IMS T801 transputer

inmos®

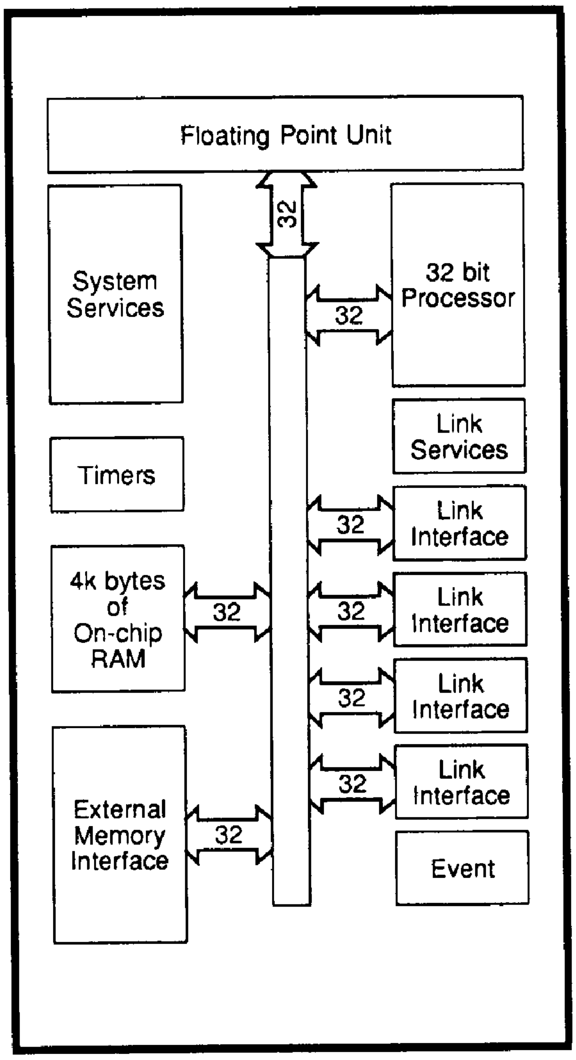
Preliminary Data

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



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.

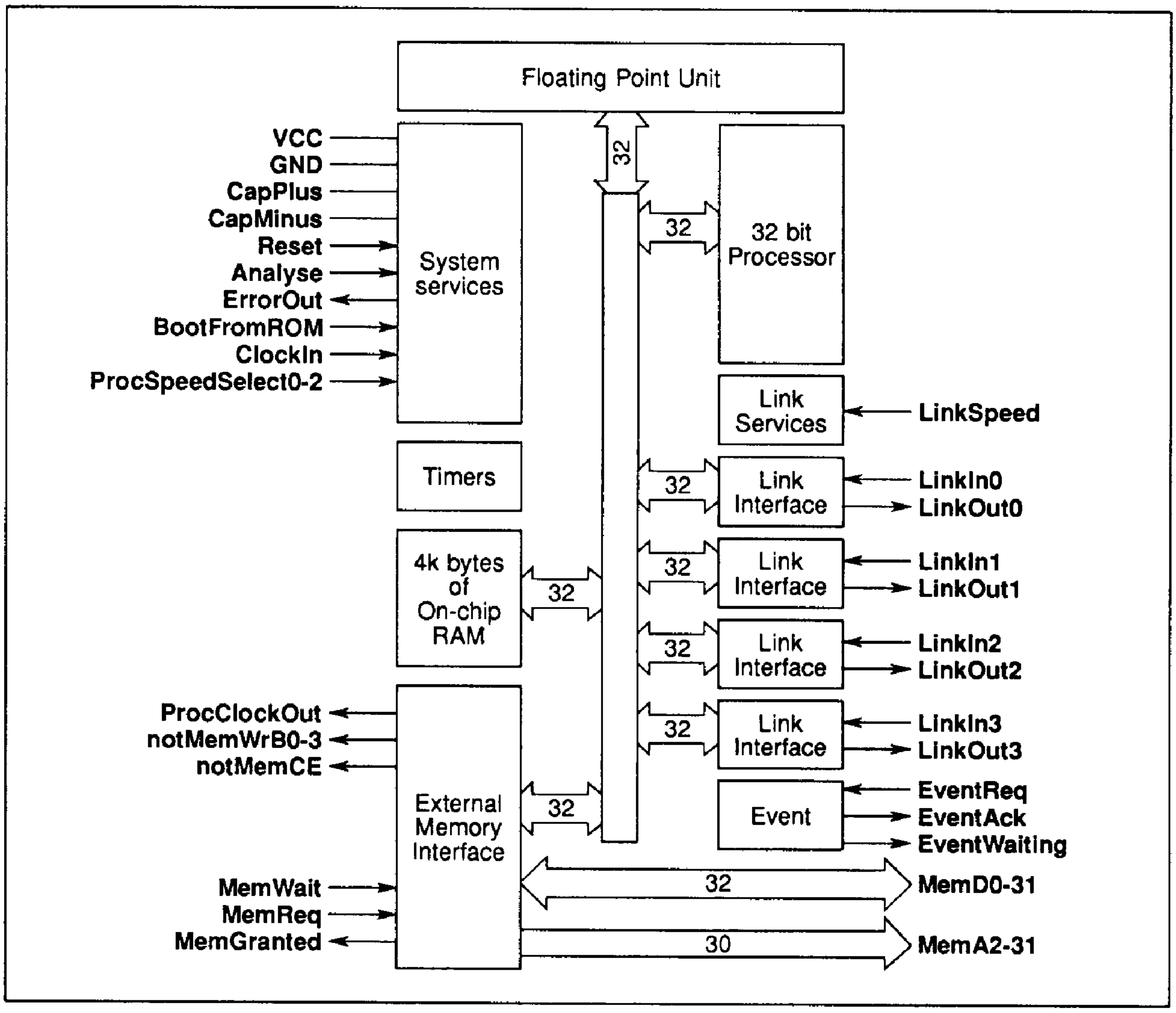


Figure 1.1 IMS T801 block diagram

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.

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.

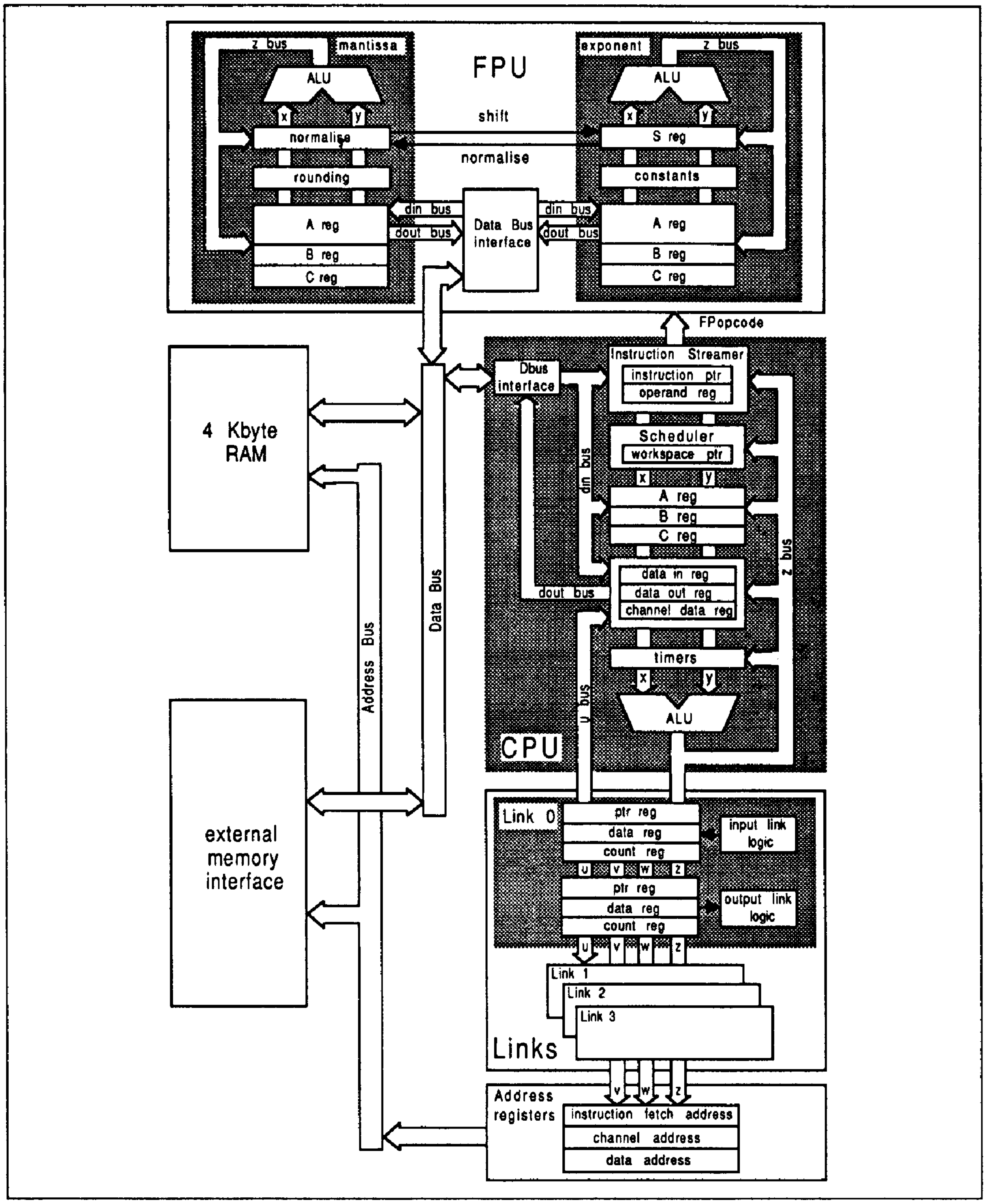


Figure 1.2 IMS T801 internal datapaths

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.*

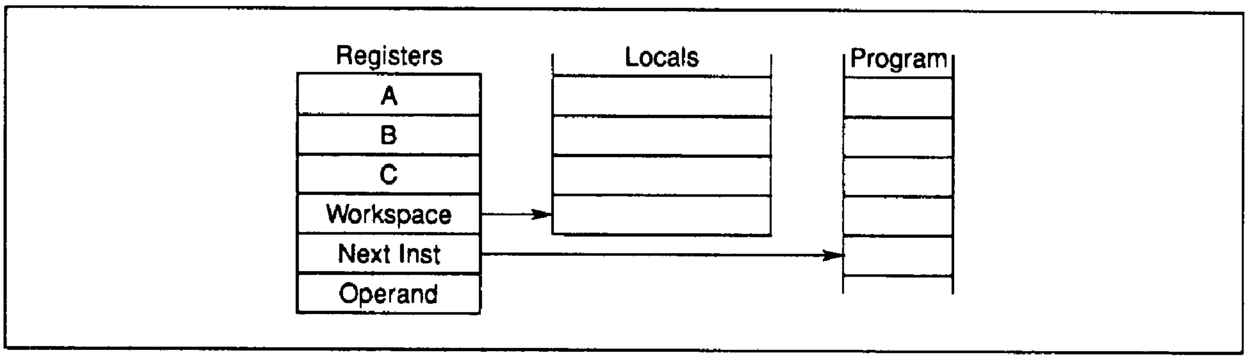


Figure 3.1 Registers

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.

