
design 32	447
example IMS T222 409	Interrupt Enable
example IMS T414 343	IMS C011 517, 518
example IMS T425 272	IMS C012 541, 542
example IMS T800 200	IntSaveLoc
example IMS T801 139	IMS T222 420
example IMS T805 59	IMS T414 355
IMS T222 403, 404, 405, 409	IMS T425 <b>286</b>
IMS T225 457	IMS T800 217
IMS T414 337, 338, 339, 343, <b>345</b>	IMS T801 157
IMS T425 265, 266, 267, 272, <b>275</b>	IMS T805 <b>77</b>
IMS T800 193, 194, 195, 200, <b>203</b>	IPtr
IMS T801 132, 133, 134, 139, <b>142</b>	IMS T425 286
IMS T805 52, 53, 54, 59, <b>62</b>	IMS T805 77
INT 17, 114, 180, 250, 321, 388, 443	IS 18
INT16 17	IValid
INT32 17	IMS C011 512
conversion 117, 183, 253 INT64 17	Language 7, 12, 22, 23
conversion 117, 183, 253	communication 23
Integer performance 113, 179, 249, 320, 387,	error 21
442	IMS T222 404, 405
Integrated memory 29	IMS T414 338, 339
Interface	IMS T425 266, 267
application specific 25	IMS T800 194, 195
communication 10	IMS T801 133, 134
disk controller IMS M212 468	IMS T805 53, 54
disk programming IMS M212 467	Latency 119, 185, 255, 325, 392, 447
link 11, 39	interrupt IMS T222 407
link IMS C004 485	interrupt IMS T414 341
link IMS C011 509	interrupt IMS T425 269
link IMS C012 535	interrupt IMS T800 197
link IMS T222 434	interrupt IMS T801 136
link IMS T414 379	interrupt IMS T805 56
link IMS T425 312	process IMS T222 407
link IMS T800 241	process IMS T414 341
link IMS T801 171	process IMS T425 269
link IMS T805 105	process IMS T800 197

process IMS T801 136	message 10
process IMS T805 56	Mode 1 IMS C011 507, 512
Link 10, 24, 25	Mode 2 IMS C011 507, <b>514</b>
acknowledge 10, 39	mode select IMS C011 507
acknowledge IMS C011 513, 518	output IMS T222 418
acknowledge IMS C012 542	output IMS T414 353
acknowlege overlap 39	output IMS T425 284
adaptor 11, 25	output IMS T800 215
adaptor IMS C011 504	output IMS T801 155
adaptor IMS C012 530	output IMS T805 75
bootstrap ID IMS T222 418	packet 39
bootstrap ID IMS T414 353	parallel adaptor IMS C011 504
bootstrap ID IMS T425 284	parallel adaptor IMS C012 530
bootstrap ID IMS T800 215	peek IMS T222 418
bootstrap ID IMS T801 155	peek IMS T414 353
bootstrap IMS T805 75	peek IMS T425 284
bootstrap IMS T222 416, 418 bootstrap IMS T414 351	peek IMS T800 215 peek IMS T801 155
bootstrap IMS T425 282, 284	peek IMS 7805 75
bootstrap IMS T800 213, 215	performance 391
bootstrap IMS T801 153, 155	poke IMS T222 418
bootstrap IMS T805 73, 75	poke IMS T414 353
buffer 24	poke IMS T425 284
buffer delays 24	poke IMS T800 215
buffer IMS T222 418	poke IMS T801 155
buffer IMS T414 353	poke IMS T805 75
buffer IMS T425 284	programmable switch IMS C004 480
buffer IMS T800 215	protocol 10, 11, 39
buffer IMS T801 155	signal 24
buffer IMS T805 75	speed 39
channel 23	speed IMS C011 504
clock 25	speed IMS C012 530
communication 9, 39	speed select IMS C011 507
control IMS C004 484 crossbar switch IMS C004 480	standard 20, 24
data 10, 39	start bit <b>10</b> static IMS T222 404
data IMS C011 513, 514, 517, 518	static IMS T414 338
data IMS C012 538, 541, 542	static IMS T414 666
disk IMS M212 466	static IMS T800 194
frequency 24	static IMS T801 133
implementation 10	static IMS T805 53
IMS C004 485	stop bit 10
IMS C011 504, <b>509</b>	transfer IMS T222 418
IMS C012 530, <b>535</b>	transfer IMS T414 353
IMS T222 407, 418, <b>434</b>	transfer IMS T425 284
IMS T414 341, 353, <b>379</b>	transfer IMS T800 215
IMS T425 269, 284, <b>312</b>	transfer IMS T801 155
IMS T800 197, 215, <b>241</b>	transfer IMS T805 75
IMS T801 136, 155, <b>171</b>	transmission_39
IMS T805 56, 75, <b>105</b>	transputer 12
input IMS C011 518	wiring 30
input IMS C012 542 input IMS T222 418	Link switch
input IMS T222 418 input IMS T414 353	bit time delay IMS C004 489 configuration IMS C004 480, <b>489</b>
input IMS T414 555	configuration message IMS C004 489
input IMS T800 215	implementation IMS C004 489
input IMS T801 155	multiplexors IMS C004 489
input IMS T805 75	Linked list 35
interface 38, <b>39</b>	IMS T222 406
interface register 38	IMS T414 340

