The IMS T810

A Preliminary Survey

Guy Harriman

January 1989

Contents

IMS T810 Preliminary Specification

Guy Harriman
January 13, 1989

Introduction

The IMS T810 will be processed on the shrunk transputer CMOS process, and therefore speed distribution should extend beyond 30MHz.

The IMS T810 will reuse, unaltered from the T800, as much of the logic as possible. Hence the objective is to reuse the FPU and the internal RAM without alteration, and only alter the links to increase the data rate (if that is possible).

Process

The process to be used is the 0.8X shrunk transputer process. This process features one level of metal, and one level of silicide, and has been used successfully since 1985 at Duffryn.

The 0.8X shrunk transputer process has been in development with the T800e since May 1988. It is expected that working silicon might be produced by March 1989. This silicon is to be used to characterise the shrunk process, and the results of this characterisation will determine what can be done on the T810 to increase its speed relative to the T800d.

Packaging

The initial package to be used for the T810 will be either a 121 pin PGA or a 144 pin PGA. These packages have an area of 1.1 inch by 1.1 inch, and 1.2 inch by 1.2 inch respectively, representing a modest increase in area over the T801 PGA package (an area of 1 inch by 1 inch). The advantage of the PGA package is that it can be easily be" assembled and handled, and is ideal for prototyping and small-scale manufacturing (for example, for the Inmos boards group). Experience of problems in qualifying transputers in non-PGA packages since 1985 has been so bad that it is not sensible to try to sample customers with a plastic, non-cavity package.

It is expected that the new JEDEC standard Quad Flat Pack, which has gull-wing leads on 0.025 inch pitch, and is an area-efficient cavity package, will become the package of choice by the mid 1990's. If Inmos can demonstrate

the ability to assemble and handle this package, then the T810 would be a good candidate for the Quad Flat Pack for higher manufacturing volumes. The standard JEDEC sizes include 132 and 164 pins.

CPU

The CPU has been enlarged from the T800 to take into account the extra control logic and datapath area required for single cycle *ldl*, *lb*, and *sb* instructions. Extra area has been allowed for improving the scheduler microcode. Area has been reserved for reimplementing functions such as the instruction prefetch mechanism to improve its speed.

Multiplier

The array multiplier itself is about 3mm by 3mm, with another 2mm by 3mm allowed for control logic and routing. This fits outside the CPU, and does not impact the cycle time of the CPU. The multiplier is driven from the A and B registers directly, to maximise the speed of the multiplier.

EMI

The EMI has a complex function, with banks of addresses being compared against the memory access address. The datapath style of design is likely to result in an area of 2mm by 2mm, although some of the control logic for the EMI may have to be located in the CPU.

The multiple bank design of the EMI will result in a regular, repeated, structure. This will ensure that the silicon layout will be efficient, and relatively quick to implement.

Links

There will be eight links; this will take twice the area of the four links on the T800, and will fit below the CPU (as on the T800). The datapath, steering logic, handshake logic, round robins, byte level logic, sub-byte level logic, and sub-bit level logic are all layout out to a height of 600 microns (at 1X). Therefore eight links will require a height of 4,800 microns.

There will be a single transfer engine, but the space will be available to have more than one Event channel.

The provision of eight links allows direct connection of up to eight transputers, which is sufficient for many embedded system applications. Eight links also allows an order eight hypercube to be constructed (256 nodes). Furthermore, the data rate can be doubled by using two links instead of one for a mesh topology. In order to remove the software overhead from sharing messages on a number of links it will be necessary to allow links to be shared by a single process.

The link data rate will be made as high as possible; this will be determined by engineering investigation of the current generation transputers. The target is a link data rate of 30MBaud or 40MBaud.

Internal RAM

There will be 16KBytes of internal static RAM. The purpose of having this amount of RAM is ensuring that the local workspaces of even the largest programs will fit. This has a significant effect on performance, as has been demonstrated on previous transputers. A much larger number of highly parallel programs would fit entirely in 16KBytes than in 4KBytes, making it more attractive to sell arrays of transputers without external memory. The redundancy incorporated in the internal RAM will allow a good yield to be maintained, in spite of the area taken.

The 16KBytes will be organised as four 4KByte static RAMs; the data and address buses will be shared between them. These RAMs will have the same redundancy circuitry as the T414 and T800 internal RAMs, although it has been demonstrated that the row redundancy achieves very little compared with column redundancy. The cost of removing the row redundancy, with the associated risk of changing the circuitry, is too high, as the area saved would be very small.