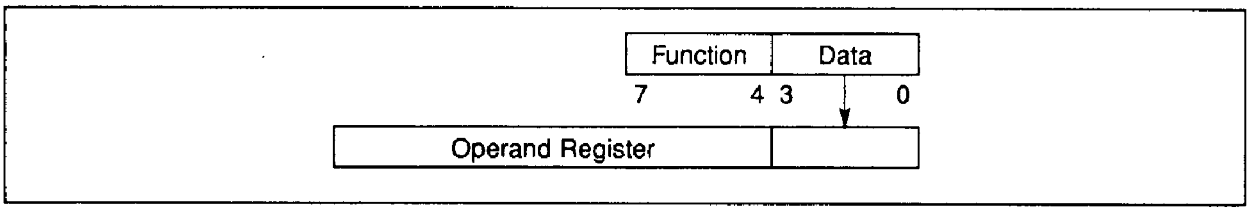


Figure 3.2 Instruction format

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.

Table 3.1 Direct functions

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 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.

Table 3.2 Expression evaluation

Program	Mnemonic	
x := 0	*ldc*	*0*
	stl	*x*
x := #24	*pfix*	*2*
	ldc	*4*
	stl	*x*
x := y + z	*ldl*	*y*
	ldl	*z*
	add	
	stl	*x*

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 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.

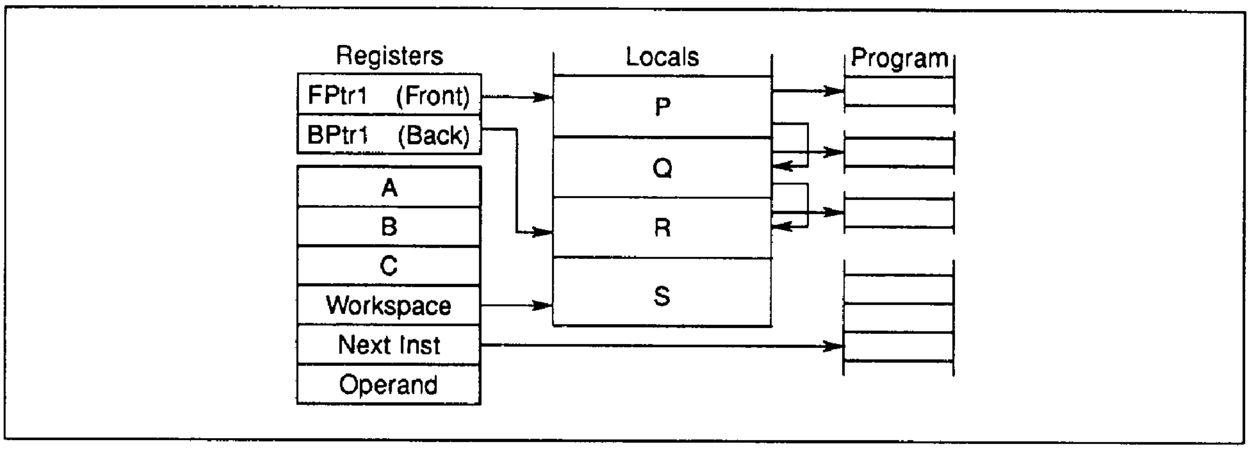


Figure 3.3 Linked process list

Table 3.3 Priority queue control registers

Function	High Priority	Low Priority
Pointer to front of active process list	*Fptr0*	*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 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 μs, 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.

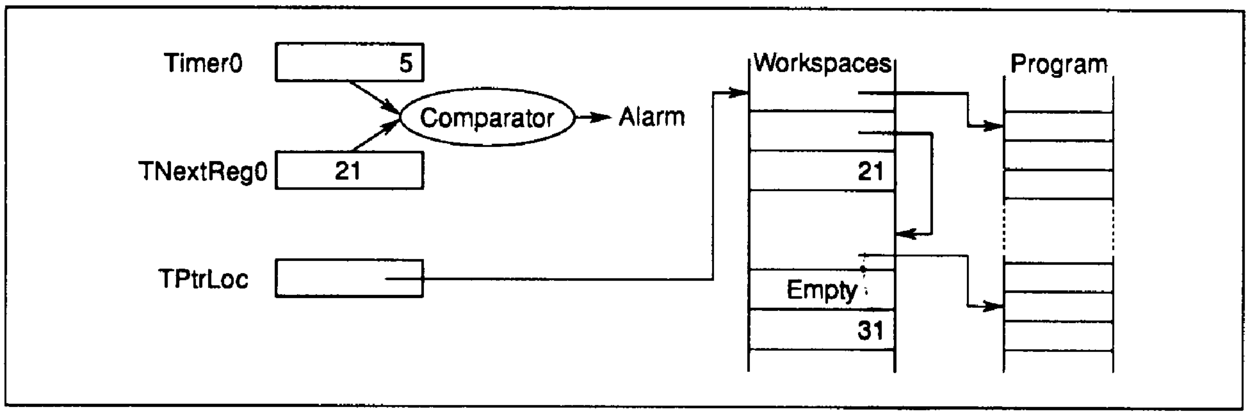


Figure 3.4 Timer registers

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*.

Table 4.1 *prefix* coding

Mnemonic		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
ldc	#7	#4	#47
ldc	-31 (*ldc* #FFFFFFE1)		
is coded as			
nfix	#1	#6	#61
ldc	#1	#4	#41

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.

Table 4.2 *operate* coding

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

The load device identity (*lddevid*) 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 *lddevid* 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 *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.
p	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

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 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*	
multiply	*fractional multiply*	*divide*	*remainder*
long add	*long subtract*	*long divide*	
set error	*testerr*	*fpcheckerror*	
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 **Iptr** and **Wptr** with an **Iptr** 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	*fpsub*	*fpmul*	*fpdiv*
fpldnladdsn	*fpldnladddb*	*fpldnlmulsn*	*fpldnlmuldb*
fpremfirst	*fpusqrtfirst*	*fpgt*	*fpeq*
fpuseterror	*fpuclearerror*	*fptesterror*	
fpuexpincby32	*fpuexpdecby32*	*fpumulby2*	*fpudivby2*
fpur32tor64	*fpur64tor32*	*fpucki32*	*fpucki64*
fprtoi32	*fpuabs*	*fpint*	

Table 4.8 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	
A	AX	cj	2	conditional jump (not taken)	
			4	conditional jump (taken)	
B	BX	ajw	1	adjust workspace	
C	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*	

Table 4.10 IMS T801 long arithmetic operation codes

Operation Code	Memory Code	Mnemonic	Processor Cycles	Name	D E
16	21F6	ladd	2	long add	E
38	23F8	lsub	2	long subtract	E
37	23F7	lsum	3	long sum	
4F	24FF	ldiff	3	long diff	
31	23F1	lmul	33	long multiply	
1A	21FA	ldiv	35	long divide	E
36	23F6	lshl	n+3	long shift left (n<32)	
			n-28	long shift left(n≥32)	
35	23F5	lshr	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 4.11 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	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	(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+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 T801 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	2w+8	move message	

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 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	

Table 4.17 IMS T801 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

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	ldpri	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	clrhalterr	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	lddevid	1	load device identity	
7E	27FE	ldmemstartval	1	load value of memstart address	

Table 4.21 IMS T801 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	

Table 4.22 IMS T801 floating point load/store operation codes

Operation Code	Memory Code	Mnemonic	Processor Cycles	Name	D E
8E	28FE	fpldnlsn	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	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	

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	

Table 4.24 IMS T801 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 4.25 IMS T801 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 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	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	
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.

Table 4.28 IMS T801 floating point arithmetic operation codes

Operation Code	Memory Code	Mnemonic	Processor cycles		Name	D E
			Single	Double		
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^{32}	F
09	s	fpuexpdec32	6/9	6/9	divide by 2^{32}	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 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

ldc 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	(op. code #AB)		#2AFB
is coded as			
pfix	#A	#2	#2A
opr	#B	#F	#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.*

Table 5.2 Typical floating point operation times for IMS T801

	T801-20		T801-30	
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

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 1μF 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.

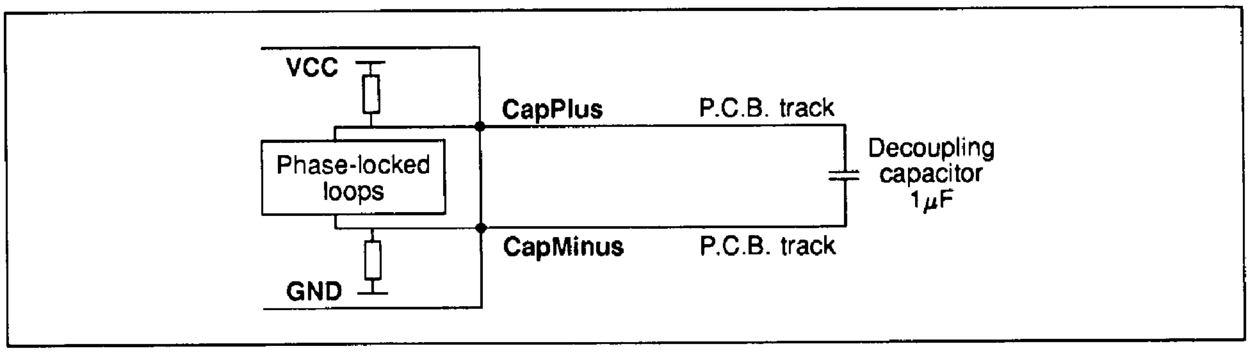


Figure 6.1 Recommended PLL decoupling

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.