IMS T425 <b>268</b>	instruction IMS T805 53
IMS T800 196	Logical 100 Made 107
IMS T801 135	address IMS M212 467
IMS T805 55 LinkIn	behaviour <b>22</b> operation IMS T222 403
IMS C004 485	operation IMS T414 337
IMS C011 508, <b>509</b>	operation IMS T425 265
IMS C012 534, <b>535</b>	operation IMS T800 193
IMS T222 434	operation IMS T801 132
IMS T414 379	operation IMS T805 52
IMS T425 312	LONGADD 116, 182, 252
IMS T800 <b>241</b>	LONGDIFF 116, 182, 252
IMS T801 171	LONGDIV 116, 182, 252
IMS T805 105	LONGPROD 116, 182, 252
LinkOut	LONGSUB 116, 182, 252
IMS C004 485 IMS C011 508, 509	LONGSUM 116, 182, 252 Loop 16
IMS C011 500, 509	100p 10
IMS T222 434	Мар
IMS T414 379	memory 23, 25
IMS T425 312	process 12
IMS T800 <b>241</b>	MemA0-15
IMS T801 171	IMS T222 <b>423</b> , 431
IMS T805 105	MemA2-31
LinkSpecial	IMS T801 159, <b>160</b> , 164, 167
IMS T222 434	MemAD2-31
IMS T414 <b>379</b> IMS T425 <b>312</b>	IMS T414 358, <b>359</b> , 366, 373 IMS T425 289, <b>290</b> , 297, 305
IMS 1423 312 IMS T800 241	IMS T800 220, <b>221</b> , 228, 235
IMS T801 171	IMS T805 79, <b>80</b> , 91, 96
IMS T805 105	MemBAcc
LINPACK benchmark 42	IMS T222 428
List	MemConfig
linked IMS T222 406	IMS T414 366, 368
linked IMS T414 340	IMS T425 <b>297</b> , 299
linked IMS T425 268	IMS T800 228, 230
linked IMS T800 196 linked IMS T801 135	IMS T805 <b>81</b> , <b>96</b> , 98 <b>MemD0-15</b>
linked IMS T805 <b>55</b>	IMS T222 <b>423</b> , 428, 431
process IMS T222 406, 407	MemD0-31
process IMS T414 340, 341	IMS T801 159, 160, 163
process IMS T425 268, 269	MemGranted
process IMS T800 196, 197	IMS T222 431
process IMS T801 135, 136	IMS T414 376
process IMS T805 55, 56	IMS T425 308
Literal value 33 IMS T222 404	IMS T800 <b>238</b> IMS T801 <b>161, 167</b>
IMS 1222 404 IMS T414 338	IMS T805 81, 94
IMS T425 266	MemnotRfD1
IMS T800 194	IMS T414 358, <b>359</b> , 366, 373
IMS T801 133	IMS T425 289, <b>290</b> , 297, 305
IMS T805 53	IMS T800 220, <b>221</b> , 228, 235
Livermore loop 41	IMS T805 79, <b>80</b> , 91, 96
Load	MemnotWrD0
capacitive 9	IMS T414 358, <b>359</b> , 366, 368, 373
instruction IMS T222 404	IMS T425 289, <b>290</b> , 297, 299, 305 IMS T800 220, <b>221</b> , 228, 230, 235
instruction IMS T414 338 instruction IMS T425 266	IMS T800 220, <b>221</b> , 228, 230, 233
instruction IMS T800 194	Memory 31
instruction IMS T801 133	access IMS T222 403

access IMS 1414 337	Interface IMS 1414 337, 353, 357
access IMS T425 265	interface IMS T425 265, 284, 288
access IMS T800 193	interface IMS T800 193, 215, <b>219</b>
access IMS T801 132	interface IMS T801 132, 155, <b>159</b>
access IMS T805 52	interface IMS T805 52, 75, <b>79</b>
address IMS T414 359	internal IMS T222 420, 423
address IMS T425 290	internal IMS T414 355, 358
address IMS T800 221	internal IMS T425 286, 289
address IMS T805 80	internal IMS T800 217, 220
bandwidth 10, 43	internal IMS T801 157, 159
bandwidth IMS T222 405	internal IMS T805 77, 79
bandwidth IMS T414 339	local 10
bandwidth IMS T425 267	map 23, 25
bandwidth IMS T800 195	map IMS T222 <b>421</b>
bandwidth IMS T801 134	map IMS T414 356
bandwidth IMS T805 54	map IMS T425 287
channel 23	map IMS T800 218
configuration IMS T222 418	map IMS T801 158
configuration IMS T414 353, 357, 366	on-chip IMS T222 403
configuration IMS T425 284, 288, <b>297</b>	on-chip IMS T414 337
configuration IMS T800 215, 219, 228	on-chip IMS T425 265
configuration IMS T801 155	on-chip IMS T800 193
configuration IMS T805 75, 79, 81, 96	on-chip IMS T801 132
configuration, external IMS T414 366, 368	on-chip IMS T805 52
configuration, external IMS T425 297, 299	performance 118, 184, 254, 324, 391, 446
configuration, external IMS T800 228, <b>230</b>	read IMS T414 361
configuration, external IMS T805 98	read IMS T425 292
configuration, internal IMS T414 366 configuration, internal IMS T425 297	read IMS T800 223
	read IMS T805 86
configuration, internal IMS T800 228	refresh IMS T414 353, 358, 368, <b>373</b> , 374,
configuration, internal IMS T805 96	376
data IMS T414 359 data IMS T425 290	refresh IMS T425 284, 289, 299, <b>305</b> , 306,
data IMS T800 221	308 refresh IMS T800, 215, 220, 220, 225, 226
data IMS 7805 80	refresh IMS T800 215, 220, 230, <b>235</b> , 236, 238
direct access IMS T414 376	
direct access IMS T414 376	refresh IMS T801 155 refresh IMS T805 75, 79, <b>80</b> , 89, <b>91</b> , 94, 98
direct access IMS T800 238	strobe IMS T222 <b>422</b>
direct access IMS T805 94	strobe IMS T414 357, <b>359</b> , 362, 374, 376
dynamic IMS T414 357	strobe IMS T425 288, <b>290</b> , 293, 306, 308
dynamic IMS T425 288	strobe IMS T800 219, <b>221</b> , 224, 236, 238
dynamic IMS T800 219	strobe IMS T801 <b>162</b>
dynamic IMS T805 79	strobe IMS T805 <b>80</b> , 82, 89, 94
external IMS T222 420, 423	wait IMS T222 423, <b>429</b>
external IMS T414 355	wait IMS T414 357, 362, <b>374</b> , <b>375</b>
external IMS T425 286	wait IMS T425 288, 293, 306, 307
external IMS T800 217	wait IMS T800 219, 224, <b>236</b> , <b>237</b>
external IMS T801 157, 159	wait IMS T801 159, <b>165</b>
external IMS T805 77	wait IMS T805 79, <b>89</b> , <b>90</b>
global 10	MemReg
IMS M212 466, 471	IMS T222 431
IMS T222 420	IMS T414 <b>376</b>
IMS T414 355	IMS T425 308
IMS T425 286	IMS T800 238
IMS T800 217	IMS T801 161, 167
IMS T801 157	IMS T805 81, 94
IMS T805 77	MemStart 25
integrated 29	IMS T222 416, 418, <b>420</b>
interface 10, 24	IMS T414 351, 353, <b>355</b>
interface IMS T222 403, 418, <b>422</b>	IMS T425 282, 284, <b>286</b>

