# CVAX CPU CHIP ENGINEERING SPECIFICATION

DC 341

21-24674-01

Rev. 1.13

Contact: Paul Rubinfeld (SHORTY::RUBINFELD)

C O M P A N Y  C O N F I D E N T I A L

TABLE OF CONTENTS

## REVISION HISTORY

| REV | DATE | REASON |
|---|---|---|
| 1.12 | 12-Apr-85 | Section 2.8.2 SEN and CEN definitions in CADR are swapped.<br>Section 2.8.4 Added SID description.<br>Section 4.1.8.2 Coprocessor return ACB met by setting PSL V.<br>Section 4.4.3 Coprocessor return ACB met by setting PSL V. |
| 1.11 | 3-Apr-85 | Section 4.1.8.2 Added POLY even/odd coefficient flag.<br>Section 4.2.7 Parity can be checked on the interrupt vector.<br>Section 4.4.2 Accelerated integer instruction defined.<br>Section 4.5.* Added test logic description.<br>Section 4.9.4 Defined TEST0L and TEST1 L functionality.<br>Section 4.4.3 Added POLY even/odd coefficient flag.<br>Section 5.2 Deleted Iils parameter.<br>Section 5.3 Added signal summary. |
| 1.10 | 26-Mar-85 | Section 2.63, 4.1.3.7, 4.2.7, 9.1 Update to incorporate VAXB ECO #85Q2-67, SCB vector 0 is assigned to IRQ passive release. |
| 1.09 | 22-Mar-85 | Section 6.2 added Tresetd, Treseth and Tbmh.<br>Section 7.2, 7.8, 7.9, 7.10 updated. |
| 1.08 | 22-Feb-85 | Section 4.1.8.2, 4.2.2, 4.2.3 updated. |
| 1.07 | 19-Feb-85 | Section 4.2.3 DAL parity error response corrected.<br>Section 6.1 Tds & Tdps min and max corrected.<br>Section 6.2 Tdsid, Tdmgsd & Tdmgsid added.<br>Section 7.5 Tdmgsd and Tdmgsid added.<br>Section 7.8, 7.9, 7.10 Tdsid added. |
| 1.06 | 7-Feb-85 | Section 2.5.5 TB description add.<br>Section 2.6.4 machine check codes 2 through 5 clarified.<br>Section 2.8.2 diagnostic mode should only be selected when one set is enabled.<br>Section 4.1.2.2 CS/DP<3> is undefined during non-cacheable references.<br>Section 4.1.7.1, 4.2.9 octaword invalidates added.<br>Section 4.1.9.6 CM L further defined.<br>Section 4.2.2 CCTL L operation added to single transfer description.<br>Section 4.2.3 CCTL L operation added to multiple transfer description. Response table add.<br>Section 4.4.4 Coprocessor reset operation defined.<br>Section 7.3 CM L AC waveforms added.<br>Section 7.4, 7.5 DMA and interrupt timing diagrams separated.<br>Section 7.6, 7.7 quadword and octaword invalidate cycle timing shown.<br>Section 8.0 interconnect diagram fixed up.<br>Section 9.2 item 14 added to highlight IPR number DAL assignment difference between uVAX and CVAX. |
| 1.05 | 24-Jan-85 | Section 1.3 item 7 minimum IO cycle is 200ns.<br>Section 2.8.3 MSER change: <10> indicates hit/miss; <9> memory parity error caused machine check; <8> cache parity error caused machine check<br>Section 4.2.2.2 CS encoding has been changed. LLL and LHL have switch meanings. LHH when WR L is asserted is reserved for DMA device use.<br>Section 4.1.2.3 External resistor now needed for DPE L.<br>Section 4.1.3.4, 4.1.3.5 Control of DS L and DBE L is relinquished when DMG L is asserted. |

| | | |
|---|---|---|
| | | Section 4.1.3.7, 4.1.6.2, 4.2.4, and 4.2.8 DMA is arbitrated after every IO cycle and retry response, irrespective of read lock. |
| | | Section 4.1.4.2, 4.1.5 Defined synchronizer operation for all internally synchronized signals. |
| | | Section 4.1.5.5 Memory Error interrupt added. |
| | | Section 4.1.7.2 Note typo corrected to describe CCTL L. |
| | | Section 4.1.8.2 Added stop coprocessor operation and last data transfer indication to CPDAT encoding. |
| | | Section 4.3.9 the nominal length of an DMA invalidate cycle is 300ns. |
| | | Section 4.4 Added I-stream Prefetching description. |
| | | Section 5.1 Absolute maximum ratings changed to tbs. |
| | | Section 5.2 Icc changed to tbs. |
| | | Section 6.1 Tclkr and Tclkf deleted. Tclke added. Tclkh and Tclkl changed to a min of 5ns, max of 25ns. Tdps added. |
| | | Section 6.2 Tparityh changed to 0ns. Tdsd added. |
| | | Section 7.1 Tclkr and Tclkf have been replaced by Tclke. |
| | | Section 7.5, 7.6, 7.6 DS L now deasserts off of P1. Tsd on DS L has been changed to Tdsd. CS L, DP L, and DPE L signals separated. |
| | | Section 9.2 item 8 interrupt grant levels for CVAX and uVAX differ. Items 13 and 14 added. |
| 1.04 | 2-Jan-85 | Section 4.1.2.3, 4.1.2.3, and 4.2.7 modified to indicate that DAL parity checking is not done during an external processor register read, or interrupt acknowledge cycle. Section 4.2.7 modified to indicate that DAL<31:7,1:0> are driven with zeros when the interrupt level is sent out. |
| 1.03 | 17-Dec-84 | Section 9 filled in. |
| 1.02 | 14-Dec-84 | General clean up. DMA invalidate cycles and timing added. Many inconsistencies and typos corrected. Section 9 added. |
| 1.01 | 11-Dec-84 | General clean up. Added that DMG L will never be asserted if read lock is pending. |
| 1.00 | 11-Nov-84 | Preliminary version. |

# 1.0 INTRODUCTION

## 1.1 Scope

This document describes the CVAX CPU chip, a CMOS/VLSI chip that implements a MicroVAX central processor. This specification describes the external interface and behavior of the chip. It does not describe the internal organization or operation of the chip. For further information, the applicable documents should be consulted.

## 1.2 Applicable Documents

VAX Architecture Standard (DEC Standard 032)
CVAX CPU Chip Design Specification

## 1.3 CVAX CPU Chip Features

The CVAX CPU chip is a 32-bit, virtual memory microprocessor. Implemented in CMOS I (double metal CMOS), the CVAX CPU chip is a high performance, low cost CPU for single board computers, single user workstations, low end systems, and other applications that do not need the flexibility of, or cannot afford the complexity of, the full VAX architecture. Its key features are:

1. Subset VAX data types. The CVAX CPU chip supports the following subset of the VAX data types: byte, word, longword, quadword, character string, and variable length bit field. Support for f_floating, d_floating, and g_floating data types is provided by an external Floating Point Coprocessor. Support for the remaining VAX data types can be provided by macrocode emulation.

2. Subset VAX instruction set. The CVAX CPU chip implements the following subset of the VAX instruction set: integer and logical, address, variable length bit field, control, procedure call, miscellaneous, queue, MOVC3/MOVC5, and operating system support. The f_floating point, d_floating point, and g_floating point instructions are implemented in an external floating point unit. The remaining VAX instructions can be implemented via macrocode emulation (the CVAX CPU chip provides microcode assists for the emulation of the character string, decimal string, EDITPC, and CRC instructions).

3. Full VAX memory management. The CVAX CPU chip includes a demand paged memory management unit which is fully compatible with VAX memory management. System space addresses are virtually mapped through single level page tables, process space addresses through double level page tables.

4. On chip cache. The CVAX CPU chip includes 1k bytes of on chip cache to optimize the performance of the memory subsystem. The cache can be configured to store I-stream only references, or both I-stream and D-stream references.

5. External interface based on industry Standards. The CVAX CPU chip's external interface is a 32-bit extension of the defacto industry standard microprocessor interface. The CVAX chip functionally replaces the MicroVAX chip in existing designs although the exact timing, pin assignments, and external protocol have slightly changed. Chapter 9 highlights the interface differences between MicroVAX and CVAX.

6. Large virtual and physical address space. The CVAX CPU chip supports four gigabytes (2**32) of virtual memory, and one gigabyte (2**30) of physical memory.

7. High performance. At its maximum frequency, the CVAX CPU chip achieves a 100 nsec microcycle and a 200 nsec I/O cycle.

8. Single package. The CVAX CPU chip is packaged in a standard 84-pin surface mounted chip carrier.

## 1.4 Summary Of Differences

The principal differences between the CVAX CPU chip and the full VAX architecture are these:

1. The CVAX CPU chip omits these data types: decimal string, octaword, h_floating. [These data types can be supported via macrocode emulation.]

2. The CVAX CPU chip omits these instruction classes: character string (except for MOVC3/MOVC5), decimal string, EDITPC, CRC, compatability mode, octaword, h_floating. [The chip provides microcode assists for the macrocode emulation of the character string, decimal string, CRC, and EDITPC instructions.]

3. The CVAX CPU chip omits some of the required full VAX internal processor registers. Specifically, NICR, ICR, TODR, RXCS, RXDB, TXCS, TXDB and PMR are not built in CVAX. MFPR or MTPR references to these registers generates an external processor register read or write cycle, respectively. Therefore, these registers can externally supported.

4. The CVAX CPU chip does not have a built-in console function.

## 2.0 ARCHITECTURE SUMMARY

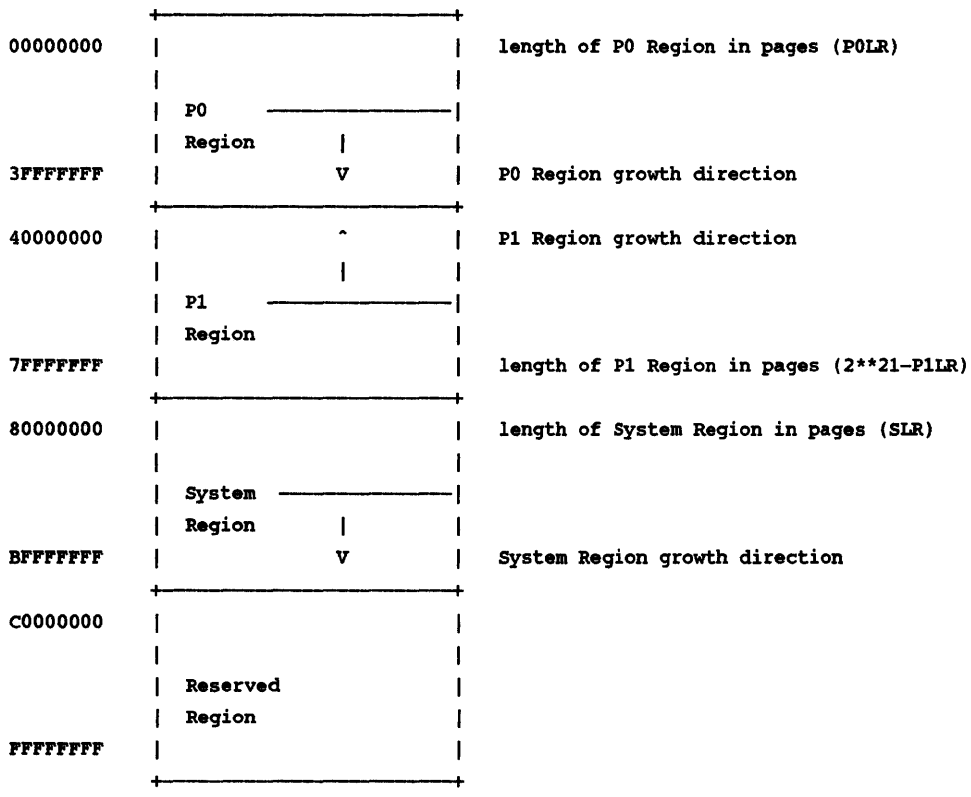
This section provides summary information about the architecture of the CVAX CPU chip. It is not intended as a complete reference but rather to give an overview of the user-visible features. For a complete description of the architecture, consult the "VAX Architecture Standard".

### 2.1 Visible State

The visible state of the processor consists of memory, both virtual and physical, the general registers, and the processor status longword (PSL).