Table 6.1 Input clock

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

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).

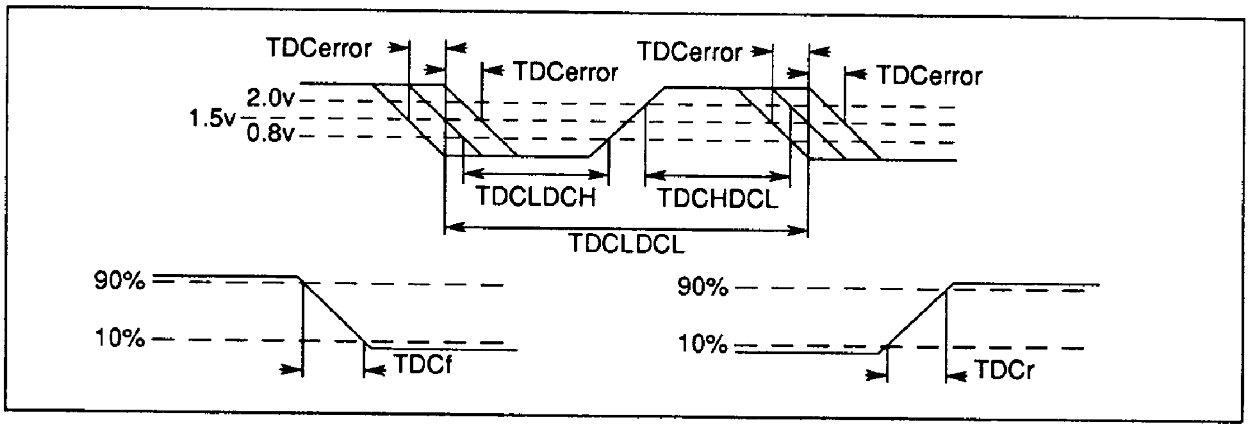


Figure 6.2 ClockIn timing

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.

Table 6.2 Processor speed selection

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

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, **BootFromRom** 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).

Table 6.3 Reset and Analyse

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

Notes

1 Full periods of **Clockln TDCLDCL** required.

2 At power-on reset.

3 Must be stable until after end of bootstrap period. See Bootstrap section.

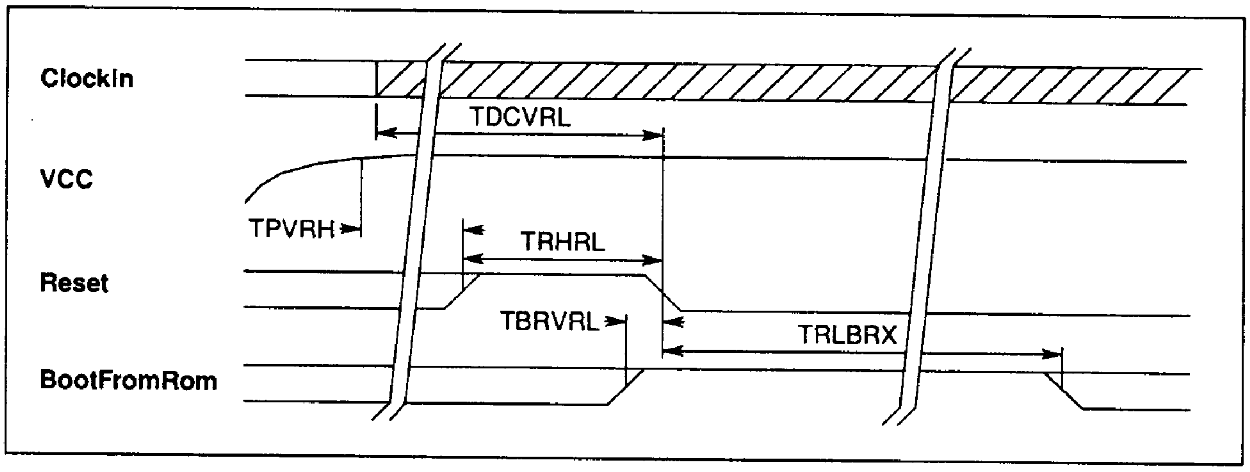


Figure 6.3 Transputer reset timing with Analyse low

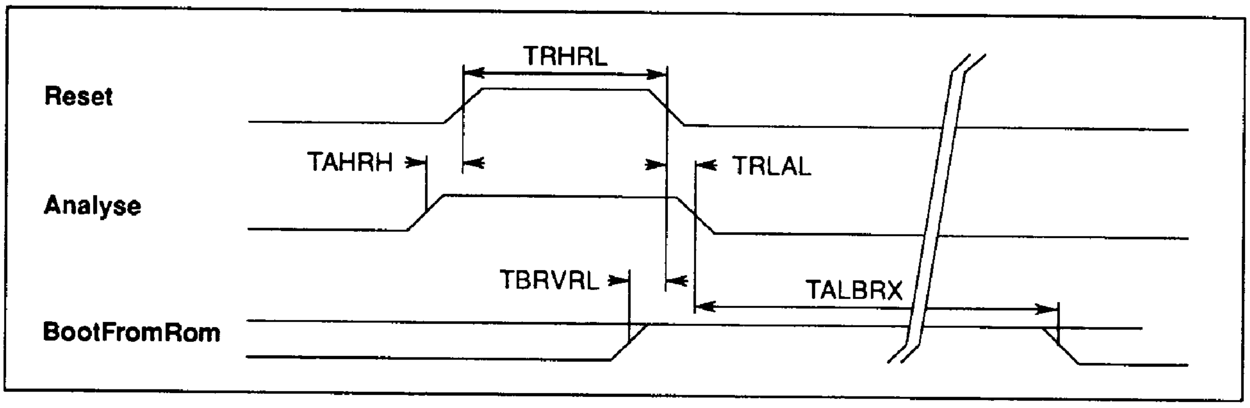


Figure 6.4 Transputer reset and analyse timing

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

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 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*.

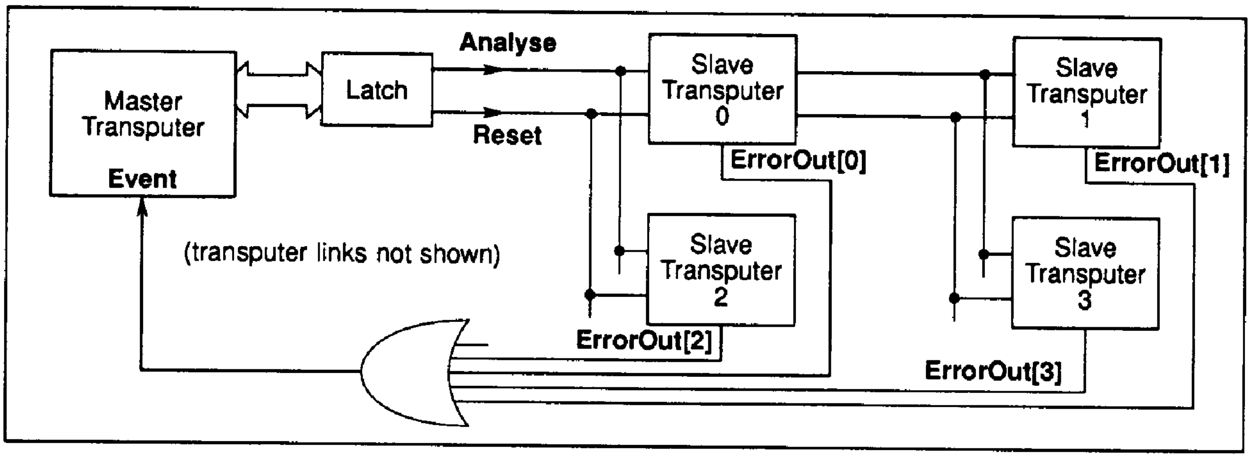


Figure 6.5 Error handling in a multi-transputer system

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 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. 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.

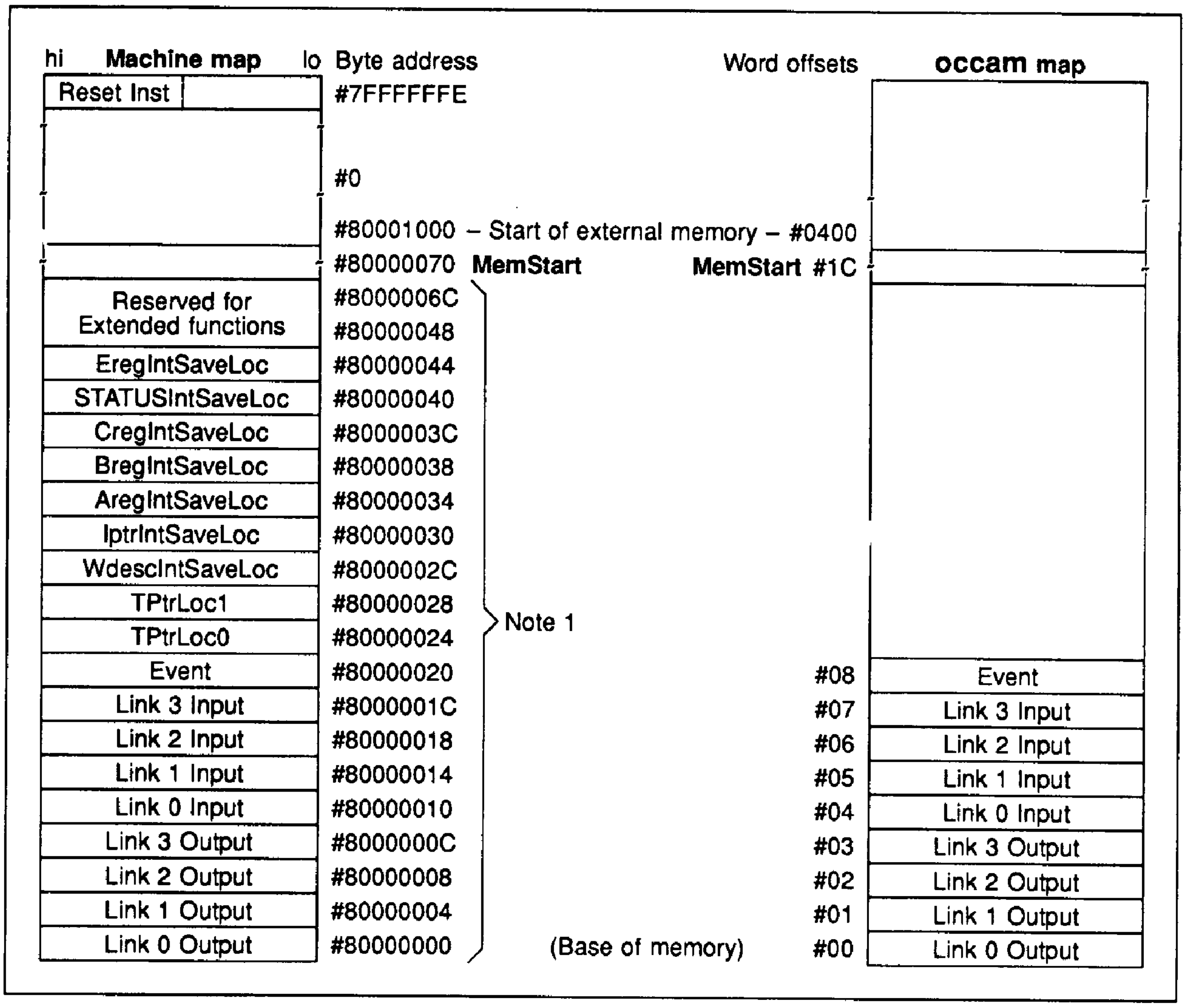


Figure 7.1 IMS T801 memory map

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).