IMS T800 213, 215, <b>217</b>	NaN
IMS T801 153, 155, <b>157</b>	IMS T800 210
IMS T805 73, 75, <b>77</b>	IMS T801 150
MemWait	IMS T805 <b>70</b>
IMS T222 429	Network 7, 9, 12, 20, 22, 25
IMS T414 374	disk processor IMS M212 471
IMS T425 306	Node 8
IMS T800 236	NORMALISE 116, 182, 252
IMS T801 161, 165	NOT 18, 114, 180, 250, 321, 388, 443
IMS T805 <b>80</b>	notCS
Message 10, 12	IMS C011 514
IMS T222 407	IMS C012 538
IMS T414 341	notMemCE
IMS T425 269	IMS T222 423, 424, <b>426</b> , 429, 431
IMS T800 197	IMS T801 159, <b>160</b> , 164, 165, 167
IMS T801 136	notMemRd
IMS T805 56	IMS T414 359
pointer 36	IMS T425 <b>290</b>
transfer 38	IMS T800 221
Microcode 32	IMS T805 <b>80</b>
computing engine IMS T800 209	notMemRf
computing engine IMS T801 149	IMS T414 373
computing engine IMS T805 69	IMS T425 305
cycle IMS T222 <b>422</b>	IMS T800 235
cycle IMS T414 357	IMS T805 80
cycle IMS T425 288	notMemS0-4
cycle IMS T800 219	IMS T414 <b>359</b> , 368, 373, 374
cycle IMS T801 162	
	IMS T425 <b>290</b> , 299, 305, 306
cycle IMS T805 <b>82</b>	IMS T800 <b>221</b> , 230, 235, 236
scheduler 30, 34	IMS T805 <b>80</b> , 89, 91, 98
scheduler IMS T222 406	notMemWrB0-1
scheduler IMS T414 340	IMS T222 423, <b>424</b> , 426, 428, 431
scheduler IMS T425 268	notMemWrB0-3
scheduler IMS T800 196	IMS T414 <b>363</b> , 373
scheduler IMS T801 135	IMS T425 294, 305
scheduler IMS T805 55	IMS T800 <b>225</b> , 235
Microprocessor	IMS T801 159, 160, <b>161</b> , <b>164</b> , 167
bus IMS C011 504	IMS T805 <b>80</b> , 91
bus IMS C012 530	7 0 40 00 00
IMS C011 514	occam 7, 8, 12, 28, <b>29</b>
IMS C012 538	channel 10, 23
MINUS 18, 114, 180, 250, 321, 388, 443	communication 12
Mode 1	concurrency 12
IMS C011 504, <b>512</b>	model <b>12</b> , 30
IMS M212 467	process 7, 21, 22, 23
link IMS C011 507, <b>512</b>	program 7, 12, 20, 22
Mode 2	synchronism 22
IMS C011 504, <b>514</b>	Operand 32
	· · · · · · · · · · · · · · · · · · ·
IMS M212 467, 468	IMS T222 403, 404, <b>409</b>
link IMS C011 507, <b>514</b>	IMS T414 337, 338, <b>343</b>
Modulo operator 114, 180, 250, 321, 388, 443	IMS T425 265, 266, <b>272</b>
MOVE2D 43, 118, 184, 254, 324	IMS T800 193, 194, <b>200</b>
Multiple length arithmetic 116, 182, 252, 322,	IMS T801 132, 133, <b>139</b>
389, 444	IMS T805 52, 53, <b>59</b>
Multiple processor 22	register 33
Multiplication performance 115, 181, 251, 322,	types 18
389, 444	Operating conditions
<del></del> ,	IMS C004 494
Name 117, 183, 253	IMS C011 519
1441110 117, 100, 200	IMS C012 543
	11VIC 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