FPU

The FPU will be the same as the FPU on the T800, and will fit above the internal RAMs. It is intended that no new design work is spent on the FPU, unless it is shown from the characterisation of the T800e (the 0.8X shrunk transputer) that critical paths exist in it which are not present on the T800d (1X transputer).

Clocks

Clock distribution will be critical on the T810, in order to achieve the best speed possible on the device. Unlike the T800, where the clock generator is in the pad ring, the T810 will have a much larger clock generator, and it will be situated in the centre of the die. This minimises the clock skew between any parts of the die, which will be crucial to maximising the speed of the T810.

Phase-Locked Loops

The phase-locked loops will be upgraded to the standard of the G300 phase-locked loop, to provide a wider range of operating frequency and lower jitter.

CPU Specification

- Object Code Compatible with IMS T800.

  The T810 will have all the instructions present on the T801 and T805 transputers, with the same entry points. The compilers for the T800 transputer family will not have to be altered in order to execute programs on the T810.

- Faster CPU with Optimised Microcode.

  The majority of the design work on the T810 will be spent on improving the speed of operation of the CPU. Extra hardware in the control logic and datapath will be used where possible to speed up instruction execution, as long as that hardware does not slow the cycle time of the CPU down.

- Single Cycle (Execution and Memory Access) *ldl*, *stl*.

  An important goal of the T810 design is to reduce the time taken to execute *ldl*, as this is dynamically the most important instruction. The same hardware mechanism will be used to reduce the execution time of *stl*.

- Other Primary Instructions Speeded Up.

  More logic will be added to the instruction streamer to speed up the most important instructions. In particular, predecoding hardware will be used to reduce the number of cycles taken in as many primary instructions as possible.

- Enhanced Instruction Buffer.

The instruction buffer will have 16 words of instructions, and the user will have control of the number of words of prefetch. The larger number of words allows unused memory cycles to be taken advantage of.

The instruction buffer will present two instructions rather than one as on previous transputers, to allow some instructions to be executed in hardware, in parallel with instructions executed in the datapath.

- Hardware Predecode of *pfix* and *nfix*.

  The prefix instructions *pfix* and *nfix* will be decoded and executed in hardware. A prefix instruction will be executed in parallel with the previous instruction, and so take no time to execute. This means that it will be equally efficient to access any local variable in the first 256 workspace locations (compared with the first 16 on previous transputers), and that long literals will be assembled in the operand register in half the number of cycles. The other effect of having the prefix instructions executed in hardware is that the prefixed (long) secondary instructions will be decoded in the same time as the short secondary instructions. Note that for two prefix instructions in sequence, the first prefix instruction will be executed in hardware while the second is executed in the datapath at the same time.

- Single Cycle *lb* and *sb*.

  The speed of single byte access is particularly important for graphics and character handling. Although block move is efficient for handling contiguous groups of bytes, individual byte manipulation is too slow on previous transputers (4 and 5 cycles).

  Single cycle execution of *sb* will be implemented. The *lb* instruction will take two cycles to execute, including the memory read cycle.

- Fast Multiply and Shift Instructions.

  One of the key features of the T810 is that it will include a single cycle multiplier. The multiplier architecture will be the fastest possible, as the speed of the array is more important than the size. The array multiplier will be used to implement all the multiply instructions in the instruction set (*mul*, *prod*, and *lmul*). The *mul* instruction will take two cycles to execute, as it must evaluate the overflow condition. The *prod* instruction does not evaluate overflow, and will therefore execute in a single cycle. The *lmul* instruction uses the 64 bits of the multiplier result, which must be added to the initial value in the C register. The *lmul* instruction will execute in two cycles.

  The most important requirement in multiplication for the new generation of microprocessors (particularly in DSPs) is that it should be

performed in floating point. The array multiplier will be used to perform the mantissa part of the multiplication. The exponent addition for the multiplication will be performed by a separate 8 bit adder. The floating point multiplication will be single length, conforming to IEEE 754.