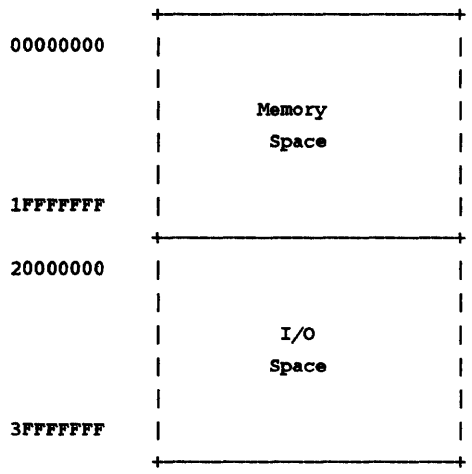
#### 2.1.1 Virtual Address Space -

The virtual address space is four gigabytes (2**32), as follows:

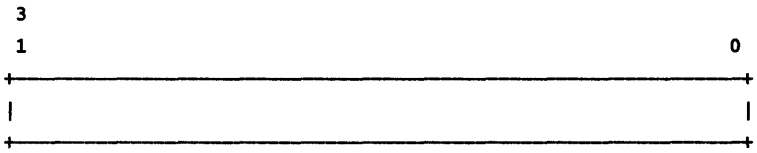


### 2.1.2 Physical Address Space -

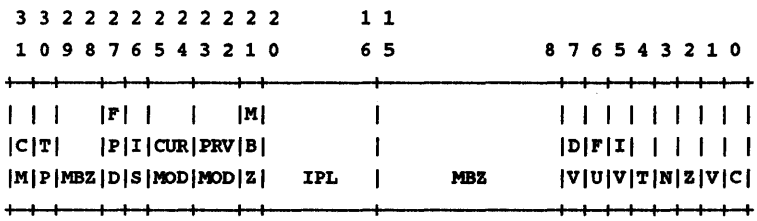
The physical address space is one gigabyte (2**30), as follows:



### 2.1.3 Registers -



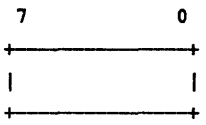
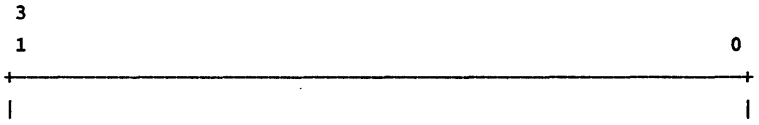
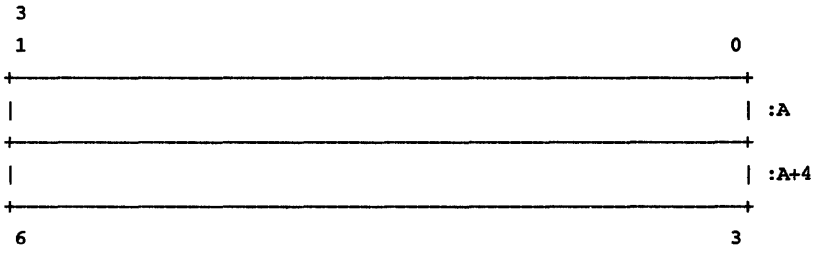
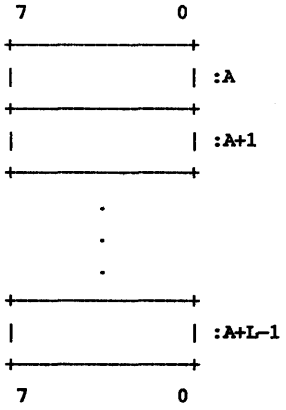

16 general registers
[R0 - R11, general purpose
R12 = AP, argument pointer
R13 = FP, frame pointer
R14 = SP, stack pointer
R15 = PC, program counter]

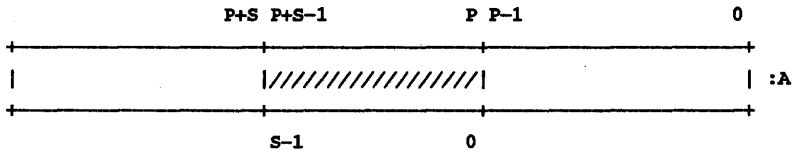
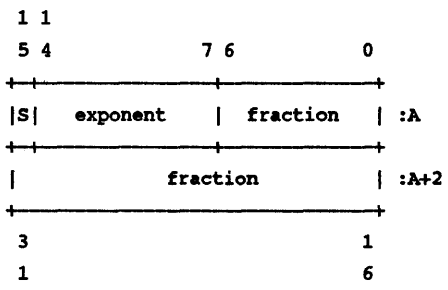
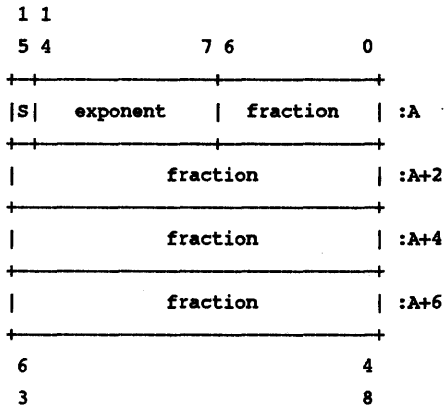
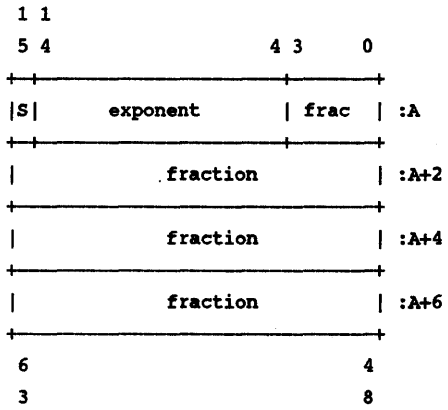



Processor Status Longword
[CM = compatability mode
TP = trace pending
FPD = first part done
IS = interrupt stack
CUR = current mode
PRV = previous mode
IPL = interrupt priority level
DV = decimal overflow trap enable
FU = floating underflow fault enable
IV = integer overflow trap enable
T = trace trap enable
N = negative condition code
Z = zero condition code
V = overflow condition code
C = carry condition code]

## 2.2 Data Types

The CVAX architecture supports nine data types: byte, word, longword, quadword, character string, variable length bit field, and, through an optional external processor, f_floating, d_floating, and g_floating. These are summarized below:

| Type | Length | Use | Graphical Representation |
|---|---|---|---|
| Byte | 8 bits | signed or unsigned integer |  |
| Word | 16 bits | signed or unsigned integer |  |
| Longword | 32 bits | signed or unsigned integer |  |
| Quadword | 64 bits | signed integer |  |
| Character String | 0-65k bytes | byte string |  |

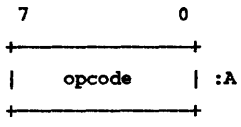
| | | | |
|---|---|---|---|
| Variable Length Bit Field | 0–32 bits | bit string |  |
| F_floating | 32 bits | floating point |  |
| D_floating | 64 bits | floating point |  |
| G_floating | 64 bits | floating point |  |

## 2.3 Instruction Formats And Addressing Modes

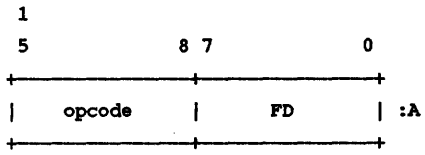
CVAX instructions consist of a one- or two-byte opcode, followed by zero to six operand specifiers.

### 2.3.1 Opcode Formats -

One byte opcode:

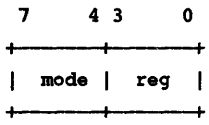



Two byte opcode:



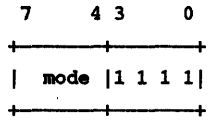

### 2.3.2 General Register Operand Specifiers -

The general register address modes are:




| Mode | Name | Assembler | Access r m w a v | PC | SP | Indexable? |
|---|---|---|---|---|---|---|
| 0-3 | literal | S^#literal | y f f f f | - | - | f |
| 4 | index | i[Rx] | y y y y y | f | y | f |
| 5 | register | Rn | y y y f y | u | uq | f |
| 6 | register deferred | (Rn) | y y y y y | u | y | y |
| 7 | autodecrement | -(Rn) | y y y y y | u | y | ux |
| 8 | autoincrement | (Rn)+ | y y y y y | p | y | ux |
| 9 | autoincrement deferred | @(Rn)+ | y y y y y | p | y | ux |
| A | byte displacement | B^d(Rn) | y y y y y | p | y | y |
| B | byte displacement deferred | @B^d(Rn) | y y y y y | p | y | y |
| C | word displacement | W^d(Rn) | y y y y y | p | y | y |
| D | word displacement deferred | @W^d(Rn) | y y y y y | p | y | y |
| E | longword displacement | L^d(Rn) | y y y y y | p | y | y |
| F | longword displacement deferred | @L^d(Rn) | y y y y y | p | y | y |

If the register is the PC, the address modes are:



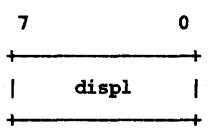

| Mode | Name | Assembler | Access r m w a v | Indexable? |
|---|---|---|---|---|
| 8 | Immediate | I^#constant | y u u y y | u |
| 9 | absolute | @#address | y y y y y | y |
| A | byte relative | B^address | y y y y y | y |
| B | byte relative deferred | @B^address | y y y y y | y |
| C | word relative | W^address | y y y y y | y |
| D | word relative deferred | W^address | y y y y y | y |
| E | longword relative | L^address | y y y y y | y |
| F | longword relative deferred | L^address | y y y y y | y |

Addressing Legend

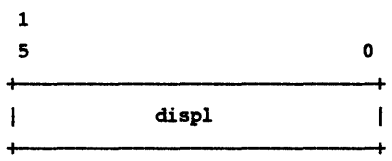
| Access: | | Syntax: | | Results: | |
|---|---|---|---|---|---|
| r = | read | i = | any indexable address mode | y = | yes, always valid address mode |
| m = | modify | d = | displacement | f = | reserved address mode fault |
| w = | write | Rn = | general register, n = 0 to 15 | - = | logically impossible |
| a = | address | Rx = | general register, x = 0 to 14 | p = | program counter addressing |
| v = | field | | | u = | unpredictable |
| | | | | uq = | unpredictable for quad, d/g_floating, and field if pos + size > 32 |
| | | | | ux = | unpredictable if index reg = base reg |

2.3.3 Branch Operand Specifiers -

Signed byte displacement:




Signed word displacement:



## 2.4 Instruction Set

The standard notation for operand specifiers is:

<name>.<access type><data type>

where:

1. Name is a suggestive name for the operand in the context of the instruction. It is the capitalized name of a register or block for implied operands.

2. Access type is a letter denoting the operand specifier access type.

   a = address operand
   b = branch displacement
   m = modified operand (both read and written)
   r = read only operand
   v = if not "Rn", same as a, otherwise R[n+1]'R[n]
   w = write only operand

3. Data type is a letter denoting the data type of the operand.

   b = byte
   d = d_floating
   f = f_floating
   g = g_floating
   l = longword
   q = quadword
   v = field (used only in implied operands)
   w = word
   * = multiple longwords (used only in implied operands)

4. Implied operands, that is, locations that are accessed by the instruction, but not specified in an operand, are denoted by curly braces {}.

The abbreviations for condition codes are:

   * = conditionally set/cleared
   - = not affected
   0 = cleared
   1 = set

The abbreviations for exceptions are:

   rsv = reserved operand fault
   iov = integer overflow trap
   idvz = integer divide by zero trap
   fov = floating overflow fault
   fuv = floating underflow fault
   fdvz = floating divide by zero fault
   dov = decimal overflow trap

ddvz = decimal divide by zero trap
sub = subscript range trap
prv = privileged instruction fault

### 2.4.1 Integer Arithmetic And Logical Instructions -

| Opcode | Instruction | N Z V C | Exceptions |
|---|---|---|---|
| 58 | ADAWI add.rw, sum.mw | * * * * | iov |
| 80 | ADDB2 add.rb, sum.mb | * * * * | iov |
| C0 | ADDL2 add.rl, sum.ml | * * * * | iov |
| A0 | ADDW2 add.rw, sum.mw | * * * * | iov |
| 81 | ADDB3 add1.rb, add2.rb, sum.wb | * * * * | iov |
| C1 | ADDL3 add1.rl, add2.rl, sum.wl | * * * * | iov |
| A1 | ADDW3 add1.rw, add2.rw, sum.ww | * * * * | iov |
| D8 | ADWC add.rl, sum.ml | * * * * | iov |
| 78 | ASHL cnt.rb, src.rl, dst.wl | * * * 0 | iov |
| 79 | ASHQ cnt.rb, src.rq, dst.wq | * * * 0 | iov |
| 8A | BICB2 mask.rb, dst.mb | * * 0 - | |
| CA | BICL2 mask.rl, dst.ml | * * 0 - | |
| AA | BICW2 mask.rw, dst.mw | * * 0 - | |
| 8B | BICB3 mask.rb, src.rb, dst.wb | * * 0 - | |
| CB | BICL3 mask.rl, src.rl, dst.wl | * * 0 - | |
| AB | BICW3 mask.rw, src.rw, dst.ww | * * 0 - | |
| 88 | BISB2 mask.rb, dst.mb | * * 0 - | |
| C8 | BISL2 mask.rl, dst.ml | * * 0 - | |
| A8 | BISW2 mask.rw, dst.mw | * * 0 - | |
| 89 | BISB3 mask.rb, src.rb, dst.wb | * * 0 - | |
| C9 | BISL3 mask.rl, src.rl, dst.wl | * * 0 - | |
| A9 | BISW3 mask.rw, src.rw, dst.ww | * * 0 - | |
| 93 | BITB mask.rb, src.rb | * * 0 - | |
| D3 | BITL mask.rl, src.rl | * * 0 - | |
| B3 | BITW mask.rw, src.rw | * * 0 - | |
| 94 | CLRB dst.wb | 0 1 0 - | |
| D4 | CLRL{=F} dst.wl | 0 1 0 - | |
| 7C | CLRQ{=D=G} dst.wq | 0 1 0 - | |
| B4 | CLRW dst.ww | 0 1 0 - | |
| 91 | CMPB src1.rb, src2.rb | * * 0 * | |
| D1 | CMPL src1.rl, src2.rl | * * 0 * | |