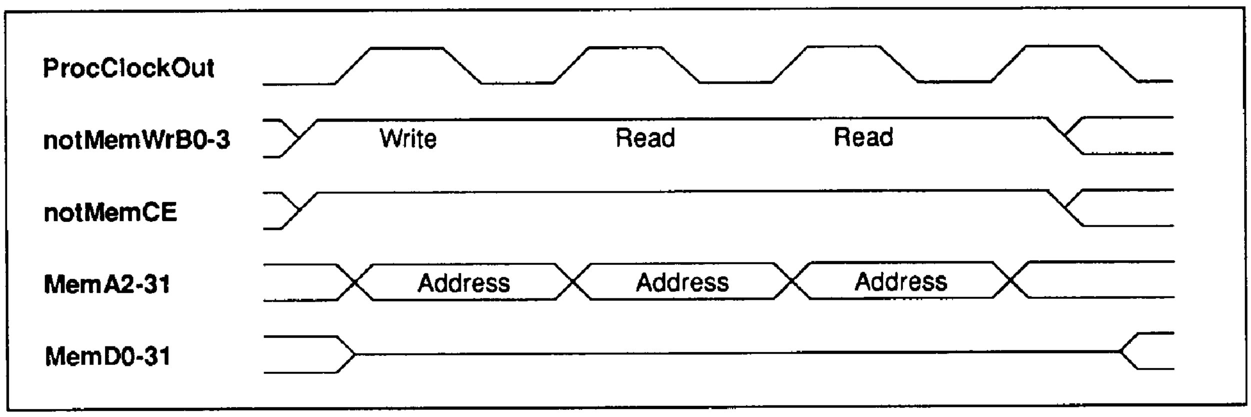


Figure 8.1 IMS T801 bus activity for 3 internal memory cycles

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

SYMBOL	PARAMETER	T801-30		T801-25		T801-20		NOTE
		MIN	MAX	MIN	MAX	MIN	MAX	
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.

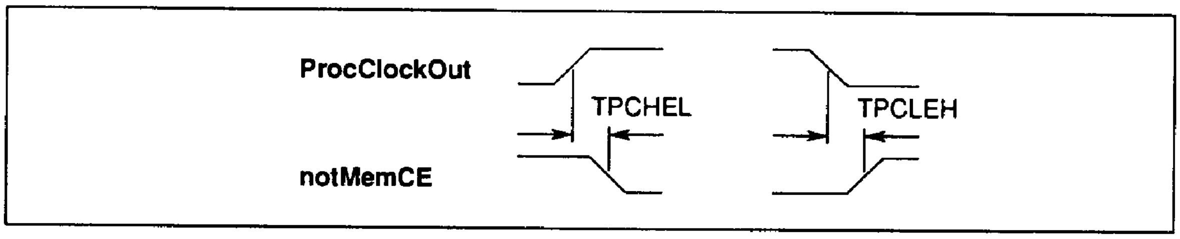


Figure 8.2 IMS T801 skew of **notMemCE** to **ProcClockOut**

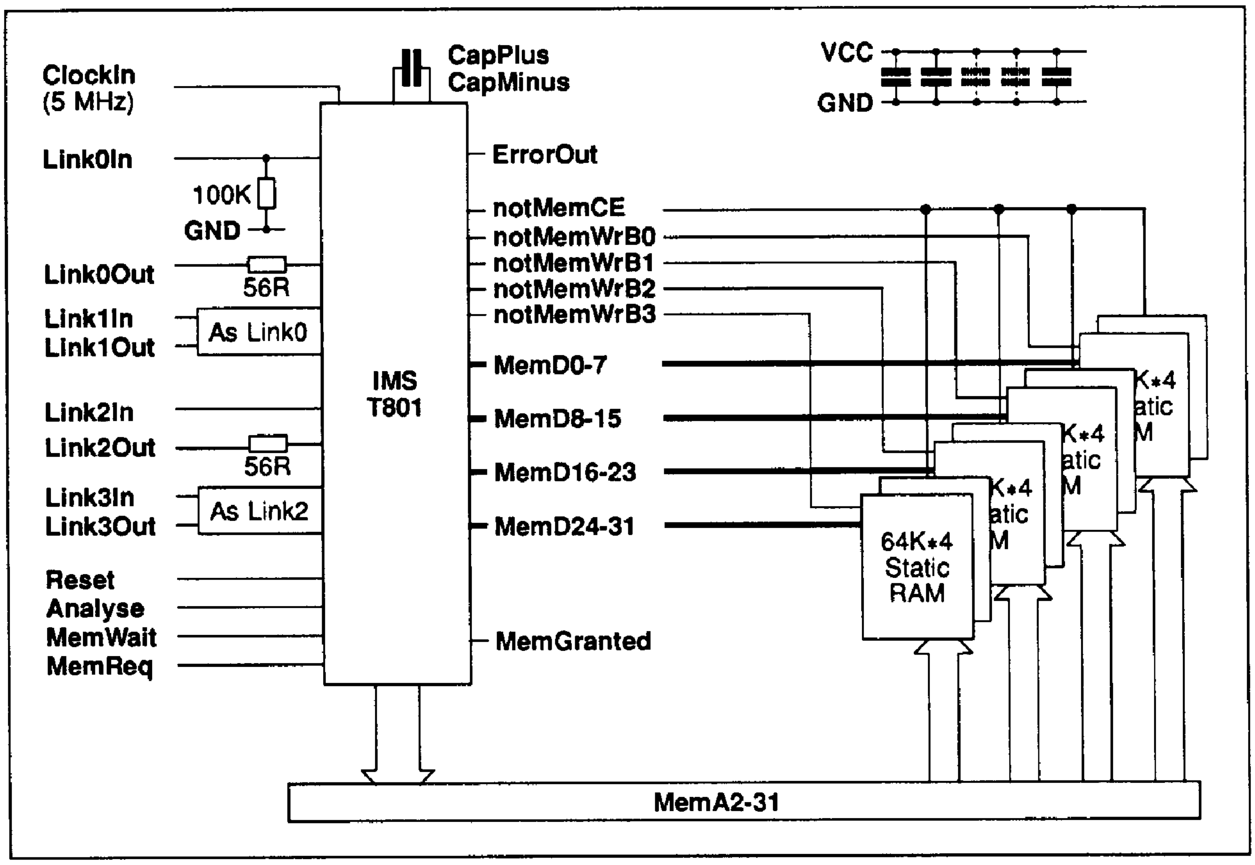


Figure 8.3 IMS T801 static RAM application

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 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**.

8.1.7 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 **ProcClockOut**

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTE
TPCLPCL	ProcClockOut period	a-1	a	a+1	ns	1
TPCHPCL	ProcClockOut pulse width high	b-2.5	b	b+2.5	ns	2
TPCLPCH	ProcClockOut pulse width low		c		ns	3
TPCstab	ProcClockOut stability			4	%	4

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.

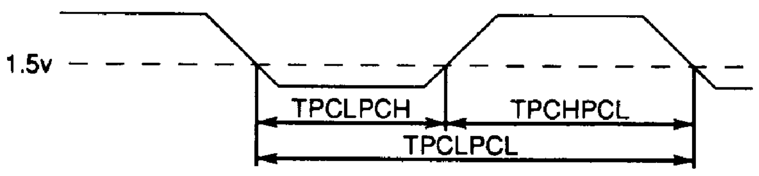


Figure 8.4 IMS T801 ProcClockOut timing

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.

Table 8.3 Read cycle

		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 low	14		16		20		3,4
TPCHEL	ProcClockOut high to chip enable low	6		8		10		1,3

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 Units are ns.

4 Timing is for all four write enables **notMemWrB0-3**.

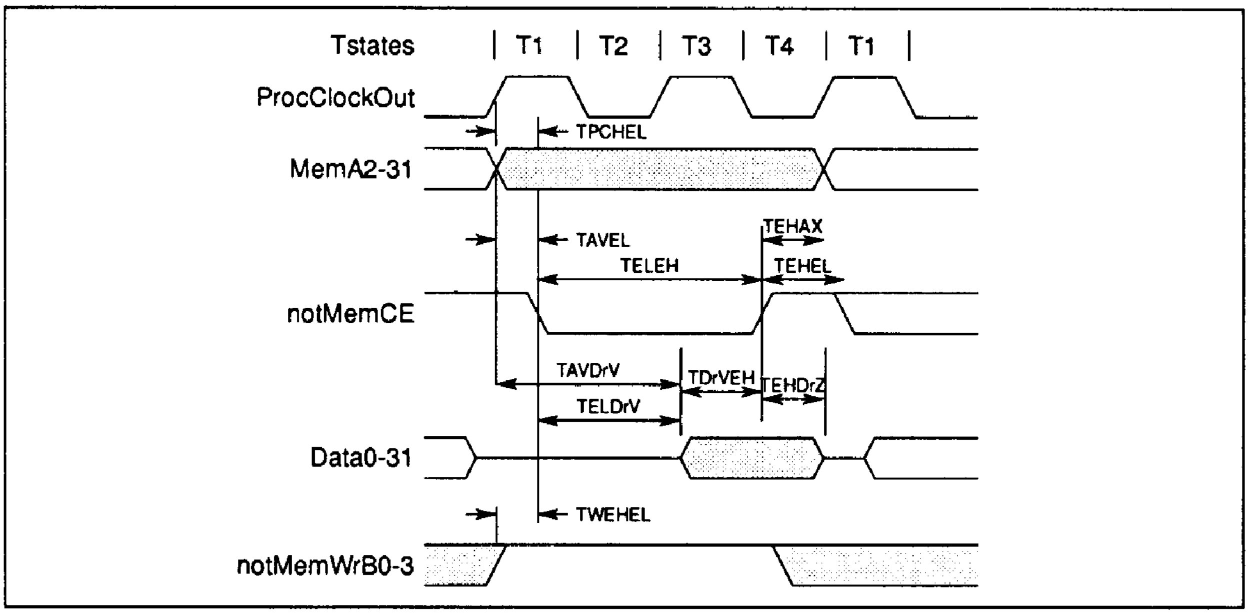


Figure 8.5 IMS T801 external read cycle

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.