574\_\_\_\_\_\_B index

IMS T222 437	pins 24
IMS T414 382	port IMS C011 513
IMS T425 315	process 12, 13, 15
IMS T800 244	process IMS T222 408
IMS T801 174	process IMS T414 342
IMS T805 108	process IMS T425 270
Operation	process IMS T800 198
arithmetic IMS T222 403	process IMS T801 137
arithmetic IMS T414 337	process IMS T805 57
arithmetic IMS T425 265	register IMS C011 517
arithmetic IMS T800 193	register IMS C012 541
arithmetic IMS T801 132	Output Ready
arithmetic IMS T805 52	IMS C011 517, 518
code IMS T222 409	IMS C012 541, 542
code IMS T414 343	OutputInt
code IMS T425 272	IMS C011 508, 517, <b>518</b>
code IMS T800 200	IMS C012 534, 541, <b>542</b>
code IMS T801 139	Overflow
code IMS T805 59	stack IMS T222 403
logical IMS T222 403	stack IMS T414 337
logical IMS T414 337	stack IMS T425 265
logical IMS T425 265	stack IMS T800 193
logical IMS T800 193	stack IMS T801 132
logical IMS T801 132	stack IMS T805 52
logical IMS T805 52	
Operator 18	Packet
arithmetic 18, 114, 117, 180, 183, 250, 253,	link 39
321, 388, 443	PAR 14, 16, 29, 115, 119, 181, 184, 251, 255,
bit 18, 114, 180, 250, 321, 388, 443	322, 325, 389, 391, 444, 446
boolean 18, 114, 180, 250, 321, 388, 443	Parallel
comparison 114, 117, 180, 183, 250, 253,	communication IMS C011 504
321, 388, 443	communication IMS C012 530
modulo 18, 114, 180, 250, 321, 388, 443	construction 12, <b>14</b> , <b>20</b> , 29, 35
relational 18	construction IMS T222 407
shift 18	construction IMS T414 341
Optimisation	construction IMS T425 269
IMS T222 419	construction IMS T800 197
IMS T414 354	construction IMS T801 136
IMS T425 285	construction IMS T805 56
IMS T800 216	interface IMS C011 514
IMS T801 156	interface IMS C012 538
IMS T805 76	port IMS C011 512
program 21	process IMS T222 406
OR 18, 114, 180, 250, 321, 388, 443	process IMS T414 340
Ordering details 126, 188, 260, 332, 397, 452,	process IMS T425 268
461, 477, 502, 528, 550	process IMS T800 196
Output 12, 13, 19, 25, 29, 114, 180, 250, 321,	process IMS T801 135
388, 443	process IMS T805 55
buffer IMS C011 517	Part program 22
buffer IMS C012 541	Pascal 23
channel 29	Pattern recognition 34
link IMS C004 485	Peek
link IMS C011 509	IMS T222 418
link IMS C012 535	IMS T414 353
link IMS T222 418, <b>434</b>	IMS T425 284
link IMS T414 353, <b>379</b>	IMS T800 215
link IMS T425 284, <b>312</b>	IMS T801 155
link IMS T800 215, <b>241</b>	IMS T805 <b>75</b>
link IMS T801 155, <b>171</b>	
	renormance 12, 20, 34, 113, 179, 249 320
link IMS T805 75, <b>105</b>	Performance 12, 28, 34, 113, 179, 249, 320, 387, 442

	bit counting 118, 184, 254, 324	IMS T805 71
	bit reversal 118, 184, 254, 324	Pipelined vector processor 42
	block move 118, 184, 254, 324 construction 118, 184, 254, 325, 391, 446	PLACE 20 PLACED PAR 20
	CRC 118, 184, 254, 324	Placement 20, 23
	Cyclic Redundancy Checking 118, 184, 254, 324	PLLx 126, 188, 260, 332, 397, 452, 461, 477, 502, 528, 550
	estimation 113, 179, 249, 320, 387, 442	IMS T222 <b>422</b>
	external memory 118, 119, 184, 254, 255,	IMS T414 357
	324, 325, 391, 446	IMS T425 288
	external RAM 118, 184, 254, 324, 391, 446	IMS T800 219
	floating point 113, <b>117</b> , 179, <b>183</b> , 249, <b>253</b> ,	IMS T801 162
	323, 390, 445	IMS T805 82
	Floating point processor 41	PLUS 18, 114, 180, 250, 321, 388, 443
	IMS T222 405	Pointer
	IMS T414 339	IMS T222 403 IMS T414 337
	IMS T425 267 IMS T800 195	IMS T414 337
	IMS T801 134	IMS T800 193
	IMS T805 54	IMS T801 132
	integer 113, 179, 249, 320, 387, 442	IMS T805 52
	interrupt latency 119, 185, 255, 325, 392, 447	instruction 35
	link 391	instruction IMS T222 407
	link IMS C004 485	instruction IMS T414 341
	link IMS C011 509	instruction IMS T425 269
	link IMS C012 535 link IMS T222 434	instruction IMS T800 197 instruction IMS T801 136
	link IMS T414 379	instruction IMS T805 56
	link IMS T425 312	message 36
	link IMS T800 241	workspace 33
	link IMS T801 171	Poke
	link IMS T805 105	IMS T222 418
	measurement 22	IMS T414 353
	multiple length arithmetic 116, 182, 252, 322, 389, 444	IMS T425 <b>284</b> IMS T800 <b>215</b>
	multiplication 115, 181, 251, 322, 389, 444	IMS T800 213
	predefined maths 116, 182, 252, 322, 389,	IMS T805 <b>75</b>
	444	Port 19
	priority 119, 185, 255, 325, 392, 447	asynchronism 19
	product 115, 181, 251, 322, 389, 444	disk IMS M212 465
	special purpose functions 118, 184, 254, 324	input IMS C011 512
	square root 117, 119, 183, 184, 253, 255, 325, 391, 446	output IMS C011 513 parallel IMS C011 512
	TIMES 115, 181, 251, 322, 389, 444	synchronism 19
	wait states 118, 184, 254, 324, 391, 446	PORT 25
P	Peripheral 25	Power 24
	access 19	IMS C004 482
	control transputer 25	IMS C011 506
	device 29 interface IMS M212 465	IMS C012 532 IMS T222 415
	memory mapping 25	IMS T414 349
P	Phase lock loop	IMS T425 280
	IMS C004 482	IMS T800 211
	IMS C011 506	IMS T801 151
	IMS C012 532	IMS T805 71
	IMS T222 415	rating IMS C004 497
	IMS T414 349	rating IMS C011 523
	IMS T425 <b>280</b> IMS T800 <b>211</b>	rating IMS C012 547 rating IMS T222 441
	IMS T801 151	rating IMS T414 386
		<u> </u>