The shift instructions will also utilise the multiplier array, by taking the binary exponent of the shift length n to form a number 2^n which is used as a multiplier. The shift left (*shl*) instruction is implemented by multiplying by 2^n, and the shift right (*shr*) instruction is implemented by multiplying by 2^{-n} and taking the more significant word of the result. Both the single length instructions will execute in two cycles. The double length shift instructions will perform a conditional word shift in the CPU stack before using the multiplier array; it will execute in four cycles.

- Faster Scheduling; More User Flexibility in Scheduling.

  The scheduling microcode will be optimised to take advantage of any added hardware in the CPU.

  One of the major problems which has led to the failure of the transputer being used in high volume real-time systems is the lack of a standard, popular, real-time kernel such as VRTX from Ready Systems Inc. It is not possible to implement VTRX with the microcoded scheduler on the transputer, as there is no way for a kernel process to know that a user process has been scheduled. Therefore, the scheduling microcode must be altered to add 'hooks' for the user. The 'hooks' consist of indirections through user routines. The indirection is performed using the context swap mechanism of the breakpointing support, to make it consistent with breakpointing. The kernel places the code which manages the process queue in the context swap location. Note that VRTX allows precisely the same to be done itself - the user can place his own code in the scheduling points of VRTX.

- Internally Controlled Microcode Rom Dump to Register.

  In order to improve the testability of the T810 in a p.c.b., it is desirable to to be able to do a check of the microcode rom contents without the need for external test logic. It is intended that this facility will be added to the T810, if timescales permit.

- Error State per Process.

  The Error state will be held in the unused bit of the workspace descriptor (WDesc[1]). This is required because in previous transputers, Error was global to the processor, and this prevented the user from handling errant low priority processes with a high priority supervisor

process. There will not be a sixteen bit version (which does not have an unused bit in the workspace descriptor) of the T810, as there is no economic rationale for such a device.

New instructions

- *ldhw* - Half-Word Load

  Load A register with half-word value; executes in 2 cycles.

- *sthw* - Half-Word Store

  Store half-word value from A register; executes in 1 cycle.

- *macc* - Integer Multiply-Accumulate

  Multiply and accumulate integer result; executes one multiply and one addition in two cycles (achieves 15 MOPS on T810-30), in parallel with accessing the data and coefficient words. Sustains this MOPS rating as long as data and coefficient are in internal RAM.

- *fpmacc* - Floating Point Multiply-Accumulate

  Multiply and accumulate single length IEEE 754 floating point result; executes one multiply and one addition in three cycles (achieves 20 MFLOPS on T810-30), in parallel with accessing the data and coefficient words. Sustains this MFLOPS rating as long as data and coefficient are in internal RAM.

- *xprod* - Integer Cross Product

  Multiply two integer values and store result; executes one multiply and one store in three cycles, in parallel with accessing the data and coefficient words. Sustains this speed as long as data and coefficient and result vectors are in internal RAM.

- *fpxprod* - Floating Point Cross Product

  Multiply two single length values and store result; executes one multiply and one store in four cycles, in parallel with accessing the data and coefficient words. Sustains this speed as long as data and coefficient and result vectors are in internal RAM.

- *delay* - Delay

  Wait for A register cycles; this is required because the shift instructions on the T810 do not perform that function, as they do on previous transputers.

- *dislinkinth* - Disable of High Priority Link Interrupts

  Prevent high priority links from rescheduling after completion of link data transfer.

- *dislinkintl* - Disable of Low Priority Link Interrupts

  Prevent low priority links from rescheduling after completion of link data transfer.

- *enlinkinth* - Enable of High Priority Link Interrupts

  Allow high priority links to reschedule after completion of link data transfer.

- *enlinkintl* - Enable of Low Priority Link Interrupts

  Allow low priority links to reschedule after completion of link data transfer.

- *distimesl* - Disable Timeslicing

  Prevent timeslicing of low priority processes.

- *entimesl* - Enable Timeslicing

  Allow timeslicing of low priority processes.

- *pause* - Pause Process

  Force process to be rescheduled (placed at the back of the process queue).

- *checkaddr* - Check If Address In Internal RAM.

  Test if the A register holds an address which is in one of the internal RAMs. Returns TRUE or FALSE in the A register.

Of the above instructions, *ldhw* and *sthw* require support in C compilers for the T810. The *macc*, *fpmacc*, *xprod*, and *fpxprod* instructions can be accessed from assembler satisfactorily, although the provision of simple predefined procedures (which merely invoke the instructions, as in the Move2D predefined procedures for the T800), would be tidier.