Table 8.4 Write cycle

SYMBOL	PARAMETER	T801-30		T801-25		T801-20		NOTE
		MIN	MAX	MIN	MAX	MIN	MAX	
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

Notes

1 Units are ns.

2 Timing is for all four write enables **notMemWrB0-3**.

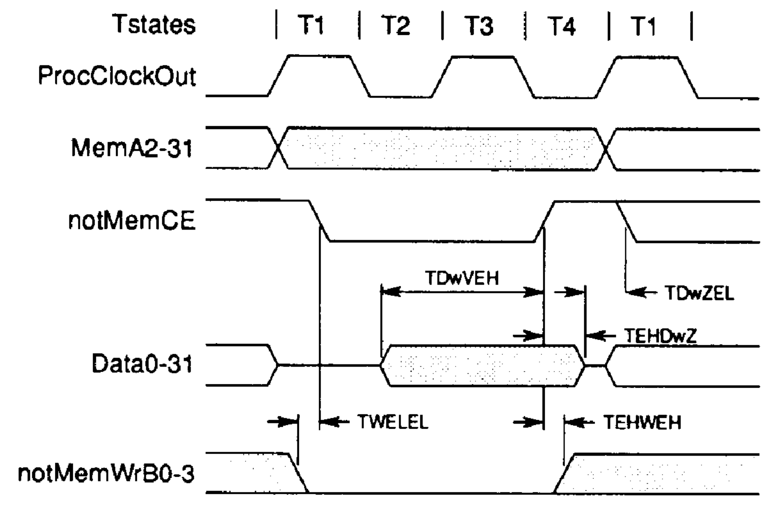


Figure 8.6 IMS T801 external write cycle

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.

Table 8.5 Memory wait

SYMBOL	PARAMETER	T801-30 MIN	T801-30 MAX	T801-25 MIN	T801-25 MAX	T801-20 MIN	T801-20 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

Notes

1 Units are ns.

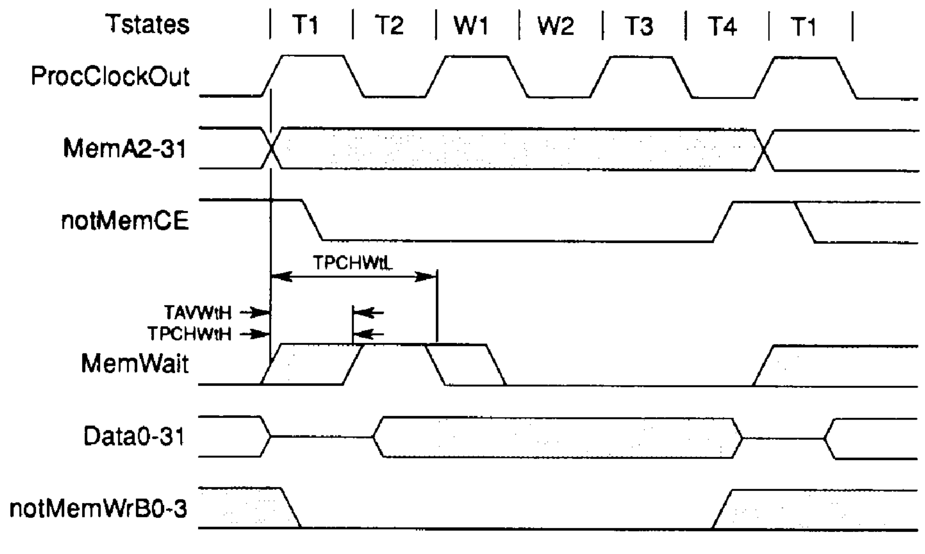


Figure 8.7 IMS T801 memory wait timing

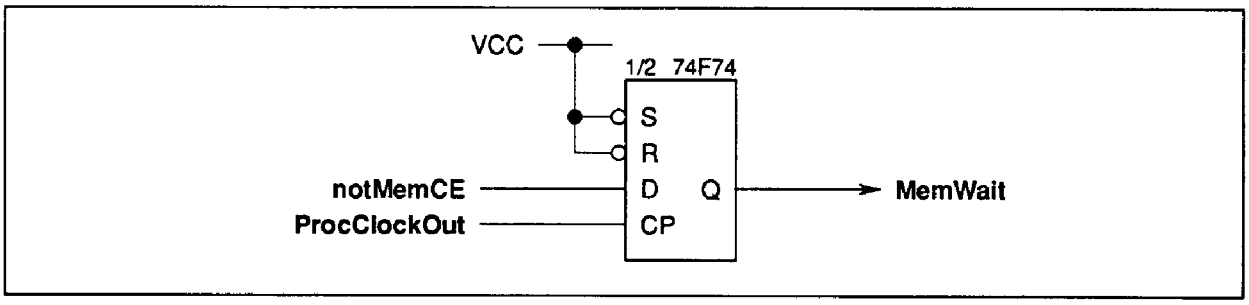


Figure 8.7 Single wait state generator

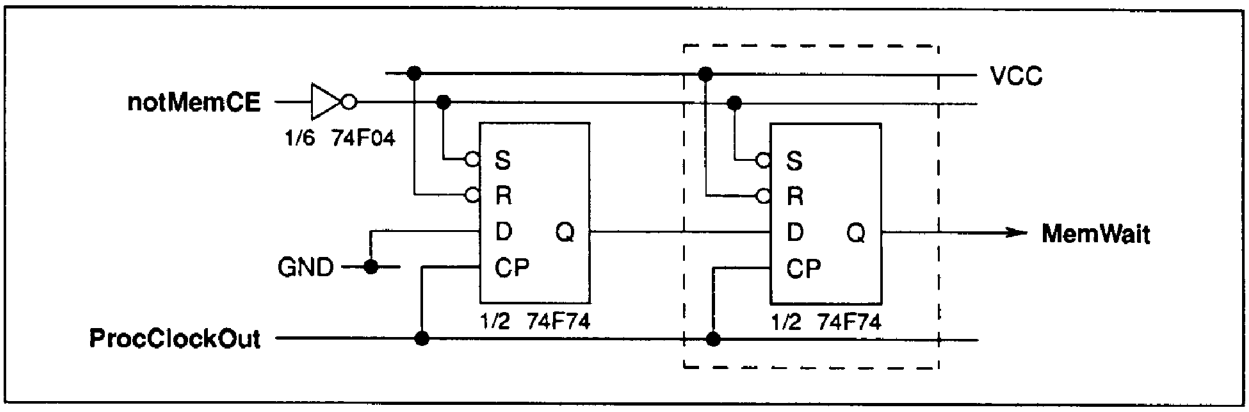


Figure 8.8 Extendable wait state generator

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 **notMemCE** 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.

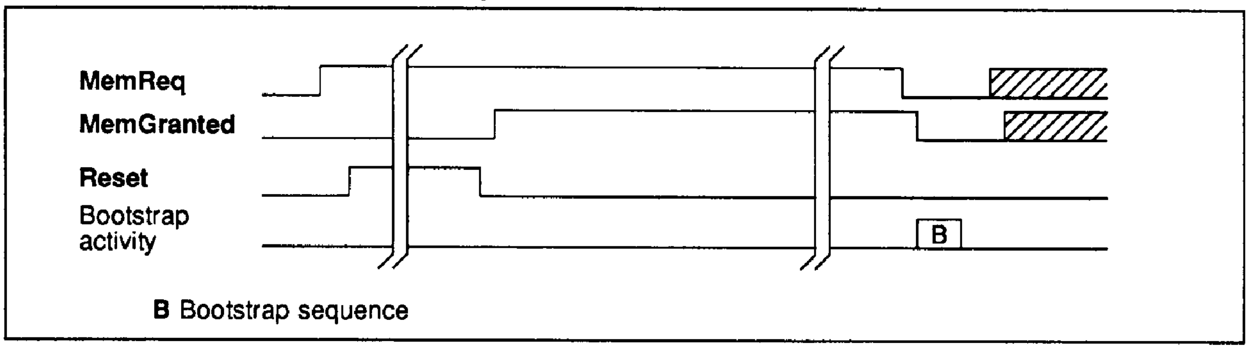


Figure 8.9 IMS T801 DMA sequence at reset

Table 8.6 Memory request

SYMBOL	PARAMETER	T801-30		T801-25		T801-20		NOTE
		MIN	MAX	MIN	MAX	MIN	MAX	
TMRHMGH	Memory request response time	58	a	70	a	85	a	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