roting IMC TAGE 240	communication IMC TOOL 407
rating IMS T425 319 rating IMS T800 248	communication IMS T222 407 communication IMS T414 341
rating IMS T800 240	communication IMS T414 341
rating IMS T805 112	communication IMS T800 197
Prefetch 34	communication IMS T801 136
Prefix function 33, 34	communication IMS T805 56
IMS T222 404	concurrent 14, 22, 29
IMS T414 338	deschedule 38
IMS T425 <b>266</b>	execution 35, 36, 37
IMS T800 194	hardware 9
IMS T801 133	high priority IMS T222 419
IMS T805 <b>53</b>	high priority IMS T414 354
PRI PAR 20	high priority IMS T425 285
Primitive 114, 180, 250, 321, 388, 443	high priority IMS T800 216
Primitive type 17	high priority IMS T801 156
Priority 20	high priority IMS T805 76
bootstrap IMS T222 418	IMS T222 406, 419
bootstrap IMS T414 351 bootstrap IMS T425 284	IMS T414 <b>340</b> , 354 IMS T425 <b>268</b> , 285
bootstrap IMS T800 215	IMS T800 <b>196</b> , 216
bootstrap IMS T801 155	IMS T800 136, 216
bootstrap IMS T805 75	IMS T805 <b>55</b> , 76
floating point IMS T800 209	input 12, <b>13</b> , 15
floating point IMS T801 149	input IMS T222 408
floating point IMS T805 69	input IMS T414 342
IMS T222 406, 407, 418	input IMS T425 270
IMS T414 <b>340</b> , <b>341</b> , 353	input IMS T800 198
IMS T425 <b>268</b> , <b>269</b> , 284	input IMS T801 137
IMS T800 196, 197, 215	input IMS T805 57
IMS T801 135, 136, 155	latency IMS T222 407
IMS T805 <b>55</b> , <b>56</b> , 75	latency IMS T414 341
level 30, 31	latency IMS T425 269
performance 119, 185, 255, 325, 392, 447	latency IMS T800 197
timer IMS T222 408	latency IMS T801 136
timer IMS T414 342	latency IMS T805 56
timer IMS T425 270	list 35
timer IMS T800 198	list IMS T222 <b>406</b> , 407
timer IMS T801 137 timer IMS T805 57	list IMS T414 <b>340</b> , 341
PROC 18, 22, 23, 114, 117, 180, 183, 250, 253,	list IMS T425 <b>268</b> , 269 list IMS T800 <b>196</b> , 197
321, 388, 443	list IMS T800 136, 137
ProcClockOut	list IMS T805 <b>55</b> , 56
IMS T222 422	low priority IMS T222 419
IMS T414 357	low priority IMS T414 354
IMS T425 288	low priority IMS T425 285
IMS T800 219	low priority IMS T800 216
IMS T801 162	low priority IMS T801 156
IMS T805 82	low priority IMS T805 76
Procedure 115, 181, 251, 322, 389, 444	mapping 12
Procedures 18	monitor_22
Process 8, 9, 12, 13, 14, 18	new 35
active/inactive IMS T333 405 407	occam 22, 23
active/inactive IMS T222 406, 407 active/inactive IMS T414 340, 341	output 12, <b>13</b> , 15
active/inactive IMS T414 340, 341 active/inactive IMS T425 268, 269	output IMS T222 408 output IMS T414 342
active/inactive IMS T800 196, 197	output IMS T414 342
active/inactive IMS T801 135, 136	output IMS T425 276
active/inactive IMS T805 55, 56	output IMS T801 137
assignment 12, 13	output IMS T805 57
channel 29	parallel IMS T222 406

parallel IMS T414 340	speed select IMS T800 212
parallel IMS T425 268	speed select IMS T801 152
parallel IMS T800 196	speed select IMS T805 72
parallel IMS T801 135	ProcSpeedSelect0-2
parallel IMS T805 55	IMS T425 281
primitive 29	IMS T800 212
queue IMS T222 406	IMS T801 152
queue IMS T414 340	IMS T805 72
queue IMS T425 <b>268</b>	Product performance 115, 181, 251, 322, 389
queue IMS T800 196	444 Program
queue IMS T801 135 queue IMS T805 55	Program bootstrap IMS T222 432
reschedule 38	bootstrap IMS T801 167
seguential IMS T222 403	configuration 20
sequential IMS T414 337	development 22
sequential IMS T425 265	occam 22
sequential IMS T800 193	optimisation 21
sequential IMS T801 132	part 22
sequential IMS T805 52	Programmable
simulation 22	components 9
software 8	device 29
switch time 35	i/o IMS C011 504
switch time IMS T222 407	Programming 8
switch time IMS T414 341	model 12
switch time IMS T425 269	structure 33
switch time IMS T800 197	protocol
switch time IMS T801 136	CHAN OF 114, 180, 250, 321, 388, 443
switch time IMS T805 56	Protocol
timing IMS T222 408	link 39
timing IMS T414 342	
timing IMS T425 270	Q0-7
timing IMS T800 198	IMS C011 508, 513
timing IMS T801 137	QAck
timing IMS T805 57	IMS C011 513
Processor 24, 31	Queue 36
clock IMS T222 408	priority IMS T222 406
clock IMS T414 342	priority IMS T414 340
clock IMS T425 270	priority IMS T425 268
clock IMS T800 198	priority IMS T800 196
clock IMS T801 137	priority IMS T801 135
clock IMS T805 57	priority IMS T805 55
flags IMS T222 418	process IMS T222 407
flags IMS T414 353	process IMS T414 341
flags IMS T425 284	process IMS T425 269
flags IMS T800 215	process IMS T800 197
flags IMS T801 155	process IMS T801 136
flags IMS T805 75	process IMS T805 56
IMS M212 465, <b>467</b>	timer IMS T222 408
IMS T222 403	timer IMS T414 342
IMS T414 337	timer IMS T425 <b>270</b>
IMS T425 <b>265</b>	timer IMS T800 198
IMS T800 193	timer IMS T801 138
IMS T801 132	timer IMS T805 58
IMS T805 <b>52</b>	QValid
multiple 22	IMS C011 508, 513
speed IMS T425 262 speed IMS T800 190	DAM 110 104 254 224 201 446
speed IMS T800 190 speed IMS T801 128	RAM 118, 184, 254, 324, 391, 446
speed IMS T801 128 speed IMS T805 48	IMS T222 420 IMS T414 351, 355
speed into 1805 48	IMS 1414 351, 355 IMS T425 282 286
AUGGU AGIGU IIVIO 1470 ZOT	IIVI-3 147:3 707 70D