The purpose of the *dislinkinth*, *dislinkintl*, *enlinkinth*, *enlinkintl*, *distimesl*, and *entimesl* instructions is to complement the timerenableh, timerenablel, timerdisableh, and timerdisablel instructions of the T425, T801, and T805, in allowing the user to implement atomic (noninterruptable) functions such as memory allocation and semaphores. The *pause* instruction gives the user more control of the scheduling of processes, particularly in an environment where timeslicing has been disabled. These instructions can be accessed in

assembler adequately, as they are low level functions of no interest to the average programmer.

The *checkaddr* instruction supports the use of the relocatable internal RAMs, and is provided to make the allocation of the internal RAMs by system service routine more efficient. This instruction can be accessed in assembler adequately, as it is a low level function.

Internal RAM Specification

The 16KByte internal RAM will be organised as four independent 4KByte RAMs each operating with the same timing, and having a cycle time of one processor cycle.

The major use is to allow the location of the RAMs to be altered dynamically by the programmer; this gives the C programmer a 'heap' of single cycle memory in 4KByte blocks which can be independently allocated and freed. In particular, the internal RAMs could be used to hold the local variables of the dynamically most common procedures, by placing them in the stack area.

Each RAM has a register to define the address of the least significant location in the memory, and a register to define the address of the most significant location in the memory. For each memory address provided by the CPU, the four internal RAMs compare that address with the values in their two registers, to determine whether the address lies within their range. A memory can be disabled by setting the least significant address register to the same value as the most significant address register.

It will be possible for the internal RAMs to be accessed while external memory is in use; this will significantly improve performance with slow external memory.

External Memory Interface Specification

The external memory interface (EMI) of the T810 will have two modes, externally selectable by a pin. The simple mode will be identical to the T801 non-multiplexed, non-programmable memory interface, with the addition of user selectable static column mode. One purpose of the non-mux mode is to provide the user with the ability to interface to any type of memory system, no matter how complex, without any timing constraint. Another purpose of the non-mux mode is to provide the fastest and simplest interface to static RAMs.

The second EMI mode is a high performance, highly programmable inter-

face to DRAMs or mixed systems including DRAMs, VRAMs, SRAMs, and EPROMs. This mux mode is primarily intended for embedded system and cost sensitive applications, where zero external logic is an important requirement. The mux mode is targeted primarily at DRAMs, while not penalising the interface to SRAMs. The mux mode takes advantage of static column mode, which is provided on all DRAMs of the 1MBit and above generations.

The following features are the main points of the mux mode:

- 32 Data Pins, 16 Address Pins.
- Multiplexed RAS and CAS Addresses.
- "Virtual Caching" in DRAM Pages.
- 4 Independent DRAM banks.
- Bank Address Allocation And Memory Protection.
- VRAM Support.
- 2 Cycle Access At -30.
- Static Column Mode For Fast Access.
- Configuration via Memory-Mapped Registers.

The term "virtual caching" refers to the ability to have four separate pages of DRAMs selected in static column mode, in the four banks. Each bank of DRAMs has its own RAS and CAS pin, so that, independently, each bank can be held in static column mode with RAS low. The provision of four banks allows the programmer to allocate, for example, a bank to the C stack or local workspaces, a bank to the heap or vector space, a bank to code, and a bank to link buffer space. This effectively localises the sequential flow of memory accesses, which are a mix of these types of accesses, and orders them. This localisation will tend to maximise the usage of static column mode, although the proportion of static column accesses to random accesses will be program dependent.

The user can allocate the banks of external memory space in the same way as for internal memory that is through the configuration registers. Each bank will have a register to hold the address of the top of the bank, and a register to hold the address of the bottom of the bank. The addresses will be selectable down to a word boundary.

As a result of allowing the user to select areas of the address space for the banks, it is necessary to handle the resulting exceptions which can arise. The

two error conditions which can occur are accesses to a non-existent address, and accesses to more than one bank (which can occur as a result of mis-programming the bank address registers). The exception will be handled by interrupting the CPU in the same way as the links and timers interrupt on previous transputers, and using the context-swap mechanism to execute a user-provided exception routine.

Memory protection is naturally implemented by moving the address boundaries of the banks to exclude areas of memory which are to be protected. Any subsequent access to a protected part of memory will invoke the exception handler.