Notes

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.**

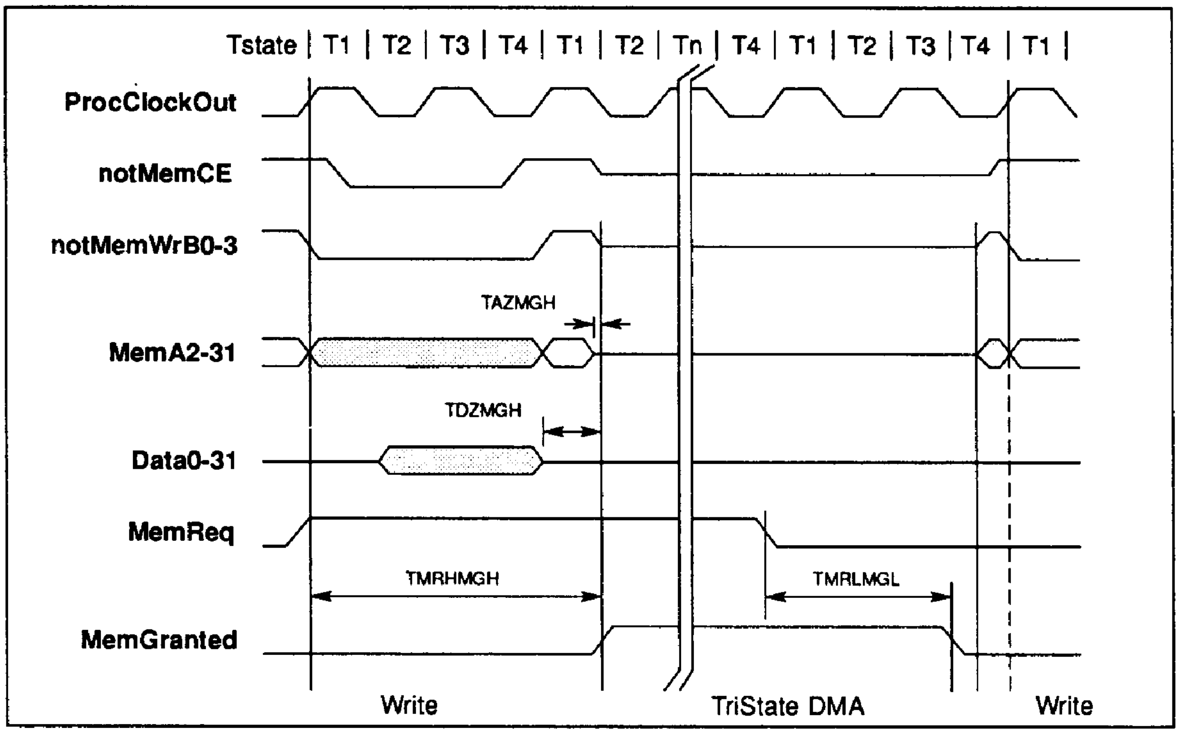


Figure 8.10 IMS T801 memory request timing

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			ns	
TVLKL	Delay before removal of event acknowledge	0		a+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**.

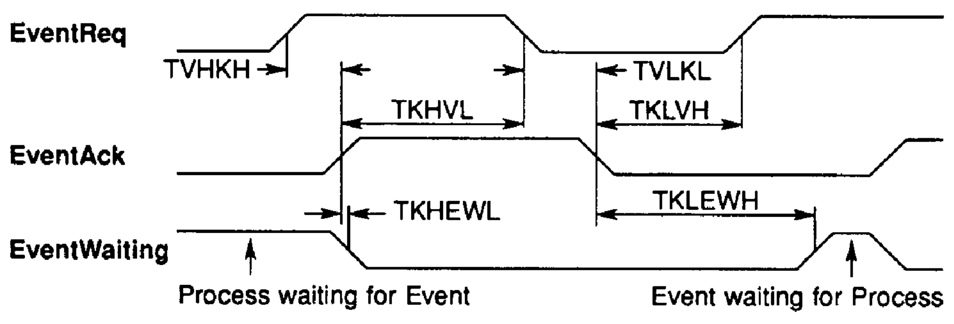


Figure 9.1 IMS T801 event timing

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

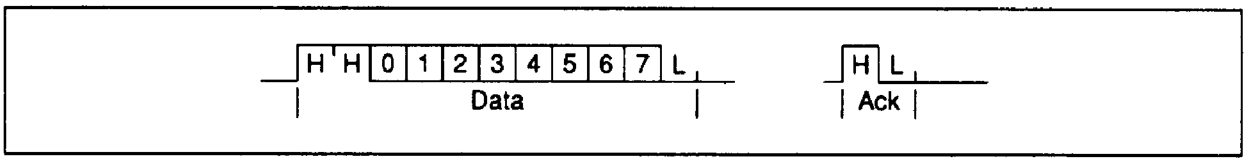


Figure 10.1 IMS T801 link data and acknowledge packets

Table 10.2 Link

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			7	pF	1
CLL	LinkOut load capacitance				50	pF	
RM	Series resistor for 100Ω transmission line			56		ohms	

Notes

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.

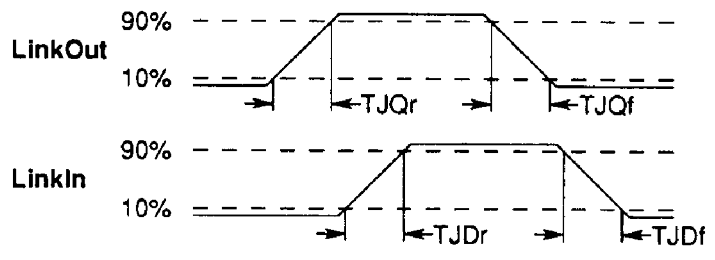


Figure 10.2 IMS T801 link timing

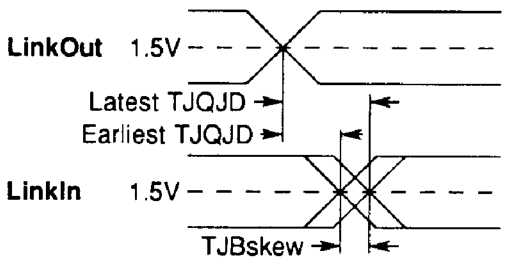


Figure 10.3 IMS T801 buffered link timing

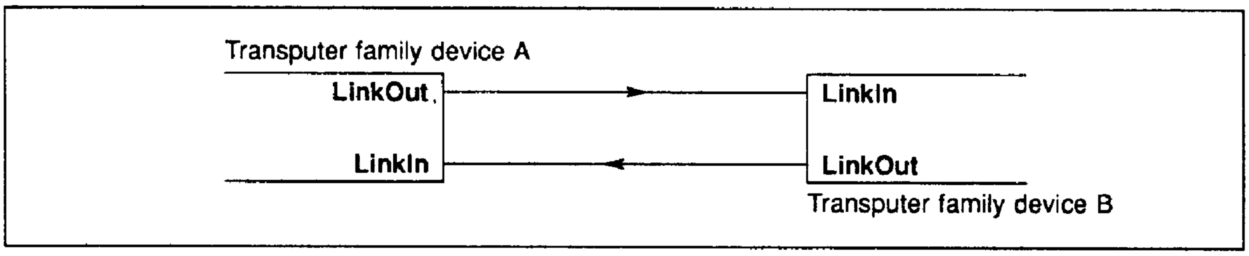


Figure 10.4 IMS T801 Links directly connected

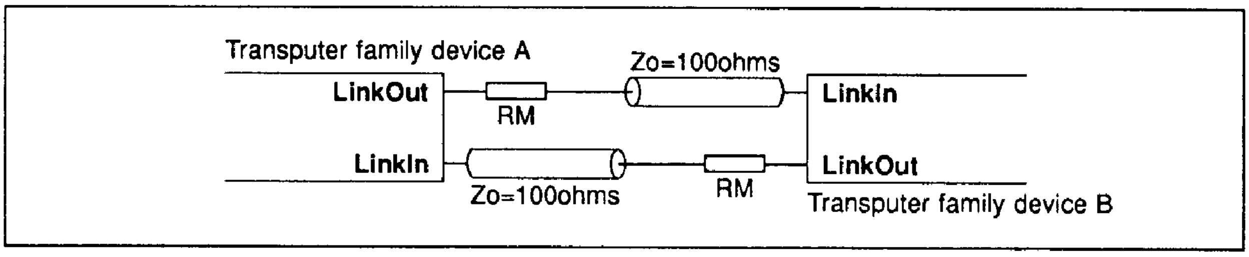


Figure 10.5 IMS T801 Links connected by transmission line

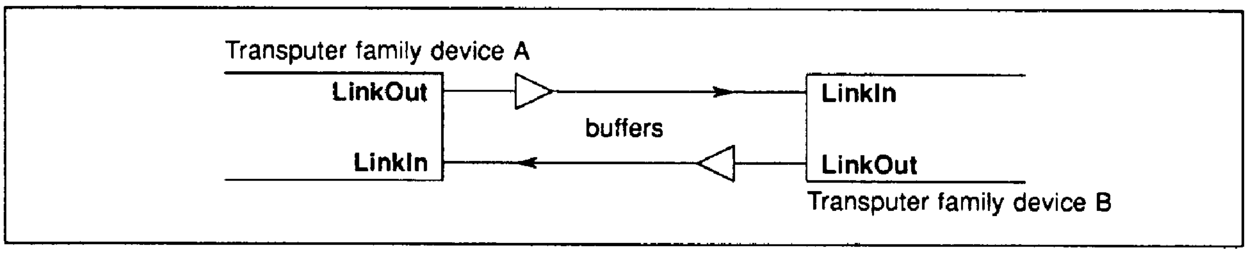


Figure 10.6 IMS T801 Links connected by buffers

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	V	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	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	0	70	°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.

Table 11.3 DC characteristics

SYMBOL	PARAMETER	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
II	Input current @ GND<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	36	65	mA	1,2,3,6
		65	100	mA	1,2,4,6
IOZ	Tristate output current @ GND<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

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.

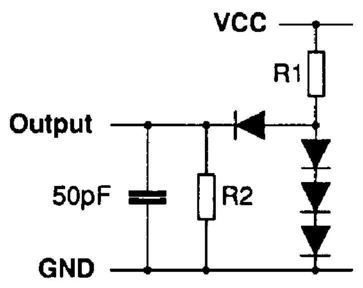
3 Current sourced from non-link outputs excluding **ProcClockOut**.