| | | | |
|---|---|---|---|
| B1 | CMPW src1.rw, src2.rw | * * 0 * | |
| 98 | CVTBL src.rb, dst.wl | * * 0 0 | |
| 99 | CVTBW src.rb, dst.wl | * * 0 0 | |
| F6 | CVTLB src.rl, dst.wb | * * * 0 | iov |
| F7 | CVTLW src.rl, dst.ww | * * * 0 | iov |
| 33 | CVTWB src.rw, dst.wb | * * * 0 | iov |
| 32 | CVTWL src.rw, dst.wl | * * 0 0 | |
| 97 | DECB dif.mb | * * * * | iov |
| D7 | DECL dif.ml | * * * * | iov |
| B7 | DECW dif.mw | * * * * | iov |
| 86 | DIVB2 divr.rb, quo.mb | * * * 0 | iov, idvz |
| C6 | DIVL2 divr.rl, quo.ml | * * * 0 | iov, idvz |
| A6 | DIVW2 divr.rw, quo.mw | * * * 0 | iov, idvz |
| 87 | DIVB3 divr.rb, divd.rb, quo.wb | * * * 0 | iov, idvz |
| C7 | DIVL3 divr.rl, divd.rl, quo.wl | * * * 0 | iov, idvz |
| A7 | DIVW3 divr.rw, divd.rw, quo.ww | * * * 0 | iov, idvz |
| 7B | EDIV divr.rl, divd.rq, quo.wl, rem.wl | * * * 0 | iov, idvz |
| 7A | EMUL mulr.rl, muld.rl, add.rl, prod.wq | * * 0 0 | |
| 96 | INCB sum.mb | * * * * | iov |
| D6 | INCL sum.ml | * * * * | iov |
| B6 | INCW sum.mw | * * * * | iov |
| 92 | MCOMB src.rb, dst.wb | * * 0 - | |
| D2 | MCOML src.rl, dst.wl | * * 0 - | |
| B2 | MCOMW src.rw, dst.ww | * * 0 - | |
| 8E | MNEGB src.rb, dst.wb | * * * * | iov |
| CE | MNEGL src.rl, dst.wl | * * * * | iov |
| AE | MNEGW src.rw, dst.ww | * * * * | iov |
| 90 | MOVB src.rb, dst.wb | * * 0 - | |
| D0 | MOVL src.rl, dst.wl | * * 0 - | |
| 7D | MOVQ src.rq, dst.wq | * * 0 - | |
| B0 | MOVW src.rw, dst.ww | * * 0 - | |
| 9A | MOVZBW src.rb, dst.wb | 0 * 0 - | |
| 9B | MOVZBL src.rb, dst.wl | 0 * 0 - | |
| 3C | MOVZWL src.rw, dst.ww | 0 * 0 - | |
| 84 | MULB2 mulr.rb, prod.mb | * * * 0 | iov |
| C4 | MULL2 mulr.rl, prod.ml | * * * 0 | iov |
| A4 | MULW2 mulr.rw, prod.mw | * * * 0 | iov |
| 85 | MULB3 mulr.rb, muld.rb, prod.wb | * * * 0 | iov |
| C5 | MULL3 mulr.rl, muld.rl, prod.wl | * * * 0 | iov |
| A5 | MULW3 mulr.rw, muld.rw, prod.ww | * * * 0 | iov |

| | | | |
|---|---|---|---|
| DD | PUSHL src.rl, {-(SP).wl} | * * 0 - | |
| 9C | ROTL cnt.rb, src.rl, dst.wl | * * 0 - | |
| D9 | SBWC sub.rl, dif.ml | * * * * | iov |
| 82 | SUBB2 sub.rb, dif.mb | * * * * | iov |
| C2 | SUBL2 sub.rl, dif.ml | * * * * | iov |
| A2 | SUBW2 sub.rw, dif.mw | * * * * | iov |
| 83 | SUBB3 sub.rb, min.rb, dif.wb | * * * * | iov |
| C3 | SUBL3 sub.rl, min.rl, dif.wl | * * * * | iov |
| A3 | SUBW3 sub.rw, min.rw, dif.ww | * * * * | iov |
| 95 | TSTB src.rb | * * 0 0 | |
| D5 | TSTL src.rl | * * 0 0 | |
| B5 | TSTW src.rw | * * 0 0 | |
| 8C | XORB2 mask.rb, dst.mb | * * 0 - | |
| CC | XORL2 mask.rl, dst.ml | * * 0 - | |
| AC | XORW2 mask.rw, dst.mw | * * 0 - | |
| 8D | XORB3 mask.rb, src.rb, dst.wb | * * 0 - | |
| CD | XORL3 mask.rl, src.rl, dst.wl | * * 0 - | |
| AD | XORW3 mask.rw, src.rw, dst.ww | * * 0 - | |

### 2.4.2 Address Instructions -

| Opcode | Instruction | N Z V C | Exceptions |
|---|---|---|---|
| 9E | MOVAB src.ab, dst.wl | * * 0 - | |
| DE | MOVAL{=F} src.al, dst.wl | * * 0 - | |
| 7E | MOVAQ{=D=G} src.aq, dst.wl | * * 0 - | |
| 3E | MOVAW src.aw, dst.wl | * * 0 - | |
| 9F | PUSHAB src.ab, {-(SP).wl} | * * 0 - | |
| DF | PUSHAL{=F} src.al, {-(SP).wl} | * * 0 - | |
| 7F | PUSHAQ{=D=G} src.aq, {-(SP).wl} | * * 0 - | |
| 3F | PUSHAW src.aw, {-(SP).wl} | * * 0 - | |

### 2.4.3 Variable Length Bit Field Instructions -

| Opcode | Instruction | N Z V C | Exceptions |
|---|---|---|---|
| EC | CMPV pos.rl, size.rb, base.vb, {field.rv}, src.rl | * * 0 * | rsv |

| | | | |
|---|---|---|---|
| ED | CMPZV pos.rl, size.rb, base.vb, {field.rv}, src.rl | * * 0 * | rsv |
| EE | EXTV pos.rl, size.rb, base.vb, {field.rv}, dst.wl | * * 0 - | rsv |
| EF | EXTZV pos.rl, size.rb, base.vb, {field.rv}, dst.wl | * * 0 - | rsv |
| F0 | INSV src.rl, pos.rl, size.rb, base.vb, {field.wv} | - - - - | rsv |
| EB | FFC startpos.rl, size.rb, base.vb, {field.rv}, findpos.wl | 0 * 0 0 | rsv |
| EA | FFS startpos.rl, size.rb, base.vb, {field.rv}, findpos.wl | 0 * 0 0 | rsv |

### 2.4.4 Control Instructions -

| Opcode | Instruction | N Z V C | Exceptions |
|---|---|---|---|
| 9D | ACBB limit.rb, add.rb, index.mb, displ.bw | * * * - | iov |
| F1 | ACBL limit.rl, add.rl, index.ml, displ.bw | * * * - | iov |
| 3D | ACBW limit.rw, add.rw, index.mw, displ.bw | * * * - | iov |
| F3 | AOBLEQ limit.rl, index.ml, displ.bb | * * * - | iov |
| F2 | AOBLSS limit.rl, index.ml, displ.bb | * * * - | iov |
| 1E | BCC{=BGEQU} displ.bb | - - - - | |
| 1F | BCS{=BLSSU} displ.bb | - - - - | |
| 13 | BEQL{=BEQLU} displ.bb | - - - - | |
| 18 | BGEQ displ.bb | - - - - | |
| 14 | BGTR displ.bb | - - - - | |
| 1A | BGTRU displ.bb | - - - - | |
| 15 | BLEQ displ.bb | - - - - | |
| 1B | BLEQU displ.bb | - - - - | |
| 19 | BLSS displ.bb | - - - - | |
| 12 | BNEQ{=BNEQU} displ.bb | - - - - | |
| 1C | BVC displ.bb | - - - - | |
| 1D | BVS displ.bb | - - - - | |
| E1 | BBC pos.rl, base.vb, displ.bb, {field.rv} | - - - - | rsv |
| E0 | BBS pos.rl, base.vb, displ.bb, {field.rv} | - - - - | rsv |
| E5 | BBCC pos.rl, base.vb, displ.bb, {field.mv} | - - - - | rsv |
| E3 | BBCS pos.rl, base.vb, displ.bb, {field.mv} | - - - - | rsv |
| E4 | BBSC pos.rl, base.vb, displ.bb, {field.mv} | - - - - | rsv |
| E2 | BBSS pos.rl, base.vb, displ.bb, {field.mv} | - - - - | rsv |
| E7 | BBCCI pos.rl, base.vb, displ.bb, {field.mv} | - - - - | rsv |
| E6 | BBSSI pos.rl, base.vb, displ.bb, {field.mv} | - - - - | rsv |
| E9 | BLBC src.rl, displ.bb | - - - - | |
| E8 | BLBS src.rl, displ.bb | - - - - | |

| | | | |
|---|---|---|---|
| 11 | BRB displ.bb | - - - - | |
| 31 | BRW displ.bw | - - - - | |
| 10 | BSBB displ.bb, {-(SP).wl} | - - - - | |
| 30 | BSBW displ.bw, {-(SP).wl} | - - - - | |
| 8F | CASEB selector.rb, base.rb, limit.rb, displ.bw-list | * * 0 * | |
| CF | CASEL selector.rl, base.rl, limit.rl, displ.bw-list | * * 0 * | |
| AF | CASEW selector.rw, base.rw, limit.rw, displ.bw-list | * * 0 * | |
| 17 | JMP dst.ab | - - - - | |
| 16 | JSB dst.ab, {-(SP).wl} | - - - - | |
| 05 | RSB {(SP)+.rl} | - - - - | |
| F4 | SOBGEQ index.ml, displ.bb | * * * - | iov |
| F5 | SOBGTR index.ml, displ.bb | * * * - | iov |

### 2.4.5 Procedure Call Instructions -

| Opcode | Instruction | N Z V C | Exceptions |
|---|---|---|---|
| FA | CALLG arglist.ab, dst.ab, {-(SP).w*} | 0 0 0 0 | rsv |
| FB | CALLS numarg.rl, dst.ab, {-(SP).w*} | 0 0 0 0 | rsv |
| 04 | RET {(SP)+.r*} | * * * * | rsv |

### 2.4.6 Miscellaneous Instructions -

| Opcode | Instruction | N Z V C | Exceptions |
|---|---|---|---|
| B9 | BICPSW mask.rw | * * * * | rsv |
| B8 | BISPSW mask.rw | * * * * | rsv |
| 03 | BPT {-(KSP).w*} | 0 0 0 0 | |
| 00 | HALT {-(KSP).w*} | - - - - | prv |
| 0A | INDEX subscript.rl, low.rl, high.rl, size.rl, indexin.rl, indexout.wl | * * 0 0 | sub |
| DC | MOVPSL dst.wl | - - - - | |

| | | |
|---|---|---|
| 01 | NOP | - - - - |
| BA | POPR mask.rw, {(SP)+.r*} | - - - - |
| BB | PUSHR mask.rw, {-(SP).w*} | - - - - |
| FC | XFC {unspecified operands} | 0 0 0 0 |

2.4.7 Queue Instructions -

| Opcode | Instruction | N Z V C | Exceptions |
|---|---|---|---|
| 5C | INSQHI entry.ab, header.aq | 0 * 0 * | rsv |
| 5D | INSQTI entry.ab, header.aq | 0 * 0 * | rsv |
| 0E | INSQUE entry.ab, pred.ab | * * 0 * | |
| 5E | REMQHI header.aq, addr.wl | 0 * * * | rsv |
| 5F | REMQTI header.aq, addr.wl | 0 * * * | rsv |
| 0F | REMQUE entry.ab, addr.wl | * * * * | |

2.4.8 Character String Instructions -

| Opcode | Instruction | N Z V C | Exceptions |
|---|---|---|---|
| 28 | MOVC3 len.rw, srcaddr.ab, dstaddr.ab, {R0-5.wl} | 0 1 0 0 | |
| 2C | MOVC5 srclen.rw, srcaddr.ab, fill.rb, dstlen.rw, dstaddr.ab, {R0-5.wl} | * * 0 * | |

2.4.9 Operating System Support Instructions -

| Opcode | Instruction | N Z V C | Exceptions |
|---|---|---|---|
| BD | CHME param.rw, {-(ySP).w*} | 0 0 0 0 | |
| BC | CHMK param.rw, {-(ySP).w*} | 0 0 0 0 | |
| BE | CHMS param.rw, {-(ySP).w*} | 0 0 0 0 | |
| BF | CHMU param.rw, {-(ySP).w*} | 0 0 0 0 | |

Where y=MINU(x, PSL<current_mode>)

| | | | |
|---|---|---|---|
| 06 | LDPCTX {PCB.r*, -(KSP).w*} | - - - - | rsv, prv |
| DB | MFPR procreg.rl, dst.wl | * * 0 - | rsv, prv |
| DA | MTPR src.rl, procreg.rl | * * 0 - | rsv, prv |
| 0C | PROBER mode.rb, len.rw, base.ab | 0 * 0 - | |
| 0D | PROBEW mode.rb, len.rw, base.ab | 0 * 0 - | |
| 02 | REI {(SP)+.r*} | * * * * | rsv |
| 07 | SVPCTX {(SP)+.r*, PCB.w*} | - - - - | prv |

### 2.4.10 Floating Point Instructions -

These instructions are implemented in hardware only if an external floating point unit is present in the system.