IMS T800 213, 217	data input IMS C012 538
IMS T801 157	FA IMS T800 209
IMS T805 73, 77	FA IMS T801 149
Read	FA IMS T805 69
cycle IMS T222 423	FB IMS T800 209
cycle IMS T414 359, 360	FB IMS T801 149
cycle IMS T425 290, 291	FB IMS T805 69
cycle IMS T800 221, 222	FC IMS T800 209
cycle IMS T801 163	FC IMS T801 149
cycle IMS T805 83, 84	FC IMS T805 69
data IMS C011 518	Floating point 31
data IMS C012 542	I IMS T222 419
dynamic memory cycle IMS T414 361	I IMS T414 354
dynamic memory cycle IMS T425 292	I IMS T425 285
dynamic memory cycle IMS T800 223	I IMS T800 216
dynamic memory cycle IMS T805 86	I IMS T801 156
external cycle IMS T414 360, 361	I IMS T805 76
external cycle IMS T425 291, 292	IMS C011 514
external cycle IMS T800 222, 223	IMS C012 538
external cycle IMS T805 84, 86	IMS M212 465
<b>REAL</b> 117, 183, 253	IMS T222 403, 406, 418
Real time 22	IMS T414 <b>337</b> , 340, 353
<b>REAL32</b> 17, 117, 183, 253, 323, 390, 445	IMS T425 <b>265</b> , 268, 284
conversion 117, 183, 253	IMS T800 <b>193</b> , 196, 215
<b>REAL 64</b> 17, 117, 183, 253, 323, 390, 445	IMS T801 <b>132</b> , 135, 155
conversion 117, 183, 253	IMS T805 <b>52</b> , 55, 75
Refresh	input data IMS C011 514, 518
memory IMS T414 353, 358, 368, <b>373</b> , 374,	input data IMS C012 538, 542
376	input status IMS C011 514, <b>517</b> , 518
memory IMS T425 284, 289, 299, <b>305</b> , 306,	input status IMS C012 538, <b>541</b> , 542
308	link interface 38
memory IMS T800 215, 220, 230, <b>235</b> , 236,	operand 33
238	operand IMS T222 404
memory IMS T801 155	operand IMS T414 338
memory IMS T805 75, 79, <b>80</b> , 89, <b>91</b> , 92, 94,	operand IMS T425 266
98 Define h Bending	operand IMS T800 194
RefreshPending	operand IMS T801 133
IMS T425 304	operand IMS T805 53
IMS T805 <b>80</b> , 91	output data IMS C011 514, 517, 518
Register 404 410	output data IMS C012 538, <b>541</b> , 542 output status IMS C011 514, <b>517</b> , 518
A IMS T222 403, 404, 419	
A IMS T414 337, 338, 354	output status IMS C012 538, <b>541</b> , 542
<i>A IMS T425</i> <b>265</b> , 266, 285 <i>A IMS T800</i> <b>193</b> , 194, 216	process list 35
A IMS 7800 193, 194, 216 A IMS 7801 132, 133, 156	timer IMS T222 408 timer IMS T414 342
A IMS 7807 132, 133, 136 A IMS 7805 52, 53, 76	timer IMS T414 342
B IMS T222 403	timer IMS T800 198
B IMS T414 337	timer IMS T801 137, 138
B IMS T425 265	timer IMS T805 <b>57</b> , 58
B IMS T800 193	W IMS T222 416
B IMS T801 132	W IMS T414 351
B IMS T805 52	W IMS T425 282
C IMS 7222 403	W IMS 1423 202 W IMS T800 213
C IMS T414 337	W IMS 7800 213 W IMS 7801 153
C IMS T474 337 C IMS T425 265	W IMS 7805 73
C IMS T800 193	workspace IMS T222 404
C IMS T800 193 C IMS T801 132	workspace IMS T414 338
C IMS 7807 132 C IMS 7805 52	workspace IMS T425 266
CPU 31, 32	workspace IMS T800 194
data input IMS C011 514	workspace IMS T801 133
data input into Out i 314	workspace livio 1001 100