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



Load for:	R1	R2	Equivalent load:
Link outputs	1K96	47K	1 Schottky TTL input
Other outputs	970R	24K	2 Schottky TTL inputs

Diodes are 1N916

Figure 11.1 Load circuit for AC measurements

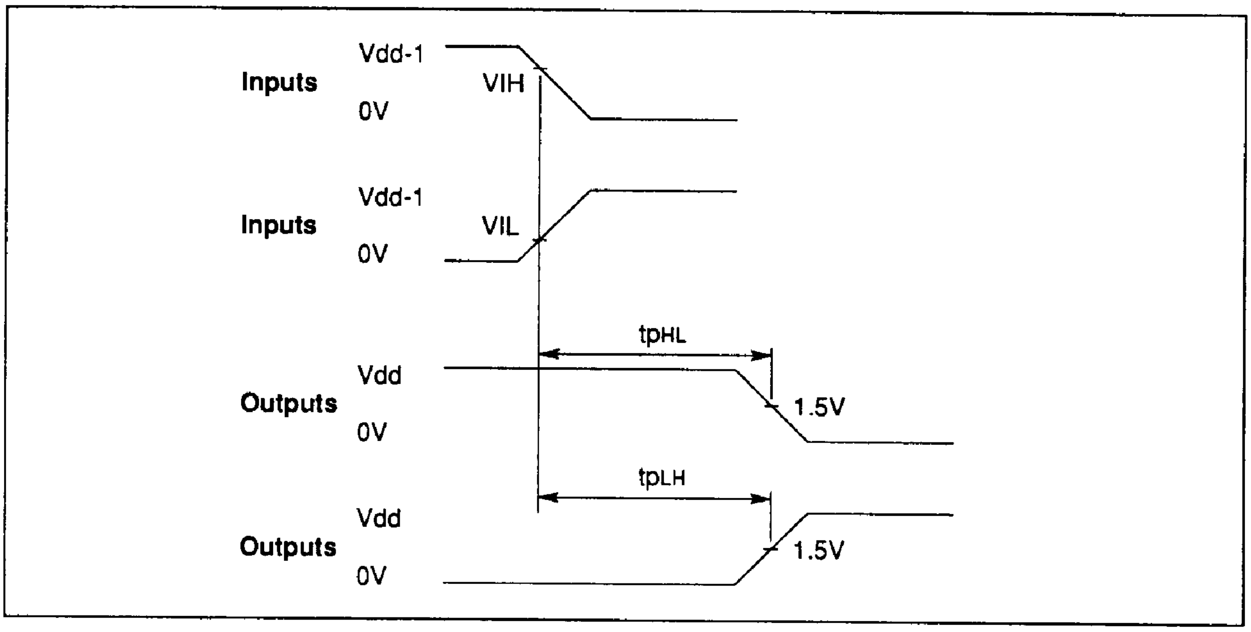


Figure 11.2 AC measurements timing waveforms

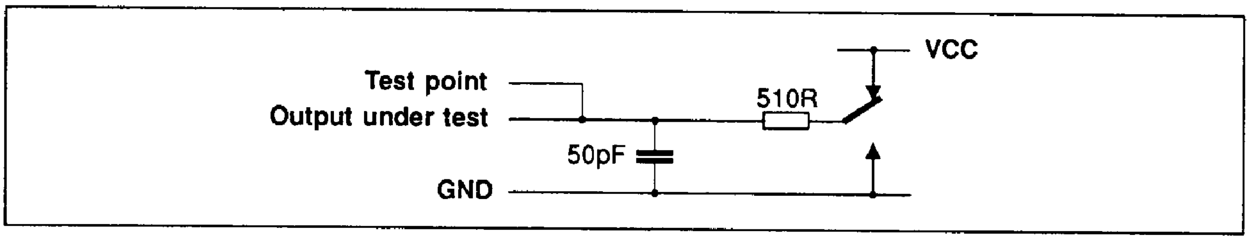


Figure 11.3 Tristate load circuit for AC measurements

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

Notes

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.

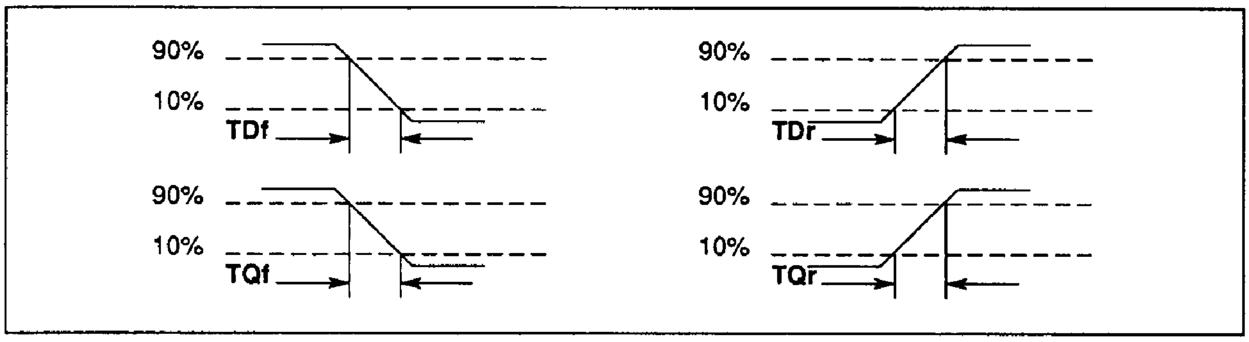


Figure 11.4 IMS T801 input and output edge timing

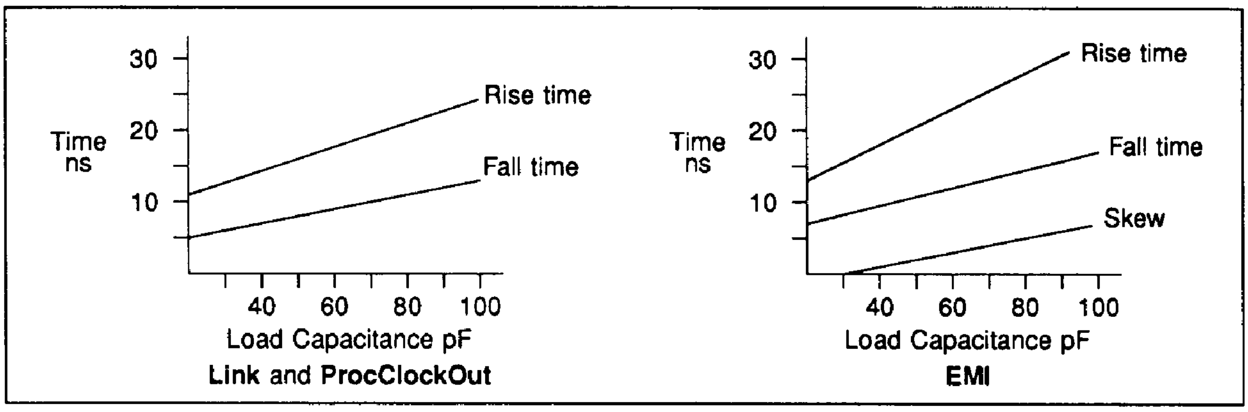


Figure 11.5 Typical rise/fall times

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 θJ_A is the junction-to-ambient thermal resistance in °C/W. θJ_A for each package is given in the Packaging Specifications section.

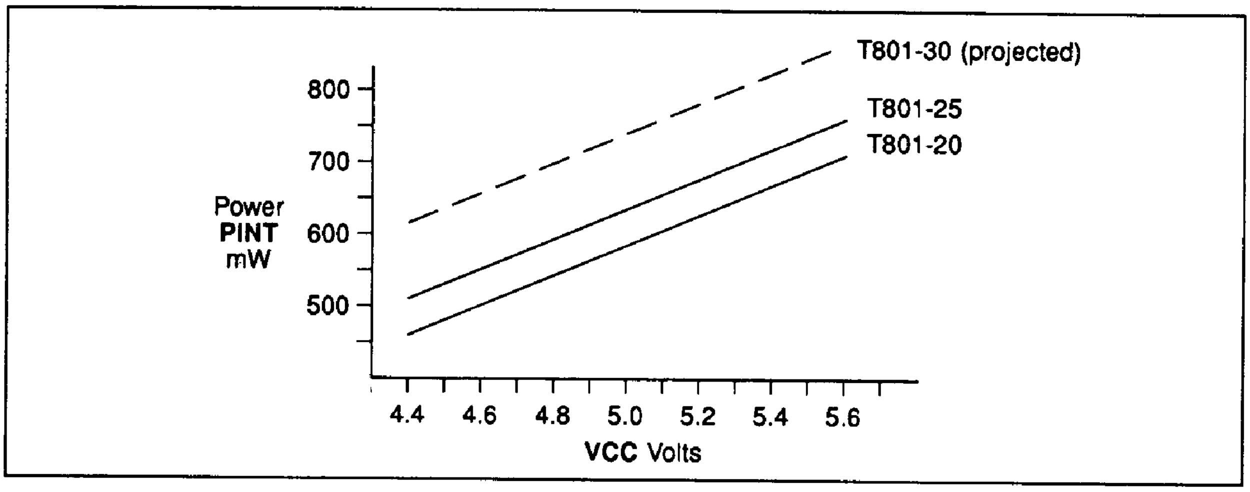


Figure 11.6 IMS T801 internal power dissipation vs VCC

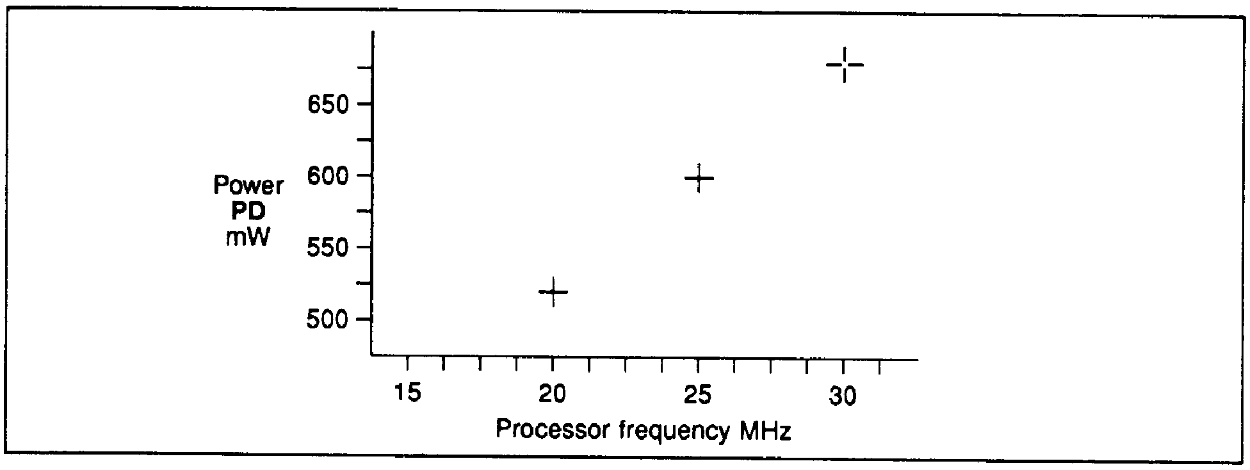


Figure 11.7 IMS T801 typical power dissipation with processor speed

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
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

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***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
>> <<	2	3+p
Modulo Arithmetic operators		
PLUS	2	2
MINUS	1	1
TIMES (fast multiply, positive operand)	1	4+Tbp
TIMES (fast multiply, negative operand)	1	5+Tbc
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		
/\ \/ >< ~	2	2
Expressions		
constant in expression	w	w
check if error	4	6

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	31
non-empty timer queue	2	38+ne*9
ALT (timer)		
with empty timer queue	6	52
non-empty timer queue	6	59+ne*9
timer alt guard	8+2Eg+2Et	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+2Eg	20+2Eg
skip alt guard	8+2Eg	10+2Eg
PAR	11.5+(np-1)*7.5	19.5+(np-1)*30.5
WHILE	4	12
Procedure or function call		
	3.5+(nsp-2)*1.1 +nap*2.3	16.5+(nsp-2)*1.1 +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.