| Opcode | Instruction | N Z V C | Exceptions |
|---|---|---|---|
| 6F | ACBD limit.rd, add.rd, index.md,displ.bw | * * 0 - | rsv, fov, fuv |
| 4F | ACBF limit.rf, add.rf, index.mf,displ.bw | * * 0 - | rsv, fov, fuv |
| 4FFD | ACBG limit.rg, add.rg, index.mg,displ.bw | * * 0 - | rsv, fov, fuv |
| 60 | ADDD2 add.rd, sum.md | * * 0 0 | rsv, fov, fuv |
| 40 | ADDF2 add.rf, sum.mf | * * 0 0 | rsv, fov, fuv |
| 40FD | ADDG2 add.rg, sum.mg | * * 0 0 | rsv, fov, fuv |
| 61 | ADDD3 add1.rd, add2.rd, sum.wd | * * 0 0 | rsv, fov, fuv |
| 41 | ADDF3 add1.rf, add2.rf, sum.wf | * * 0 0 | rsv, fov, fuv |
| 41FD | ADDG3 add1.rg, add2.rg, sum.wg | * * 0 0 | rsv, fov, fuv |
| 71 | CMPD src1.rd, src2.rd | * * 0 0 | rsv |
| 51 | CMPF src1.rf, src2.rf | * * 0 0 | rsv |
| 51FD | CMPG src1.rg, src2.rg | * * 0 0 | rsv |
| 6C | CVTBD src.rb, dst.wd | * * 0 0 | |
| 4C | CVTBF src.rb, dst.wf | * * 0 0 | |
| 4CFD | CVTBG src.rb, dst.wg | * * 0 0 | |
| 68 | CVTDB src.rd, dst.wb | * * * 0 | rsv, iov |
| 76 | CVTDF src.rd, dst.wf | * * 0 0 | rsv, fov |
| 6A | CVTDL src.rd, dst.wl | * * * 0 | rsv, iov |
| 69 | CVTDW src.rd, dst.ww | * * * 0 | rsv, iov |
| 48 | CVTFB src.rf, dst.wb | * * * 0 | rsv, iov |
| 56 | CVTFD src.rf, dst.wd | * * 0 0 | rsv |
| 99FD | CVTFG src.rf, dst.wg | * * 0 0 | rsv |
| 4A | CVTFL src.rf, dst.wl | * * * 0 | rsv, iov |
| 49 | CVTFW src.rf, dst.ww | * * * 0 | rsv, iov |
| 48FD | CVTGB src.rg, dst.wb | * * * 0 | rsv, iov |

| | | | |
|---|---|---|---|
| 33FD | CVTGF src.rg, dst.wf | * * 0 0 | rsv, fov, fuv |
| 4AFD | CVTGL src.rg, dst.wl | * * * 0 | rsv, iov |
| 49FD | CVTGW src.rg, dst.ww | * * * 0 | rsv, iov |
| 6E | CVTLD src.rl, dst.wd | * * 0 0 | |
| 4E | CVTLF src.rl, dst.wf | * * 0 0 | |
| 4EFD | CVTLG src.rl, dst.wg | * * 0 0 | |
| 6D | CVTWD src.rw, dst.wd | * * 0 0 | |
| 4D | CVTWF src.rw, dst.wf | * * 0 0 | |
| 4DFD | CVTWG src.rw, dst.wg | * * 0 0 | |
| 6B | CVTRDL src.rd, dst.wl | * * * 0 | rsv, iov |
| 4B | CVTRFL src.rf, dst.wl | * * * 0 | rsv, iov |
| 4BFD | CVTRGL src.rg, dst.wl | * * * 0 | rsv, iov |
| 66 | DIVD2 divr.rd, quo.md | * * 0 0 | rsv, fov, fuv, fdvz |
| 46 | DIVF2 divr.rf, quo.mf | * * 0 0 | rsv, fov, fuv, fdvz |
| 46FD | DIVG2 divr.rg, quo.mg | * * 0 0 | rsv, fov, fuv, fdvz |
| 67 | DIVD3 divr.rd, divd.rd, quo.wd | * * 0 0 | rsv, fov, fuv, fdvz |
| 47 | DIVF3 divr.rf, divd.rf, quo.wf | * * 0 0 | rsv, fov, fuv, fdvz |
| 47FD | DIVG3 divr.rg, divd.rg, quo.wg | * * 0 0 | rsv, fov, fuv, fdvz |
| 74 | EMODD mulr.rd, mulrx.rb, muld.rd, int.wl, fract.wd | * * * 0 | rsv, fov, fuv, iov |
| 54 | EMODF mulr.rf, mulrx.rb, muld.rf, int.wl, fract.wf | * * * 0 | rsv, fov, fuv, iov |
| 54FD | EMODG mulr.rg, mulrx.rw, muld.rg, int.wl, fract.wg | * * * 0 | rsv, fov, fuv, iov |
| 72 | MNEGD src.rd, dst.wd | * * 0 0 | rsv |
| 52 | MNEGF src.rf, dst.wf | * * 0 0 | rsv |
| 52FD | MNEGG src.rg, dst.wg | * * 0 0 | rsv |
| 70 | MOVD src.rd, dst.wd | * * 0 – | rsv |
| 50 | MOVF src.rf, dst.wf | * * 0 – | rsv |
| 50FD | MOVG src.rg, dst.wg | * * 0 – | rsv |
| 64 | MULD2 mulr.rd, prod.md | * * 0 0 | rsv, fov, fuv |
| 44 | MULF2 mulr.rf, prod.mf | * * 0 0 | rsv, fov, fuv |
| 44FD | MULG2 mulr.rg, prod.mg | * * 0 0 | rsv, fov, fuv |
| 65 | MULD3 mulr.rd, muld.rd, prod.wd | * * 0 0 | rsv, fov, fuv |
| 45 | MULF3 mulr.rf, muld.rf, prod.wf | * * 0 0 | rsv, fov, fuv |
| 45FD | MULG3 mulr.rg, muld.rg, prod.wg | * * 0 0 | rsv, fov, fuv |
| 75 | POLYD arg.rd, degree.rw, table.ab | * * 0 0 | rsv, fov, fuv |
| 55 | POLYF arg.rf, degree.rw, table.ab | * * 0 0 | rsv, fov, fuv |
| 55FD | POLYG arg.rf, degree.rw, table.ab | * * 0 0 | rsv, fov, fuv |
| 62 | SUBD2 sub.rd, dif.md | * * 0 0 | rsv, fov, fuv |
| 42 | SUBF2 sub.rf, dif.mf | * * 0 0 | rsv, fov, fuv |
| 42FD | SUBG2 sub.rg, dif.mg | * * 0 0 | rsv, fov, fuv |
| 63 | SUBD3 sub.rd, min.rd, dif.wd | * * 0 0 | rsv, fov, fuv |
| 43 | SUBF3 sub.rf, min.rf, dif.wf | * * 0 0 | rsv, fov, fuv |
| 43FD | SUBG3 sub.rg, min.rg, dif.wg | * * 0 0 | rsv, fov, fuv |

| | | | |
|---|---|---|---|
| 73 | TSTD src.rd | * * 0 0 | rsv |
| 53 | TSTF src.rf | * * 0 0 | rsv |
| 53FD | TSTG src.rg | * * 0 0 | rsv |

2.4.11 Microcode-Assisted Emulated Instructions -

The chip provides microcode assistance for the macrocode emulation of these instructions. The chip processes the operand specifiers, creates a standard argument list, and invokes an emulation routine to perform emulation.

| Opcode | Instruction | N Z V C | Exceptions |
|---|---|---|---|
| 20 | ADDP4 addlen.rw, addaddr.ab, sumlen.rw, sumaddr.ab | * * * 0 | rsv, dov |
| 21 | ADDP6 add1len.rw, add1addr.ab, add2len.rw, add2addr.ab, sumlen.rw, sumaddr.ab | * * * 0 | rsv, dov |
| F8 | ASHP cnt.rb, srclen.rw, srcaddr.ab, round.rb, dstlen.rw, dstaddr.ab | * * * 0 | rsv, dov |
| 29 | CMPC3 len.rw, src1addr.ab, src2addr.ab | * * 0 * | |
| 2D | CMPC5 src1len.rw, src1addr.ab, fill.rb, src2len.rw, src2addr.ab | * * 0 * | |
| 35 | CMPP3 len.rw, src1addr.ab, src2addr.ab | * * 0 0 | |
| 37 | CMPP4 src1len.rw, src1addr.ab, src2len.rw, src2addr.ab | * * 0 0 | |
| 0B | CRC tbl.ab, inicrc.rl, strlen.rw, stream.ab | * * 0 0 | |
| F9 | CVTLP src.rl, dstlen.rw, dstaddr.ab | * * * 0 | rsv, dov |
| 36 | CVTPL srclen.rw, srcaddr.ab, dst.wl | * * * 0 | rsv, iov |
| 08 | CVTPS srclen.rw, srcaddr.ab, dstlen.rw, dstaddr.ab | * * * 0 | rsv, dov |
| 09 | CVTSP srclen.rw, srcaddr.ab, dstlen.rw, dstaddr.ab | * * * 0 | rsv, dov |
| 24 | CVTPT srclen.rw, srcaddr.ab, tbladdr.ab, dstlen.rw, dstaddr.ab | * * * 0 | rsv, dov |
| 26 | CVTTP srclen.rw, srcaddr.ab, tbladdr.ab, dstlen.rw, dstaddr.ab | * * * 0 | rsv, dov |
| 27 | DIVP divrlen.rw, divraddr.ab, divdlen.rw, divdaddr.ab, quolen.rw, quoaddr.ab | * * * 0 | rsv, dov, ddvz |
| 38 | EDITPC srclen.rw, srcaddr.ab, pattern.ab, dstaddr.ab | * * * * | rsv, dov |
| 3A | LOCC char.rb, len.rw, addr.ab | 0 * 0 0 | |
| 39 | MATCHC objlen.rw, objaddr.ab, srclen.rw, srcaddr.ab | 0 * 0 0 | |

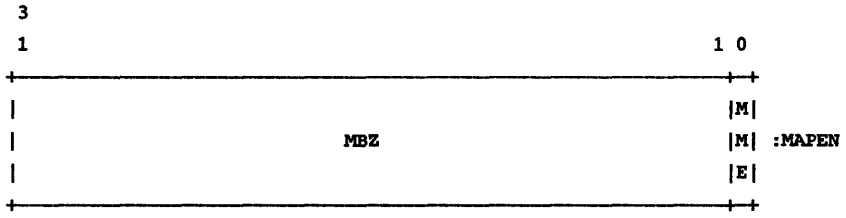
| | | | |
|---|---|---|---|
| 34 | MOVP len.rw, srcaddr.ab, dstaddr.ab | * * 0 0 | |
| 2E | MOVTC srclen.rw, srcaddr.ab, fill.rb, tbladdr.ab, dstlen.rw, dstaddr.ab | * * 0 * | |
| 2F | MOVTUC srclen.rw, srcaddr.ab, esc.rb, tbladdr.ab, dstlen.rw, dstaddr.ab | * * * * | |
| 25 | MULP mulrlen.rw, mulraddr.ab, muldlen.rw, muldaddr.ab, prodlen.rw, prodaddr.ab | * * * 0 | rsv, dov |
| 2A | SCANC len.rw, addr.ab, tbladdr.ab, mask.rb | 0 * 0 0 | |
| 3B | SKPC char.rb, len.rw, addr.ab | 0 * 0 0 | |
| 2B | SPANC len.rw, addr.ab, tbladdr.ab, mask.rb | 0 * 0 0 | |
| 22 | SUBP4 sublen.rw, subaddr.ab, diflen.rw, difaddr.ab | * * * 0 | rsv, dov |
| 23 | SUBP6 sublen.rw, subaddr.ab, minlen.rw, minaddr.ab, diflen.rw, difaddr.ab | * * * 0 | rsv, dov |

## 2.5 Memory Management

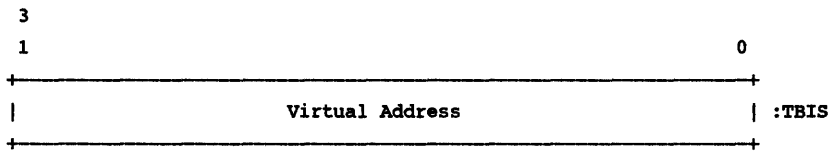
The CVAX architecture provides a four gigabyte (2**32) virtual address space, divided into two sections, system space and process space. Process space is further subdivided into the P0 region and the P1 region.

### 2.5.1 Memory Management Control Registers -

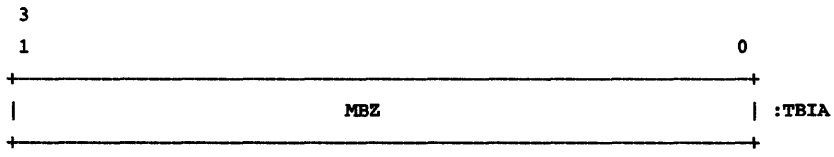
Memory management is controlled by three processor registers: Memory Management Enable (MAPEN), Translation Buffer Invalidate Single (TBIS), and Translation Buffer Invalidate All (TBIA). MAPEN contains one bit: MAPEN<0> = MME enables memory management.




TBIS controls translation buffer invalidation. Writing a virtual address into TBIS invalidates any entry which maps that virtual address.