workspace IMS T805 53 REM 18, 114, 117, 180, 183, 250, 253, 321,	floating point IMS T805 59, 69 SeparatelQ
388, 443	IMS C011 507
Repetition construction 16 Replication construction 16	<b>SEQ 14</b> , 15, 16, 29, 115, 119, 181, 184, 251, 255, 322, 325, 389, 391, 444, 446
Replication performance 115, 119, 181, 184,	Sequential
251, 255, 322, 325, 389, 391, 444, 446	construction 12, 14, 15, 29
Reschedule <b>35</b> , 38, 40	process IMS T222 403
Reset 24	process IMS T414 337
IMS C004 484	process IMS T425 265
IMS C011 508	process IMS T800 193
IMS C012 534	process IMS T801 132
IMS T222 <b>416</b> , 418, 419 IMS T414 <b>351</b> , 353, 354	process IMS T805 52
IMS T414 331, 333, 334 IMS T425 282, 284, 285	processing 32 Serial
IMS T800 <b>213</b> , 215, 216	data <b>39</b>
IMS T801 <b>153</b> , 155, 156	protocol 39
IMS T805 <b>73</b> , 75, 76	SHIFTLEFT 116, 182, 252
RnotW	SHIFTRIGHT 116, 182, 252
IMS C011 514	Sieve of Erastosthenes 119, 184, 255, 325,
IMS C012 538	391, 446
ROM 25	Silicon 31, 45
bootstrap code IMS T222 420	Single byte instruction
bootstrap code IMS T414 355	IMS T222 404
bootstrap code IMS T425 286	IMS T414 338
bootstrap code IMS T800 217	IMS T425 <b>266</b>
bootstrap code IMS T801 157	IMS T800 194
bootstrap code IMS T805 77	IMS T801 133
IMS M212 466	IMS T805 <b>53</b>
IMS T222 416 IMS T414 351	Skew strobe IMS T414 357, <b>363</b>
IMS T414 331 IMS T425 282	strobe IMS T425 288, <b>294</b>
IMS T800 213	strobe IMS T800 219, <b>225</b>
IMS T801 153	strobe IMS T805 82, <b>87</b>
IMS T805 73	SKIP 114, 180, 250, 321, 388, 443
ROTATELEFT 116, 182, 252	Software
ROTATERIGHT 116, 182, 252	IMS M212 468
RS0-1	kernel 34
IMS C011 514	kernel IMS T222 406
IMS C012 538	kernel IMS T414 340
Run time 23	kernel IMS T425 268
04400450	kernel IMS T800 196
SA400/450	kernel IMS T801 135
IMS M212 465, <b>467</b> Schodular 30 34 <b>35</b>	kernel IMS T805 55 Special purpose functions 118 184 254 324
Scheduler 30, 34, <b>35</b> IMS T222 <b>406</b> , 407, 408	Special purpose functions 118, 184, 254, 324 Speed
IMS T414 <b>340</b> , 341, 342	benchmark 41
IMS T425 <b>268</b> , 269, 270	communication 30
IMS T800 <b>196</b> , 197, 198	link 39
IMS T801 135, 136, 137	processor 41
IMS T805 <b>55</b> , 56, 57	processor IMS T425 262
list 36	processor IMS T800 190
operation 35	processor IMS T801 128
SCSI	processor IMS T805 48
bus IMS M212 471	select IMS T425 281
interface IMS M212 472	select IMS T800 212
Selection construction 16	select IMS T801 152
Selector sequence floating point IMS T800 200, 209	select IMS T805 <b>72</b> SQRT 117, 183, 253
floating point IMS T800 200, 209	Square 18
HOUSING POINT HITO TOO! 100, 170	~~~~~ ·~

Square root 117, 183, 253 performance 119, 184, 255, 325, 391, 446	memory IMS T801 <b>162</b> memory IMS T805 <b>80</b> , 82, 89, 94
ST506/412	skew IMS T414 357, 363
IMS M212 465, <b>467</b>	skew IMS T425 288, <b>294</b>
Stability 1990 1990 1990	skew IMS T800 219, <b>225</b>
clock IMS C004 482	skew IMS T805 82, 87
clock IMS C011 506	timing IMS T414 362 timing IMS T425 293
clock IMS C012 532 clock IMS T222 415	timing IMS T800 224
clock IMS T414 349	timing IMS 7805 224
clock IMS T425 280	write IMS T414 363
clock IMS T800 211	write IMS T425 294
clock IMS T801 151	write IMS T800 225
clock IMS T805 71	write IMS T805 80
Stack	Structure
evaluation 31, <b>32</b> , 36	data IMS T222 404
evaluation IMS T222 403, 407, 408	data IMS T414 338
evaluation IMS T414 <b>337</b> , 341, 342	data IMS T425 266
evaluation IMS T425 <b>265</b> , 269, 270	data IMS T800 194
evaluation IMS T800 193, 197, 198	data IMS T801 133
evaluation IMS T801 132, 136, 137	data IMS T805 53
evaluation IMS T805 <b>52</b> , 56, 57	Subscript 17
floating point IMS T800 209	constant 114, 180, 250, 321, 388, 443
floating point IMS T801 149	expression 114, 180, 250, 321, 388, 443
floating point IMS T805 69	variable 114, 180, 250, 321, 388, 443
optimise 41 overflow 32, 41	Synchronisation link IMS C004 485
overflow IMS T222 403	link IMS C004 483
overflow IMS T414 337	link IMS C011 509
overflow IMS T425 265	link IMS T222 434
overflow IMS T800 193	link IMS T414 379
overflow IMS T801 132	link IMS T425 312
overflow IMS T805 52	link IMS T800 241
Start	link IMS T801 171
bit <b>39</b>	link IMS T805 105
Status	point IMS T800 209
IMS T222 418	point IMS T801 149
IMS T414 353	point IMS T805 69
IMS T425 284	System services 24
IMS T800 215	IMS C004 482
IMS T801 155	IMS C011 506
IMS T805 75	IMS C012 532
register IMS C011 514	IMS T222 415
register IMS C012 538	IMS T414 <b>349</b> IMS T425 <b>280</b>
Stop bit 39	IMS T800 211
STOP 21, 114, 180, 250, 321, 388, 443	IMS T800 211 IMS T801 151
Store 21, 114, 100, 230, 321, 330, 443	IMS T805 71
instruction IMS T222 404	IMO 1000 71
instruction IMS T414 338	testerr
instruction IMS T425 266	IMS T222 419
instruction IMS T800 194	IMS T414 354
instruction IMS T801 133	IMS T425 285
instruction IMS T805 53	IMS T800 216
String 19	IMS T801 156
Strobe	IMS T805 76
memory IMS T222 422	tim 19
memory IMS T414 357, <b>359</b> , 362, 374, 376	Time 19
memory IMS T425 288, <b>290</b> , 293, 306, 308	delay IMS T222 406
memory IMS TROO 210 221 224 236 238	delay IMS T414 340