The VRAM support will be provided by having a reserved location in memory in the address space of the memory-mapped configuration registers which will initiate a memory cycle for the bank selected which performs a VRAM shift register access cycle.

The following is a list of pins for the EMI:

mux mode	non-mux mode	number of pins
Data[31:0]	Data[31:0]	32
Address[20:2]	Address[20:2]	19
RAS[3:0]	Address[31,28,25,22]	4
PS[3:0]	Address[30,27,24,21]	4
CAS[3:1]	Address[29,26,23]	3
CAS[0]	CS	1
WE[3:0]	WE[3:0]	4
Refresh	Refresh	1
Wait	Wait	1
BusReq	BusReq	1
BusAck	BusAck	1
EMI Mode	EMI Mode	1
		72

Where

RAS	is RowAddressStrobe
PS	is ProgrammableStrobe
CAS	is ColumnAddressStrobe
CS	is ChipSelect
WE	is WriteEnable

The Interaction Of Internal And External Memory

Guy Harriman
January 24, 1989

Introduction

The T810 can access internal memory while external memory is being used, in order to increase the total bandwidth to the memory seen in executing typical, large, application programs.

Operation

The T810, like the T800, has a single address bus and single data bus connecting the CPU to the memory. The address of a .memory access is decoded to determine where the access will be made. There are four internal RAMs, and four external RAM banks.

The types of memory access which can be performed are:

1. Link data access
2. CPU data access (including access on behalf of FPU)
3. Code fetch

These three types of access are evaluated in parallel, but are sequentialised at the coordinator, which prioritised in the order shown above (link data access at highest priority). The coordinator is a part of the CPU control logic which interfaces with the memory.

The T810 coordinator will allow an access of one type to external memory to go ahead, and then permit one of the other independent access types to attempt to use the memory. If this second memory access is to internal memory, then it will be permitted to occur; otherwise it will be marked as pending, and the third access type permitted to attempt to access memory. It will take one cycle to mark an access type as pending, during an external access to another access type.

No overlapped accesses to internal memory will be possible during an external access which has marked both other access types as pending. After the completion of the external access, the highest priority access will follow immediately, without a wasted cycle.

While one access type is using the external memory, the access types which are not marked as pending are allowed to continue accessing internal memory; priority to the memory is arbitrated as above if more than one access type is not marked pending.

External write cycles will appear to take a single cycle, as far as address and data bus utilisation are concerned. This is because the external memory interface (EMI) on the T810 will latch address and write data in the first cycle of the access. The remaining cycles in the external write are available for use by the internal memory.

External read cycles will appear to take two cycles, as far as address and data bus utilisation are concerned. The first cycle is used to pass the address, and the last cycle to pass the read data back. The remaining cycles in the middle of the external read can be used by the internal memory.

The typical speed of external memory in static column mode will be two cycles. Therefore external writes, but not external reads, will free 50% of the memory bandwidth. This will result, in particular, in block moves from internal memory to external memory operating at one word transferred per cycle. This is the same speed as block moves from internal to internal memory.

With any external memory speed slower than two cycles, the extra cycles will be available as bandwidth to internal memory.

IMS T810 Instruction Streamer

Guy Harriman
January 24, 1989

Introduction

The Instruction Streamer on the T810 has a 16 word instruction buffer, with the capability of jumping back within the buffer to be done without flushing the decode and execution pipeline.

Logical Operation

There is a circular buffer of 16 words. There are three pointers into the buffer:

1. FetchedBehind - points to the valid word furthest behind IPtr
2. IPtr - points to the next instruction to be executed
3. FetchedAhead - points to the valid word furthest ahead of IPtr

The instruction streamer is flushed on exiting from the start instruction after reset or analyse, on any jump back which is outside the buffer, and on any jump forward. When the instruction streamer is flushed, FetchedAhead and FetchedBehind are forced to point to the same word as IPtr.

The instruction streamer can fetch words ahead of IPtr until it has fetched up to 2 words. Each word fetched causes the FetchedAhead pointer to be incremented. FetchedBehind is incremented when (FetchedAhead - FetchedBehind) = 15 and a word in the buffer is emptied.

As words of instructions are executed, they are left in the buffer. A previously executed word is reused when the FetchedBehind pointer is incremented beyond it; this is equivalent to the front of the instruction buffer wrapping around to that buffer location again. If a jump back is performed which lies within the buffer, then the FetchedAhead pointer may be more than two words ahead. The buffer is described as 'full' if the FetchedAhead pointer is equal to, or greater than, two words ahead of IPtr.