Table 12.4 Arithmetic performance

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
	(n>=32)	n-27	
SHIFTLEFT	(n<32)	4+n	8
	(n>=32)	n-27	
NORMALISE	(n<32)	n+6	7
	(n>=32)	n-25	
	(n=64)	4	
ASHIFTRIGHT		SHIFTRIGHT+2	5
ASHIFTLEFT		SHIFTLEFT+4	5
ROTATERIGHT		SHIFTRIGHT	7
ROTATELEFT		SHIFTLEFT	7
FRACMUL		LONGPROD+4	5

† Assuming local variables.

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.

Table 12.5 Floating point performance

	Size (bytes)	REAL32 Time (cycles)	REAL64 Time (cycles)
Names			
variables			
in expression	3.1	3	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	17	32
REM	11	19	34
Comparison operators			
=	2	4	4
<>	3	6	6
> <	2	5	5
>= <=	3	7	7
Conversions			
REAL32 to -	2		3
REAL64 to -	2	6	
To **INT32** from -	5	9	9
To **INT64** from -	18	32	32
INT32 to -	3	7	7
INT64 to -	14	24	22

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.

Table 12.6 IMS T801 floating point arithmetic performance

Function	Cycles	+ cycles for parameter access †	
		REAL32	**REAL64**
ABS	2	8	
SQRT	118	8	
DABS	2		12
DSQRT	244		12

† 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+Tb	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**. 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 **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**.

Table 12.10 External memory performance

	IMS T801	
	Program off chip	Data off chip
Boolean expressions	e-2	0
IF	3en-8	en
Replicated **IF**	(6e-4)n+7	(5e-2)n+8
Replicated **SEQ**	(3e-3)n+2	(4e-2)n
PAR	(3e-1)n+8	3en+4
Replicated **PAR**	(10e-8)n+8	16en-12
ALT	(2e-4)n+6e	(2e-2)n+10e-8
Array assignment and communication in one transputer	0	max (2e, e(b/2))

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.

Table 12.11 IMS T801 external memory performance

	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

12.6 Interrupt latency

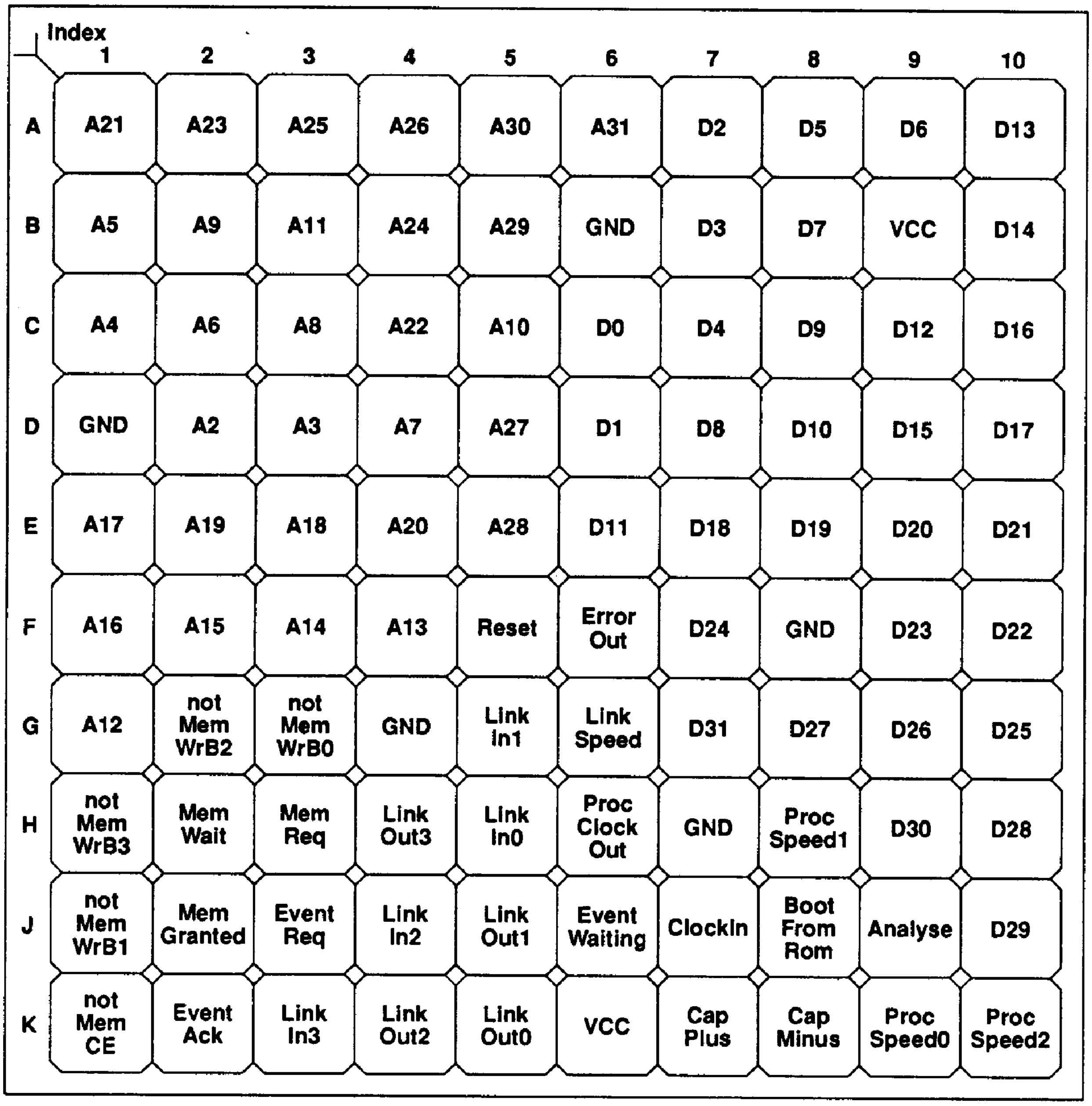
If the process is a high priority one and no other high priority process is running, the latency is as described in table 12.12. The timings given are in full processor cycles **TPCLPCL**; the number of **Tm** states is also given where relevant. Maximum latency assumes all memory accesses are internal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

Table 12.12 Interrupt latency

	Typical		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

13 Package specifications

13.1 100 pin grid array package



Index	1	2	3	4	5	6	7	8	9	10
A	A21	A23	A25	A26	A30	A31	D2	D5	D6	D13
B	A5	A9	A11	A24	A29	GND	D3	D7	VCC	D14
C	A4	A6	A8	A22	A10	D0	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
H	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
K	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**

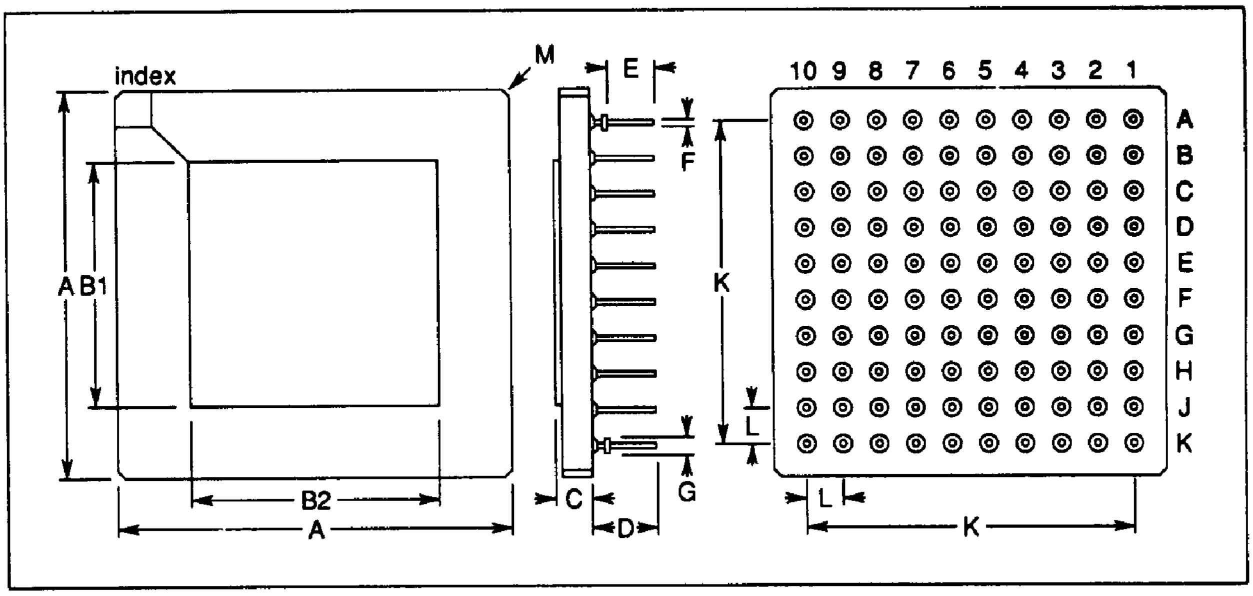


Figure 13.2 100 pin grid array package dimensions

Table 13.1 100 pin grid array package dimensions

DIM	Millimetres		Inches		Notes
	NOM	TOL	NOM	TOL	
A	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	
C	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	
M	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