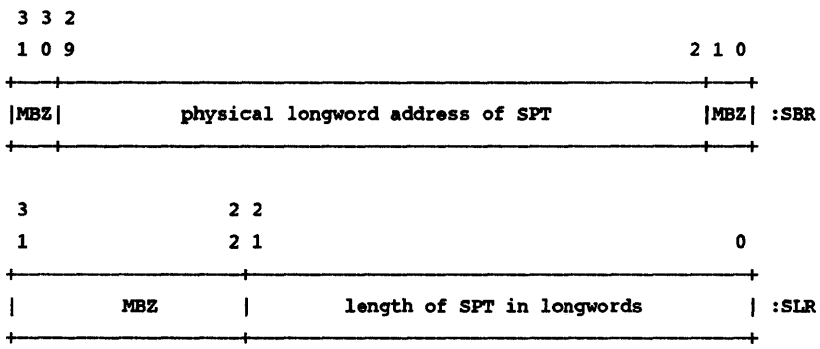

TBIA also controls translation buffer invalidation. Writing a zero into TBIA invalidates the entire translation buffer.

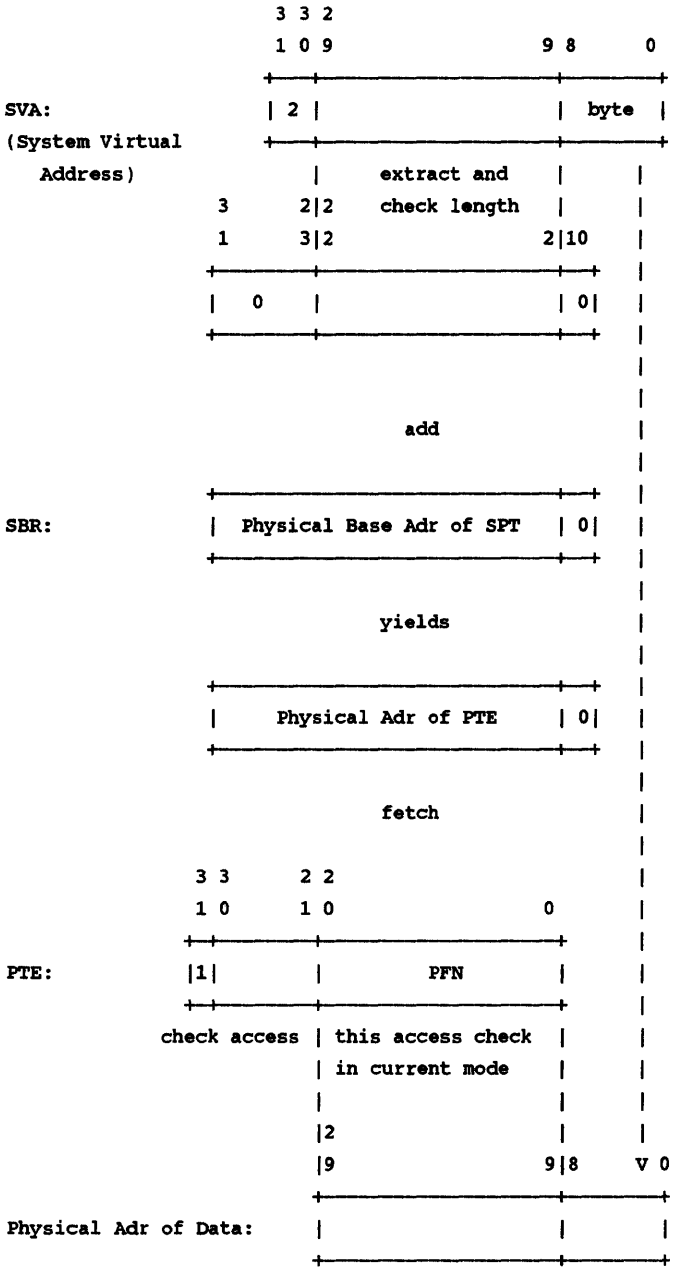


### 2.5.2 System Space Address Translation -

A virtual address with bits <31:30> = 2 is an address in the system virtual address space.

System virtual address space is mapped by the System Page Table (SPT), which is defined by the System Base Register (SBR) and the System Length Register (SLR). The SBR contains the physical address of the the System Page Table. The SLR contains the size of the SPT in longwords, that is, the number of Page Table Entries. The Page Table Entry addressed by the System Base Register maps the first page of system virtual address space, that is, virtual byte address 80000000 (hex).





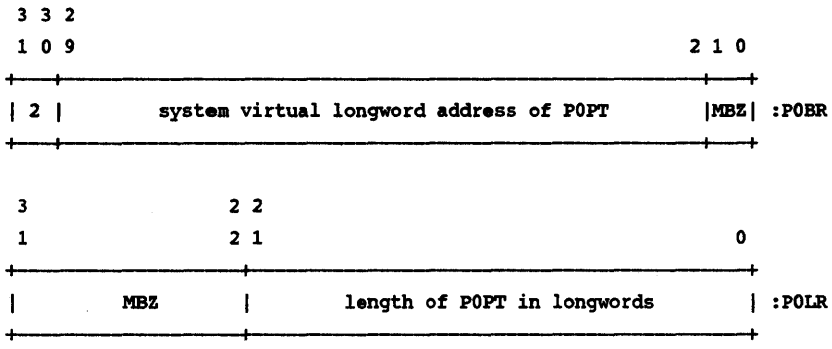

System Virtual to Physical Translation

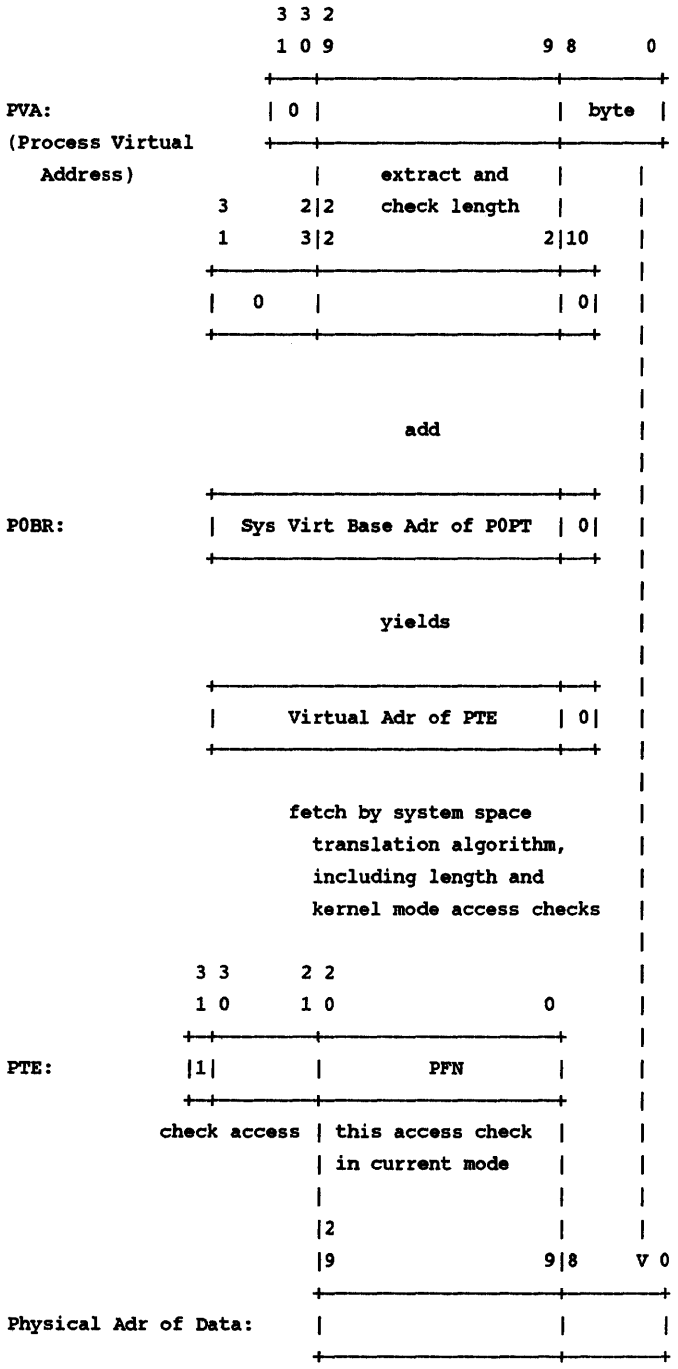
### 2.5.3 Process Space Address Translation -

A virtual address with bit <31> = 0 is an address in the process virtual address space. Process space is divided into two equal sized, separately mapped regions. If virtual address bit <30> = 0, the address is in region P0. If virtual address bit <30> = 1, the address is in region P1.

#### 2.5.3.1 P0 Region Address Translation -

The P0 region of the address space is mapped by the P0 Page Table (P0PT), which is defined by the P0 Base Register (P0BR) and the P0 Length Register (P0LR). The P0BR contains the system virtual address of the P0 Page Table. The P0LR contains the size of the P0PT in longwords, that is, the number of Page Table Entries. The Page Table Entry addressed by the P0 Base Register maps the first page of the P0 region of the virtual address space, that is, virtual byte address 0.

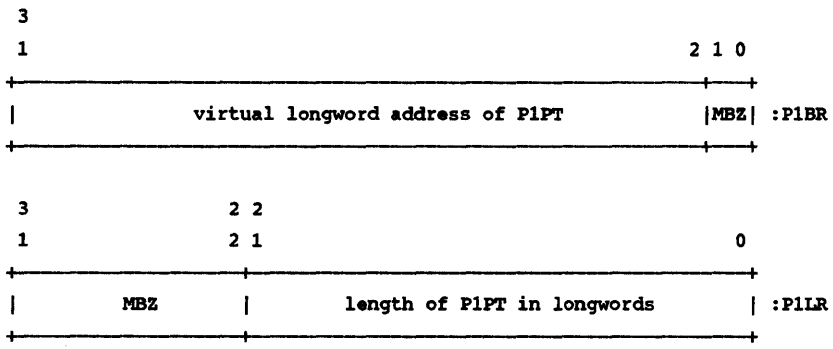


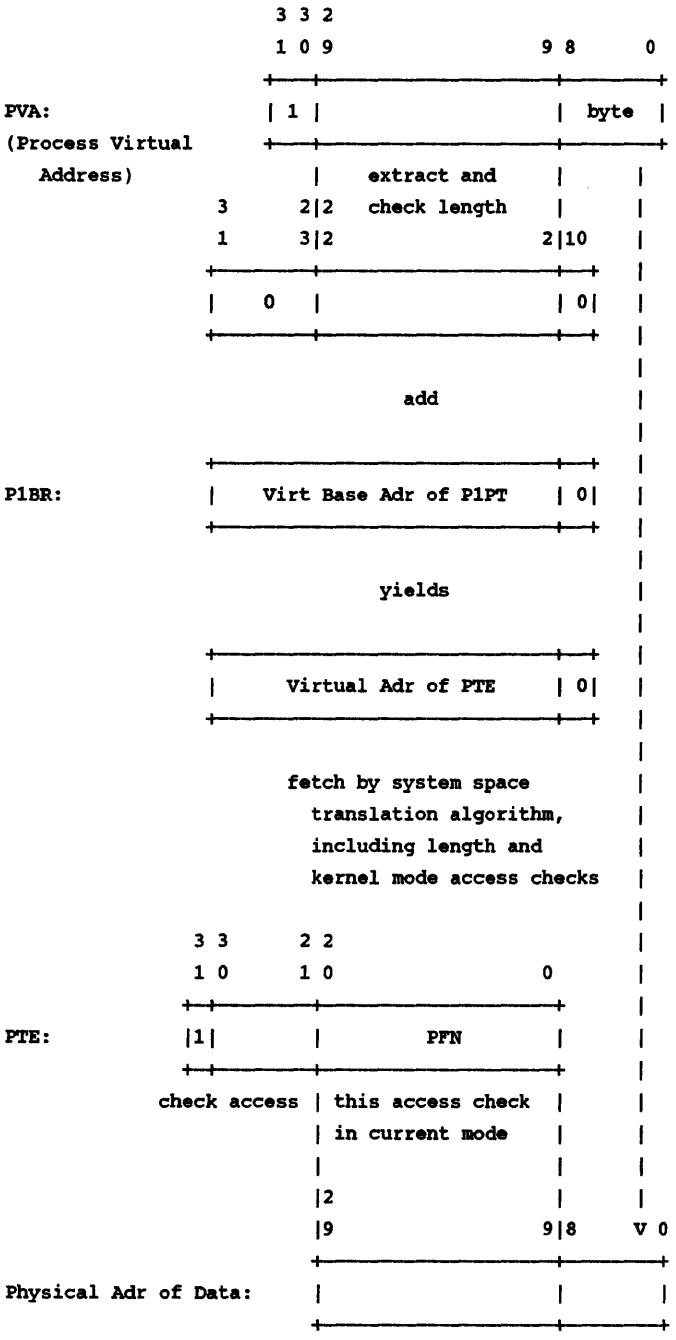



P0 Virtual to Physical Translation

2.5.3.2 P1 Region Address Translation -

The P1 region of the address space is mapped by the P1 Page Table (P1PT), which is defined by the P1 Base Register (P1BR) and the P1 Length Register (P1LR). Because P1 space grows towards smaller addresses, and because a consistent hardware interpretation of the base and length registers is desirable, P1BR and P1LR describe the portion of P1 space that is NOT accessible. Note that P1LR contains the number of nonexistent PTEs. P1BR contains the virtual address of what would be the PTE for the first page of P1, that is, virtual byte address 40000000 (hex). The address in P1BR is not necessarily a valid physical address, but all the addresses of PTEs must be valid physical addresses.