delay IMS T425 268	strobe IMS T425 293
delay IMS T800 196	strobe IMS T800 224
delay IMS T801 135	strobe IMS T805 85
delay IMS T805 55	Tm
process switch IMS T222 407	IMS T414 357
process switch IMS T414 341	IMS T425 288
process switch IMS T425 269	IMS T800 219
process switch IMS T800 197	IMS T805 79, <b>82</b>
process switch IMS T801 136	TPtrLoc1, TPtrLoc2
process switch IMS T805 56	IMS T222 420
real 22	IMS T414 355
slice IMS T222 406, 407	IMS T425 <b>286</b>
slice IMS T414 <b>340</b> , 341	IMS T800 217
slice IMS T425 <b>268</b> , 269	IMS T801 <b>157</b>
slice IMS T800 196, 197	IMS T805 77
slice IMS T801 135, 136	Transfer message 38
slice IMS T805 55, 56	Transmission link 39
slice period IMS T222 407	Transputer
slice period IMS T414 341	array IMS M212 471
slice period IMS T425 269	clock 11
slice period IMS T800 197	development system 12, 20, 21, 25
slice period IMS T801 136	development system IMS T414 357
slice period IMS T805 56	development system IMS T425 288
Timeout 21	development system IMS T800 219
Timer 17, 19, 22, 40, 115, 181, 251, 322, 389,	development system IMS T805 79
444	interconnection 31
AFTER 115, 181, 251, 322, 389, 444	TRUE 18
clock 19	Tstate
clock IMS T222 408	IMS T222 423
clock IMS T414 342	IMS T414 <b>357</b> , 359, 368
clock IMS T425 270	IMS T425 288, 290, 299
clock IMS T800 198	IMS T800 <b>219</b> , 221, 230
clock IMS T801 137	IMS T801 159
clock IMS T805 57	IMS T805 <b>79</b> , 82, 98
delay 19	TTL compatibility
IMS T222 407	link IMS C004 485
IMS T414 341	link IMS C011 509
IMS T425 269	link IMS C012 535
IMS T800 197	link IMS T222 434
IMS T801 136	link IMS T414 379
IMS T805 56	link IMS T425 312
input 115, 181, 251, 322, 389, 444	link IMS T800 241
processor 40	link IMS T801 171
queue IMS T222 408	link IMS T805 105
queue IMS T414 342	Type <b>17</b>
queue IMS T425 270	array 17
queue IMS T800 198	BOOL 17
queue IMS T801 138	BYTE 17
queue IMS T805 <b>58</b>	CHAN OF 17
register IMS T222 408	floating point 17
register IMS T414 342	INT 17
register IMS T425 270	INT16 17
register IMS T800 198	INT32 17
register IMS T801 137, 138	INT64 17
register IMS T805 <b>57</b> , 58	primitive 17
TIMER 17	REAL32 17 REAL64 17
TIMES 18, 114, 180, 250, 321, 388, 443	record 17
performance 115, 181, 251, 322, 389, 444 Timing 12	TIMER 17
strobe IMS T414 <b>362</b>	variant 17
30000 INO 1414 302	valialit 11

	instruction IMS T800 197
Value constant 33 data 32	instruction IMS T801 136 instruction IMS T805 56 pointer <b>33</b>
literal 33	register IMS T222 404
Variable 12, <b>13</b> , 17, 18, 29, 114, 117, 180, 183, 250, 253, 321, 388, 443	register IMS T414 338 register IMS T425 266
array 114, 180, 250, 321, 388, 443 IMS T222 404	register IMS T800 194 register IMS T801 133
IMS T414 338 IMS T425 266	register IMS T805 53 WPtr
IMS T800 194 IMS T801 133	IMS T425 <b>286</b> IMS T805 <b>77</b>
IMS T805 53 subscript 114, 180, 250, 321, 388, 443	Write cycle IMS T222 423, 424
temporary IMS T222 405 temporary IMS T414 339	cycle IMS T414 359, <b>363</b> cycle IMS T425 290, <b>294</b>
temporary IMS T425 267 temporary IMS T800 195	cycle IMS T800 221, <b>225</b> cycle IMS T801 <b>164</b>
temporary IMS T801 134 temporary IMS T805 54 VCC 24	cycle IMS T805 88 data IMS C011 518
IMS C004 482 IMS C011 506	data IMS C012 542 early IMS T414 368 early IMS T425 299
IMS C012 532 IMS T222 415	early IMS T800 230 early IMS T805 98
IMS T414 349 IMS T425 280	late IMS T414 368 late IMS T425 299
IMS T800 211 IMS T801 151	late IMS T800 230 late IMS T805 98
IMS T805 <b>71</b> VLSI 29, 45	strobe IMS T222 <b>424</b> strobe IMS T414 <b>363</b>
Wait	strobe IMS T425 <b>294</b> strobe IMS T800 <b>225</b>
IMS T222 <b>429</b> IMS T414 357, 362, 369, <b>374, 375</b>	strobe IMS T805 80
IMS T425 288, 293, 300, <b>306, 307</b> IMS T800 219, 224, 231, <b>236, 237</b>	
IMS T801 <b>165</b> IMS T805 79, <b>89</b> , <b>90</b> , 99 state generator IMS T222 <b>429</b>	
state generator IMS T801 165 state IMS T222 423	
state IMS T801 159 Whetstone benchmark 41	
<b>WHILE 16</b> , 29, 115, 181, 251, 322, 389, 444	
Word access IMS T222 428	
length 31, 32, 34 Workspace 35, 37	
disk IMS M212 467	
IMS T222 403, 407, 418	
IMS T414 337, 341, 351	
IMS T425 265, 269, 284	
IMS T800 193, 197, 215 IMS T801 132, 136, 155	
IMS T801 132, 136, 155 IMS T805 52, 56, 75	
instruction IMS T222 407	
instruction IMS T414 341	
instruction IMS T425 269	



The Transputer Databook contains a description of the transputer architecture and engineering data for the following members of the transputer family:

IMS T805 32 bit transputer with on-chip floating point unit
IMS T801 32 bit transputer with on-chip floating point unit
IMS T800 32 bit transputer with on-chip floating point unit
IMS T425 32 bit transputer
IMS T414 32 bit transputer
IMS T222 16 bit transputer
IMS T225 16 bit transputer preview
IMS M212 intelligent disc drive controller preview

and communication devices

IMS C011 communications link adaptor IMS C012 communications link adaptor IMS C004 communications link switch