A code fetch can be made in the following cases:

1. The buffer is not full in this cycle and not about to become full as a result of a previous code fetch completing.

2. The buffer was full in the previous cycle, and one word has become empty.

3. The buffer has been flushed in this cycle.

Summary

The T810 instruction streamer allows loops of up to 63 bytes of instructions to be executed, without any code fetches being made after the first time through the loop.

The benefit of the instruction buffer is that loops can be executed without flushing the instruction decode and execution pipeline, on backward jumps within the buffer. The reduction in the number of code fetches made, due to loops of this type, results in more memory bandwidth becoming available for data and link accesses. The effect of increasing memory bandwidth is most noticeable with slow external memory.

IMS T810 Instruction Timings

Jon Beecroft & Guy Harriman
January 23, 1989

Introduction

The following document describes the timing for the important transputer instructions. Hardware will be added to the T810 CPU to execute certain dynamically important instructions in hardware, in parallel with instructions executed in the microcode controlled datapath. This document assumes a good understanding of the current transputer instruction set.

Primary Instructions

Unlike the design of previous transputers, in which the timings of the primary instructions are constant, the timings of the primary instructions of the T810 may depend on the type and timing of the previous instruction. The hardware decoder in the instruction streamer looks at two instructions simultaneously. Some of the primaries can be executed completely in parallel with the final cycle of the previous instruction. For example, the first cycle of the *ldl* instruction can be executed in parallel with the final cycle of the previous instruction.

The term reduced cycle instruction refers to an instruction which is executed in one cycle fewer than the same normal cycle instruction. A normal cycle count instruction is executed by the microcode. A reduced cycle count instruction has its first cycle executed by the instruction streamer control logic, in parallel with the last cycle of the previous instruction.

There are three reasons why an instruction may not be executed in parallel (and therefore speeded up by one cycle) when it is detected by the hardware decoder of the instruction streamer. These are:

- A - The previous instruction's cycle count was reduced to zero.
- B - The previous instruction inhibits a reduced cycle instruction.
- C - The previous instruction requests memory in its final cycle of execution.

These restrictions do not apply to all the reduced cycle instructions.

The table below lists the cycle count of all the primary instructions for reduced and normal execution times in columns 1 and 2. If none of the restrictions in column 3 are generated by the previous instruction, and the instruction can be reduced (as shown in column 1), then that instruction will be executed as a reduced cycle count instruction. Otherwise the instruction will be executed as a normal cycle count instruction. Column 4 lists the restrictions which will be imposed by the current instruction on the next instruction, if the current instruction is executed as a reduced cycle count instruction. Column 5 lists the restrictions which will be imposed by the current instruction on the next instruction, if the current instruction is executed as a normal cycle count instruction.

Primary Instruction Name	Reduced Cycle Count	Normal Cycle Count	Reduced Unless Last Inst Of Type	Next Inst Restrictions - Reduced CycleCount	Next Inst Restrictions - Normal CycleCount
pfix	0	1	A	A	
nfix	0	1	A	A	
ldc	0	1	A, B	A	
ldl	1	2	B, C		
stl	0	1	A, B, C	A, C	C
ldlp	0	1	A, B	A	
adc	N/A	1			
eqc	N/A	1			
j in buffer	N/A	2			
j not in buffer	N/A	3			
cj taken, in buffer	2	3	B		
cj taken, not in buffer	3	4	B		
cj not taken	1	2	B		
ldnl	N/A	2			
stnl	N/A	1			B, C
ldnlp	N/A	1			
call	N/A	5			C
ajw	0	1	A, B	A	B

The jump instructions *j* and *cj* in the table are shown for the cases when the target instruction of the jump is within the instruction buffer. A jump taken on the T810 does not cause the instruction buffer to be flushed if the target instruction is present in the buffer, and therefore does not empty the instruction decode pipeline.

Secondaries

The table below shows the dynamically important secondary instructions that have been speeded up on the T810. Note that all long (prefixed) secondary instructions are usually speeded up by one cycle because of the effect of speeding up the *pfix* instruction.

The instructions which have been speeded up by the addition of extra hardware are listed below. The cycle count of all other instructions is the same as the T800 instruction cycle count, and imposes no restrictions on the next instruction. The instruction streamer reduced cycle restrictions for the next instruction have also been included.