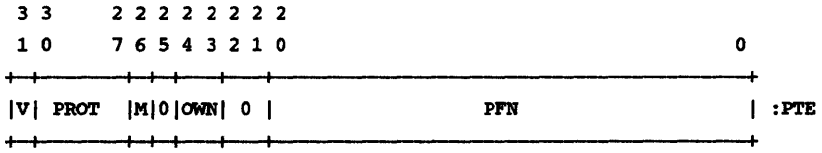





P1 Virtual to Physical Translation

### 2.5.4 Page Table Entry -

The format of a page table entry is:




The protection code access matrix is:

| code | | | current mode | | | | |
|---|---|---|---|---|---|---|---|
| decimal | binary | mnemonic | K | E | S | U | comment |
| 0 | 0000 | NA | - | - | - | - | no access |
| 1 | 0001 | | | unpredictable | | | reserved |
| 2 | 0010 | KW | RW | - | - | - | |
| 3 | 0011 | KR | R | - | - | - | |
| 4 | 0100 | UW | RW | RW | RW | RW | all access |
| 5 | 0101 | EW | RW | RW | - | - | |
| 6 | 0110 | ERKW | RW | R | - | - | |
| 7 | 0111 | ER | R | R | - | - | |
| 8 | 1000 | SW | RW | RW | RW | - | |
| 9 | 1001 | SREW | RW | RW | R | - | |
| 10 | 1010 | SRKW | RW | R | R | - | |
| 11 | 1011 | SR | R | R | R | - | |
| 12 | 1100 | URSW | RW | RW | RW | R | |
| 13 | 1101 | UREW | RW | RW | R | R | |
| 14 | 1110 | URKW | RW | R | R | R | |
| 15 | 1111 | UR | R | R | R | R | |

K = kernel R = read
E = executive W = write
S = supervisor - = no access
U = user

### 2.5.5 Translation Buffer -

In order to save actual memory references when repeatedly referencing pages, CVAX uses a translation buffer to remember successful virtual address translations and page status. The translation buffer contains 32 fully associative entries. Both system and process references share these entries.

Translation buffer entries are replaced using a not last used (NLU) algorithm. NLU guarantees that the replacement pointer is not pointing at the last translation buffer entry to be used. This is accomplished by rotating the replacement pointer to the next sequential translation buffer entry if it is pointing to an entry that has just been accessed. Both D-stream and I-stream references can cause the NLU to cycle. When the translation buffer does not contain a reference's virtual address and page status, the machine updates the translation buffer by replacing the entry that is selected by the replacement pointer.

## 2.6 Exceptions And Interrupts

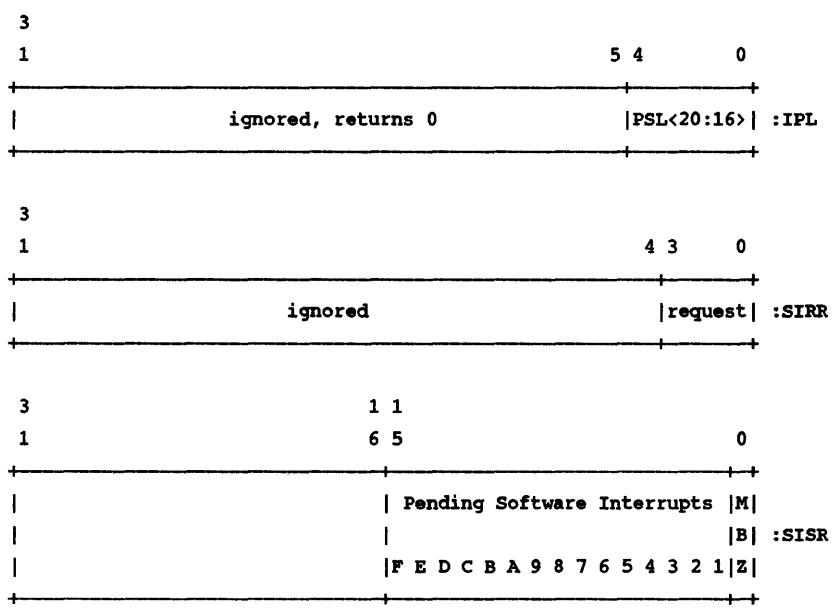
Both exceptions and interrupts divert execution from the normal flow of control. An exception is caused by the execution of the current instruction, while an interrupt is caused by some activity outside the central processor.

### 2.6.1 Interrupts -

The CVAX architecture has 31 interrupt priority levels (IPL), used as follows:

| IPL levels | interrupt condition |
|---|---|
| 1F | unused |
| 1E | PWRFL L asserted |
| 1D | MEMERR L asserted |
| 1B - 1C | unused |
| 1A | CRD L asserted |
| 18 - 19 | unused |
| 17 | IRQ<3> L asserted |
| 16 | INTTIM L asserted |
| 16 | IRQ<2> L asserted |
| 15 | IRQ<1> L asserted |
| 14 | IRQ<0> L asserted |
| 10 - 13 | unused |
| 01 - 0F | software interrupt request |

The interrupt system is controlled by the Interrupt Priority Level Register (IPL, corresponds to PSL<20:16>), the Software Interrupt Request Register (SIRR), and the Software Interrupt Summary Register (SISR).


3
1
5 4
0
ignored, returns 0
PSL<20:16>
:IPL
3
1
4 3
0
ignored
request
:SIRR
3
1
1 1
6 5
0
Pending Software Interrupts
M
B
:SISR
F E D C B A 9 8 7 6 5 4 3 2 1
Z

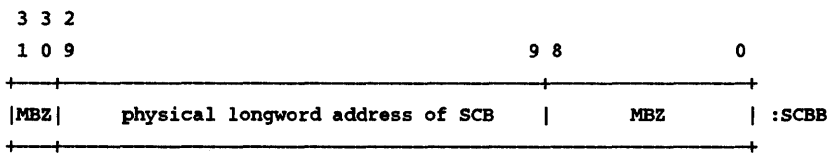
### 2.6.2 Exceptions -

The CVAX architecture recognizes six classes of exceptions.

| exception class | instances |
|---|---|
| arithmetic traps/faults | integer overflow trap<br>integer divide by zero trap<br>subscript range trap<br>floating overflow fault<br>floating divide by zero fault<br>floating underflow fault |
| memory management exceptions | access control violation fault<br>translation not valid fault |
| operand reference exceptions | reserved addressing mode fault<br>reserved operand fault or abort |
| instruction execution exceptions | reserved/privileged instruction fault<br>emulated instruction fault<br>customer reserve instruction fault<br>breakpoint fault |
| tracing exception | trace fault |
| system failure exceptions | machine check abort (including<br>read/write bus and parity errors and<br>cache parity errors)<br>kernel stack not valid abort<br>interrupt stack not valid abort |

### 2.6.3 System Control Block (SCB) -

The System Control Block (SCB) is a page containing the vectors for servicing interrupts and exceptions. The SCB is pointed to by the System Control Block Base Register (SCBB).




The System Control Block format:

| vector | name | type | #param | notes |
|---|---|---|---|---|
| 00 | passive release | interrupt | 0 | IRQ passive release |
| 04 | machine check | abort | 3 | parameters depend upon error type |
| 08 | kernel stack not valid | abort | 0 | must be serviced on interrupt stack |
| 0C | power fail | interrupt | 0 | IPL is raised to 1E |
| 10 | reserved/privileged instruction | fault | 0 | |
| 14 | customer reserved instruction | fault | 0 | XFC instruction |
| 18 | reserved operand | fault/ abort | 0 | not always recoverable |
| 1C | reserved addressing mode | fault | 0 | |
| 20 | access control violation | fault | 2 | parameters are virtual address, status code |
| 24 | translation not valid | fault | 2 | parameters are virtual address, status code |
| 28 | trace pending (TP) | fault | 0 | |
| 2C | breakpoint instruction | fault | 0 | |
| 30 | unused | - | - | compatibility mode in VAX |
| 34 | arithmetic | trap/ fault | 1 | parameter is type code |
| 38-3C | unused | - | - | - |
| 40 | CHMK | trap | 1 | parameter is sign-extended operand word |
| 44 | CHME | trap | 1 | parameter is sign-extended operand word |

| | | | | |
|---|---|---|---|---|
| 48 | CHMS | trap | 1 | parameter is sign-extended operand word |
| 4C | CHMU | trap | 1 | parameter is sign-extended operand word |
| 50 | unused | - | - | - |
| 54 | corrected read data | interrupt | 0 | IPL is 1A (CRD L) |
| 58-5C | unused | - | - | - |
| 60 | memory error | interrupt | 0 | IPL is 1D (MEM ERR L) |
| 64-80 | unused | - | - | - |
| 84 | software level 1 | interrupt | 0 | |
| 88 | software level 2 | interrupt | 0 | ordinarily used for AST delivery |
| 8C | software level 3 | interrupt | 0 | ordinarily used for process scheduling |
| 90-BC | software levels 4-15 | interrupt | 0 | |
| C0 | interval timer | interrupt | 0 | IPL is 16 (INTTIM L) |
| C4 | unused | - | - | - |
| C8 | emulation start | fault | 10 | same mode exception, FPD = 0: parameters are opcode, PC, specifiers |
| CC | emulation continue | fault | 0 | same mode exception, FPD = 1: no parameters |
| D0-FC | unused | - | - | - |
| 100-1FC | adapter vectors | interrupt | 0 | |
| 200-3FC | device vectors | interrupt | 0 | |

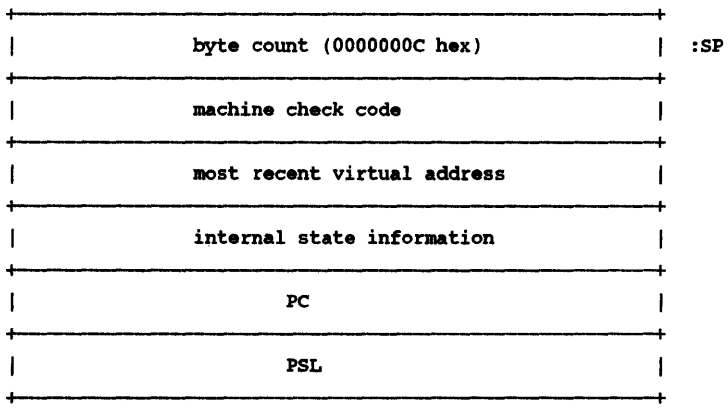
Vectors in the range of 100-3FC are used to directly vector interrupts from the external bus. The SCBB vector index is determined from bits <9:2> of the value supplied by external hardware. The new PSL priority level is determined by either the external interrupt request level that caused the interrupt or by bit <0> of the value supplied by external hardware. If bit<0> is 0, the new IPL level is determined by the interrupt request level being serviced. IRQ<3> sets the IPL to 17 (hex); IRQ<2>, 16 (hex); IRQ<1>, 15 (hex); and IRQ<0>, 14 (hex). If bit<0> of the value suppled by external hardware is 1, then the new IPL is forced to 17 (hex). The ability to force the IPL to 17 (hex) supports an external bus, such as the Q-bus, that can not guarantee that the device generating the SCBB vector index is the device that originally requested the interrupt. For example, the Q-bus has four separate interrupt request signals that correspond to IRQ<3:0> but only one signal to daisy chain the interrupt grant. Furthermore, devices on the Q-bus are ordered so that

higher priority devices are electrically closer to the bus master. If an IRQ<1> is being serviced, there is no guarantee that a higher priority device will not intercept the grant. Software must determine the level of the device that was serviced and set the IPL to the correct value.

Only device vectors in the range of 100 to 3FC (hex) should be used, except by devices emulating console storage and terminal hardware.

### 2.6.4 Machine Check Parameters -

In response to a machine check, the following parameters are pushed on the stack:




The parameters are:

machine check code (hex):

| | |
|---|---|
| 1 | = undefined MMGT.STATUS |
| 2 | = PPTE in P0 space during TB miss flows |
| 3 | = PPTE in P1 space during TB miss flows |
| 4 | = PPTE in P0 space during M=0 flows |
| 5 | = PPTE in P1 space during M=0 flows |
| 6 | = PSL<26:24> = 101 during interrupt or exception |
| 7 | = PSL<26:24> = 110 during interrupt or exception |
| 8 | = PSL<26:24> = 111 during interrupt or exception |
| 9 | = undefined INT.ID value |
| 10 | = PSL<26:24> = 101 during REI |
| 11 | = PSL<26:24> = 110 during REI |
| 12 | = PSL<26:24> = 111 during REI |
| 80 | = memory read error |
| 81 | = SCB, PCB or SPTE read error |
| 82 | = memory write error |
| 83 | = SCB, PCB or SPTE write error |

| | | | |
|---|---|---|---|
| most recent virtual address: | <31:0> | = | current contents of VAP register |
| internal state information: | tbs | | |
| PC: | <31:0> | = | PC at the start of the current instruction |
| PSL: | <31:0> | = | current contents of PSL |

### 2.6.5 Restart Process -

If the hardware or kernel software environment becomes severely corrupted, the chip may be unable to continue normal processing. In these instances, the chip executes a restart process and passes control to recovery code beginning at physical address 20040000 (hex). The current values of the PC, PSL, and MAPEN<0>, and the restart code, are saved in internal registers. The PSL is set to 041F0000 (hex). The current stack pointer is saved in the appropriate internal register and then loaded from the interrupt stack pointer. The restart codes are as follows:

| code | condition |
|---|---|
| 2 | HALT L asserted |
| 3 | RESET L asserted |
| 4 | interrupt stack not valid during exception |
| 5 | machine check during machine check or kernel stack not valid exception |
| 6 | HALT instruction executed in kernel mode |
| 7 | SCB vector bits<1:0> = 11 |
| 8 | SCB vector bits<1:0> = 10 |
| A | CHMx executed while on interrupt stack |
| 10 | ACV or TNV during machine check exception |
| 11 | ACV or TNV during kernel stack not valid exception |

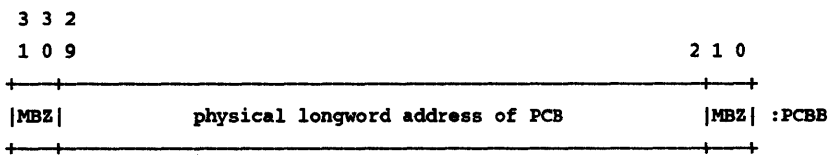
The restart process sets the state of the chip as follows:

| | | |
|---|---|---|
| proc reg SAVPC | = | saved PC |
| proc reg SAVPSL | = | saved PSL<31:16,7:0> in <31:16,7:0> |
| | | saved MAPEN<0> in <15> |
| | | valid PSL flag in <14> |
| | | saved restart code in <13:8> |
| SP | = | current interrupt stack |
| PSL | = | 041F0000 (hex) |
| PC | = | 20040000 (hex) |
| MAPEN | = | 0 |
| ICCS | = | 0 (power up only) |
| MSER | = | 0 (power up only) |
| CADR | = | 0 (power up only) |
| SISR | = | 0 (power up only) |
| ASTLVL | = | 4 (power up only) |
| all else | = | undefined |

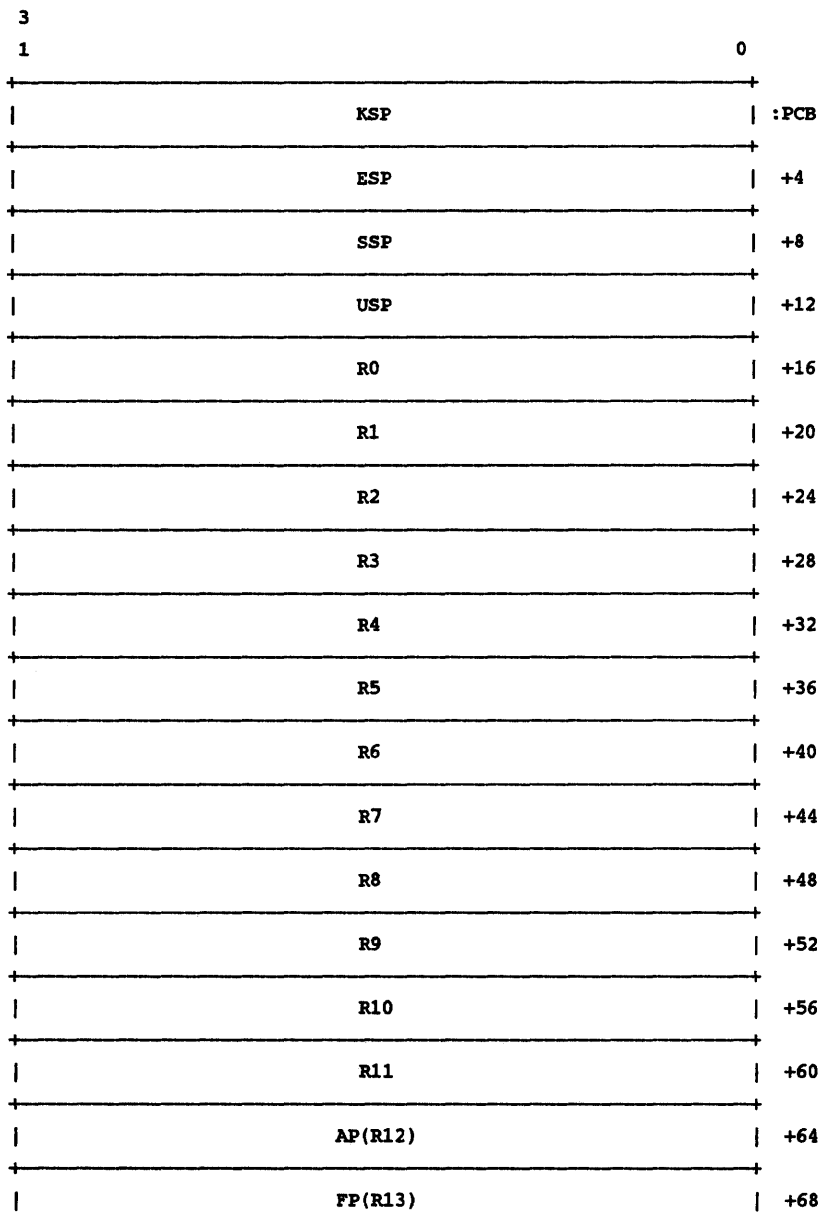
The current SP at the time of the reset is saved in its corresponding IPR.

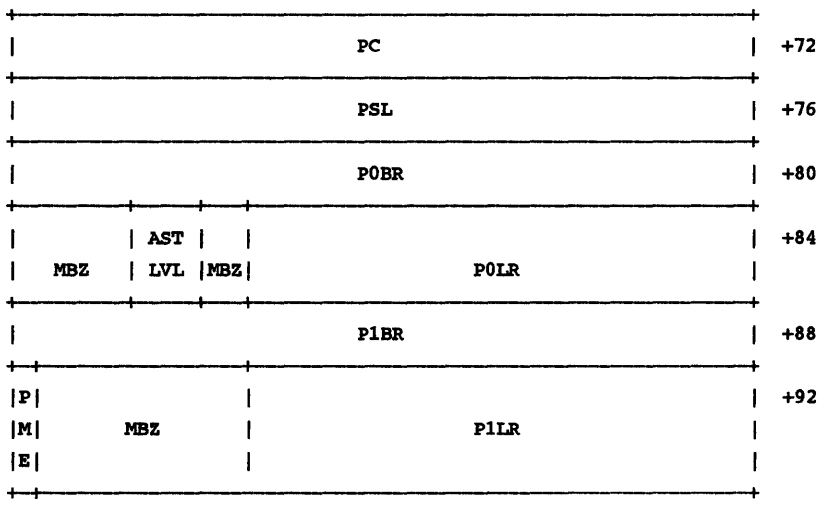
## 2.7 Process Structure

A process is a single thread of execution. The context of the current process is contained in the Process Control Block (PCB). The PCB is pointed to by the Process Control Block Base register (PCBB).




### 2.7.1 Process Control Block (PCB) -






Note: The PME field is unused.

## 2.8 Processor Registers

Each of the processor registers listed in the table below falls into one of the following categories:

1 = implemented by the CVAX CPU Chip as specified in the VAX Architecture Standard (DEC Standard 032)
2 = implemented by the CVAX CPU Chip uniquely
3 = passed to external logic via an external processor register cycle; if not implemented externally, read as zero, nopped on write
4 = access not allowed (reserved operand fault)

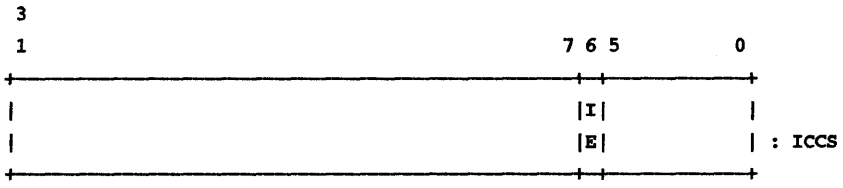
| Number | Register Name | Mnemonic | Type | Scope | Init? | Category |
|---|---|---|---|---|---|---|
| 0 | Kernel Stack Pointer | KSP | rw | proc | — | 1 |
| 1 | Executive Stack Pointer | ESP | rw | proc | — | 1 |
| 2 | Supervisor Stack Pointer | SSP | rw | proc | — | 1 |
| 3 | User Stack Pointer | USP | rw | proc | — | 1 |
| 4 | Interrupt Stack Pointer | ISP | rw | cpu | — | 1 |
| 5 | reserved | — | — | — | — | 4 |
| 6 | reserved | — | — | — | — | 4 |
| 7 | reserved | — | — | — | — | 4 |
| 8 | P0 Base Register | P0BR | rw | proc | — | 1 |
| 9 | P0 Length Register | P0LR | rw | proc | — | 1 |
| 10 | P1 Base Register | P1BR | rw | proc | — | 1 |
| 11 | P1 Length Register | P1LR | rw | proc | — | 1 |
| 12 | System Base Register | SBR | rw | cpu | — | 1 |
| 13 | System Length Register | SLR | rw | cpu | — | 1 |
| 14 | reserved | — | — | — | — | 4 |
| 15 | reserved | — | — | — | — | 4 |
| 16 | Process Control Block Base | PCBB | rw | proc | — | 1 |
| 17 | System Control Block Base | SCBB | rw | cpu | — | 1 |
| 18 | Interrupt Priority Level | IPL | rw | cpu | yes | 1 |
| 19 | AST Level | ASTLVL | rw | proc | yes | 1 |
| 20 | Software Interrupt Request | SIRR | w | cpu | — | 1 |
| 21 | Software Interrupt Summary | SISR | rw | cpu | yes | 1 |
| 22 | Interprocessor Interrupt | IPIR | rw | cpu | yes | 4 |
| 23 | CMI Error Register | CMIERR | r | cpu | yes | 4 |
| 24 | Interval Clock Control | ICCS | rw | cpu | yes | 2 |
| 25 | Next Interval Count | NICR | w | cpu | — | 3 |
| 26 | Interval Count | ICR | r | cpu | — | 3 |
| 27 | Time Of Year | TODR | rw | cpu | no | 3 |
| 28 | Console Storage Receiver Status | CSRS | rw | cpu | yes | 3 |
| 29 | Console Storage Receiver Data | CSRD | r | cpu | — | 3 |
| 30 | Console Storage Transmitter Status | CSTS | rw | cpu | yes | 3 |
| 31 | Console Storage Transmitter Data | CSTD | w | cpu | — | 3 |
| 32 | Console Receiver Status | RXCS | rw | cpu | yes | 3 |
| 33 | Console Receiver Data | RXDB | r | cpu | — | 3 |

| | | | | | | |
|---|---|---|---|---|---|---|
| 34 | Console Transmitter Status | TXCS | rw | cpu | yes | 3 |
| 35 | Console Transmitter Data | TXDB | w | cpu | — | 3 |
| 36 | Translation Buffer Disable | TBDR | rw | cpu | — | 3 |
| 37 | Cache Disable | CADR | rw | cpu | yes | 2 |
| 38 | Machine Check Error Summary | MCER | rw | cpu | — | 3 |
| 39 | Memory System Error | MSER | rw | cpu | yes | 2 |
| | | | | | | |
| 40 | Accelerator Control/Status | ACCS | rw | cpu | yes | 4 |
| 41 | Accelerator Maintenance | ACCR | r/w | cpu | no | 4 |
| 42 | Console Saved PC | SAVPC | r | cpu | — | 2 |
| 43 | Console Saved PSL | SAVPSL | r | cpu | — | 2 |
| 44 | WCS Address | WCSA | rw | cpu | no | 4 |
| 45 | WCS Data | WCSD | rw | cpu | yes | 4 |
| 46 | reserved | — | — | — | — | 4 |
| 47 | reserved | — | — | — | — | 4 |
| | | | | | | |
| 48 | SBI Fault/Status | SBIFS | rw | cpu | yes | 3 |
| 49 | SBI Silo | SBIS | r | cpu | no | 3 |
| 50 | SBI Silo Comparator | SBISC | rw | cpu | yes | 3 |
| 51 | SBI Maintenance | SBIMT | rw | cpu | yes | 3 |
| 52 | SBI Error Register | SBIER | rw | cpu | yes | 3 |
| 53 | SBI Timeout Addresss | SBITA | r | cpu | — | 3 |
| 54 | SBI Quadword Clear | SBIQC | w | cpu | — | 3 |
| 55 | IO Bus Reset | IORESET | w | cpu | — | 3 |
| | | | | | | |
| 56 | Memory Management Enable | MAPEN | rw | cpu | yes | 1 |
| 57 | Trans. Buf. Invalidate All | TBIA | w | cpu | — | 1 |
| 58 | Trans. Buf. Invalidate Single | TBIS | w | cpu | — | 1 |
| 59 | Translation Buffer Data | TBDATA | rw | cpu | — | 3 |
| 60 | Microprogram Break | MBRK | rw | cpu | — | 3 |
| 61 | Performance Monitor Enable | PMR | rw | proc | yes | 3 |
| 62 | System Identification | SID | r | cpu | no | 1 |
| 63 | Translation Buffer Check | TBCHK | w | cpu | — | 1 |
| | | | | | | |
| 64:127 | reserved | — | — | — | — | 4 |

The implementation specific processor registers are described below.

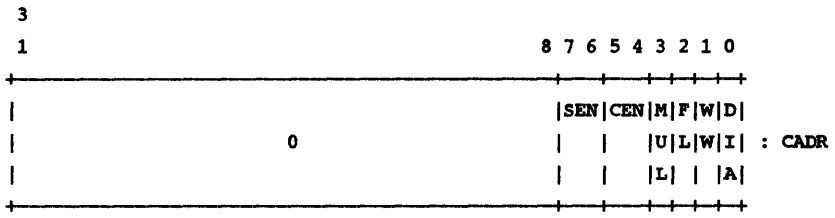
### 2.8.1 Interval Clock Control And Status Register (ICCS) -

The ICCS register controls the interval timer (INTTIM L) interrupt. It is similar to the ICCS register in the full VAX architecture but contains only a single bit to enable or disable the interval timer interrupt:




Bit <6> is read/write. When set, interval timer interrupts are enabled at IPL16; when clear, interval timer interrupts are disabled. Bits <31:7,5:0> read as zero and are ignored on writes. Bit <6> is cleared in the restart process.