Secondary Instruction Name	Cycle Count	Restrictions On Next Inst
mul (at least one operand +ve)	2	
mul (both operands -ve)	2	B
prod	1	B
shl	2	
shr	2	
gt	1	
lend jump taken	5	C
lend jump not taken	8	
wsub	1	
move	2w+6	C
lb	2	
sb	1	B, C
lhw (load half word)	2	
shw (store half word)	1	B, C
lmul	2	
lshl	2	
lshr	2	

Example Code Sequence With Timings

The following compiled code sequence has been used to give an example comparison of a T800 and a T810 transputer, using internal memory for both code and data.

```
                                T800 Cycles        T810 Cycles
6B BB       -- ajw   -181          2                  1
41          -- ldc   1             1                  1
D0          -- stl   0             1                  0
41          -- ldc   1             1                  1
D1          -- stl   1             1                  0
40          -- ldc   0             1                  1
D2          -- stl   2             1                  0
22 40       -- ldc   32            2                  1
D3          -- stl   3             1                  1
--    L11:
70          -- ldl   0             2                  2
71          -- ldl   1             2                  1
25 F2       -- sum                 2                  1
72          -- ldl   2             2                  1
15          -- ldlp  5             1                  0
FA          -- wsub                2                  1
E0          -- stnl  0             2                  1
70          -- ldl   0             2                  2
41          -- ldc   1             1                  0
24 F1       -- shl                 4                  3
D0          -- stl   0             1                  0
70          -- ldl   0             2                  2
D1          -- stl   1             1                  0
12          -- ldlp  2             1                  1
21 44       -- ldc   20            2                  1
22 F1       -- lend  L11    11 loop / 6 exit   5 loop / 7 exit
                                 ----               ---
                                1222 cycles        680 cycles
```

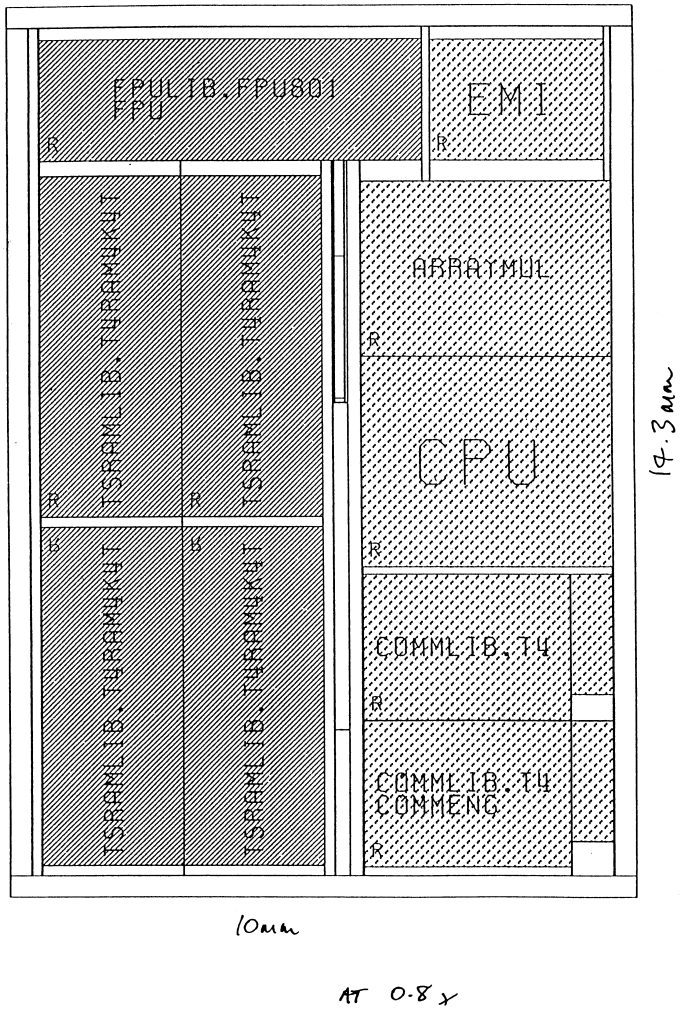

Figure 1: PLOT OF [DESIGN.T4XX.T810A]T810A AT 10X K 26-OCT-88