2.8.2 Cache Disable Register (CADR) -




CADR <31:8> always read as 0's. CADR <7:0> initialize to 0 when RESET L is asserted. CADR <7:6> (Set Enable) are read/write and are encoded as follows:

| <7:6> | Set 1 | Set 0 |
|---|---|---|
| 00 | disabled | disabled |
| 01 | disabled | enabled |
| 10 | enabled | disabled |
| 11 | enabled | enabled |

CADR <5:4> (Cache Enable) are read/write and are encoded as follows:

| <5:4> | Action |
|---|---|
| 00 | Cache disabled |
| 01 | D-stream only stored in cache (diagnostic mode) |
| 10 | I-stream only stored in cache |
| 11 | I-stream and D-stream stored in cache |

When CADR <5:4> = 10 (I-stream only stored in cache), CVAX automatically flushes the cache whenever an REI instruction is executed. The VAX SRM states that an REI instruction must be executed prior to running code out of an updated page of memory. Therefore, systems that follow the SRM need not monitor DMA writes in order to prevent stale data from accumulating in the cache. When CADR<5:4> = 11 or 01, invalidate-on-hit cycles must be used to remove stale data from the cache.

CADR <3> (Multiple Transfers) is read/write. When set, multiple transfer

read cycles are generated under certain conditions. A quadword multiple transfer read is initiated when all of the following conditions are met:

1. The read reference is to memory.

2. The read reference can potentially be stored in the cache but is not stored in the cache.

3. Multiple transfers are selected.

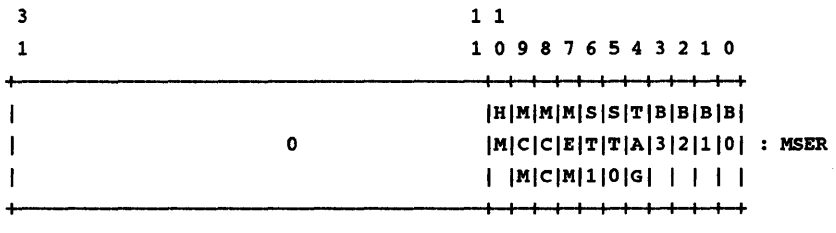
When CADR <3>=1, all read references which can not be potentially stored in the cache, initiate single transfer read cycles. These includes all I/O space references; if CADR <7> = 0, all memory space instruction stream references; if CADR <6> = 0, all memory space data stream references; and all read lock references. When CADR<3>=0, all read references initiate single transfer read cycles.

CADR <2> (Flush) is write only and always reads as a 0. When written with a 1, the contents of the cache are flushed.

CADR <1> (Write Wrong Parity) is read/write. When set, incorrect parity is stored in the cache whenever the cache is written.

CADR <0> (Diagnostic mode) is read/write. When set, all write references write through the cache irrespective of hit/miss or bus error status. In addition, no bus error induced machine check trap is generated during a bus write. When cleared, a normal cache write through operation occurs during write cycles. No allocation is done. Diagnostic mode should only be selected when one set is enable.

2.8.3 Memory System Error Register (MSER) -




MSER <31:11> always read as 0's. MSER <9:0> are read/write and initialize to 0's when RESET L is asserted. MSER <10> is read-only and initializes to 0 when RESET L is asserted.

MSER <10> (hit/miss) is updated only on references that can be potentially stored in the cache. This excludes all I/O space references; if CADR <7> = 0, all memory space instruction stream references; if CADR <6> = 0, all memory space data stream references; and all read lock references. On all references that qualify, MSER <10> is set if the reference is stored in the cache and cleared if it is not stored in the cache.

MSER <9> (machine check memory parity error) is set whenever the memory parity error information logged in MSER<7:0> caused a machine check.

MSER <8> (machine check cache parity error) is set whenever the cache parity error information logged in MSER<7:0> caused a machine check.

MSER <4:0> are individually set when a parity error occurs to show the scope of the corrupted data. Specifically, <4> indicates a tag parity error; <3>, byte 3 parity error; <2>, byte 2 parity error; <1>, byte 1 parity error; and finally, <0> byte 0 parity error. In parallel, MSER <7:5> are individually set to indicate the source of the of the parity error. Specifically, <7> indicates memory; <6>, cache data in set 1; <5>, cache data in set 0.

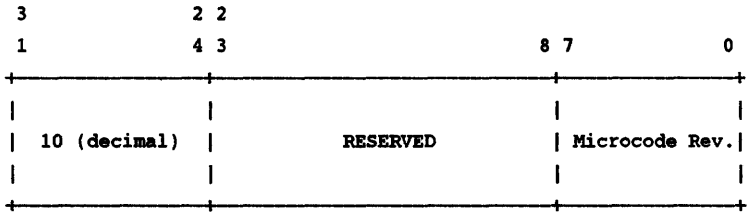
MSER <9:0> are sticky in the sense that once set, they remain set until the entire register is cleared by any MTPR MSER instruction. Parity errors occuring while an error condition is posted in MSER can set, but never clear, additional MSER bits.

### 2.8.4 Console Saved Registers (SAVPC, SAVPSL) -

The console saved registers (SAVPC, SAVPSL) record the value of the PC and PSL, respectively, at the time a chip restart occurs. See the section on Restarts, above, for details.

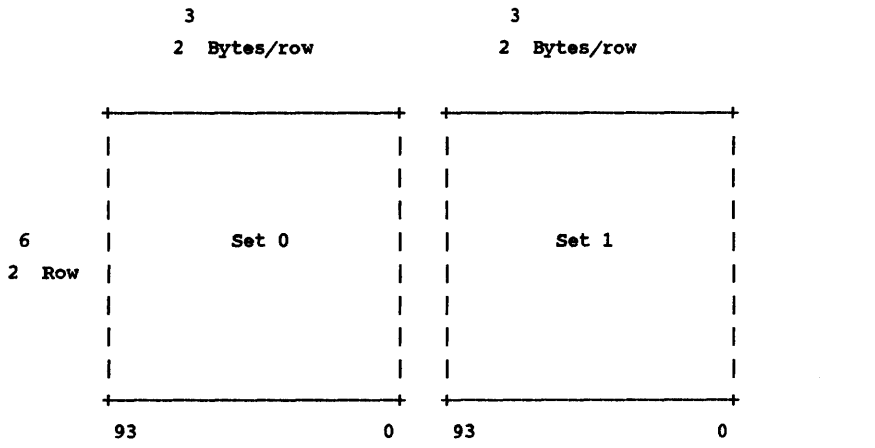
### 2.8.5 System Identification Register (SID) -

The SID is a read only constant register that specifies the processor type as a CVAX (SID<31:24>=10 decimal). The SID<7:0> reflects the microcode revision level.



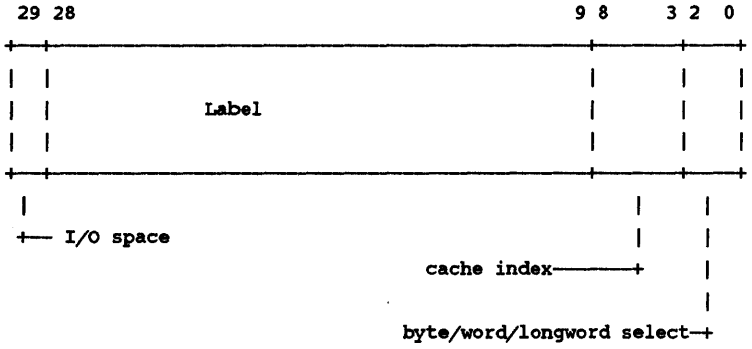
## 3.0 INTERNAL CACHE

CVAX contains a 1Kb, two-way associative, 8 byte block size cache:

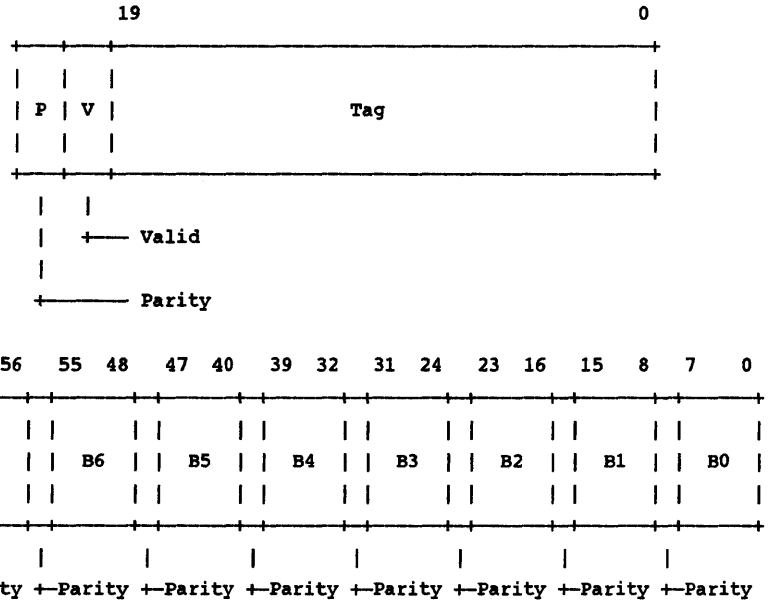



Total storeage = $2^1$ sets * $2^6$ rows * $2^3$ bytes per row = 1024 bytes

Each physical address is logically subdivided as follows:



When bit <29> is set, I/O space is referenced. No I/O space reference is ever stored in the cache. When the reference is in memory space, bits <8:3> access an entry in each set of the cache. Each entry contains tag and data information and is organized as follows:




A memory reference is stored in the cache when all of the following conditions are met:

1. The physical address label field exactly equals the cache tag field.

2. The cache tag valid bit is set.

3. No parity errors exist in the tag field.

4. No parity errors exist in the four bytes selected by bit <2> of the physical address, i.e., B3-B0 if B<2>=0; B7-B4 if B<2>=1.

## 3.1 Cache Allocation

A cache block (8 bytes) is allocated whenever a read reference that can be potentially stored in the cache is not stored in the cache. This excludes all I/O space references; if CADR <7> = 0, all memory space instruction stream references; if CADR <6> = 0, all memory space data stream references; and all read lock references. Random set selection is used. External logic can determine which set is being selected by monitoring the CS/DP<3> pin.

## 3.2 Read Cycle Classification

The CVAX chip classifies read cycles into three groups: request I-stream reads; request D-stream reads; and demand D-stream reads. In general, request reads are generated whenever the data is not immediately needed by the machine flows. Specifically, prefetching the I-stream (request I-stream) and filling the second cache longword during a D-stream read (request D-stream) generate request reads. A D-stream demand read is generated whenever data is immediately needed by the machine flows. Specifically, operand, PTE, SCB, and PCB references all generate D-stream demand cycles.

Demand and request reads differ in the way they respond to errors reported during the reference. In general, request reads do not effect the machine flow whereas demand reads cause a machine check. The following tables highlight the effects various errors have during demand and request cycles.

Bus Error effect (including DAL parity errors)

| | Prefetcher | Cache | Error Status | Machine flows |
|---|---|---|---|---|
| Demand D-stream | - | entry is invalidated** | logged in MSES<7,3:0>* | Machine check fault |
| Request D-stream | - | entry is invalidated** | logged in MSER<7,3:0>* | - |
| Request I-stream | halted | entry is invalidated** | logged in MSER<7,3:0>* | - |

* only DAL parity errors log status

** the entire cache row selected by the faulting address is invalidated irrespective of whether the reference is cacheable, i.e., the entries from both sets are invalidated.

Cache Parity Error effect

| | Prefetcher | Cache | Error Status | Machine flows |
|---|---|---|---|---|
| Demand D-stream | - | flush cache & disable caching | logged in MSER<6:0> | Machine check abort |
| Request D-stream | - | flush cache | logged in MSER<6:0> | - |
| Request I-stream | halted | flush cache | logged in MSER<6:0> | - |

Memory Management Error effect

| | Prefetcher | Machine flows |
|---|---|---|
| Demand D-stream | - | memory management fault (ACV,TNV,etc.) |
| Request D-stream | not possible | not possible |
| Request I-stream | halted | - |

- = not effected

To aid CVAX users, the demand D-stream references are further classified as read lock; read no modify intent; and read modify intent. Classification information is passes to external logic on the CS/DP<2:0> L pins.

## 3.3 Cache Parity

CVAX protects the internal cache with parity. Each eight bit byte of cache data and the eighteen bit tag field is checked by a parity bit. Odd data bytes store odd parity; even data bytes, even parity. The tag field stores odd parity. The stored parity is valid only when the valid bit associated with the cache entry is set.

The action following the detection of a cache parity error depends on the reference type: during a demand D-stream reference, the entire cache is flushed, the cache is turned off (CADR is cleared), the cause of the error is logged in MSER<6:0> and a machine check abort is initiated. During an request D-stream reference, the entire cache is flushed, the cause of the error is logged in MSER<6:0>, but no abort occurs. During a request I-stream reference, the entire cache is flushed, the cause of the error is logged in MSER<6:0>, prefetching is halted, but no abort occurs.

## 3.4 DAL H Parity

CVAX protects DAL data with parity. Each eight bit DAL byte is conditionally checked by a parity bit. Odd data bytes show odd parity; even data bytes, even parity. The parity sense is alternated in order to

catch both stuck at one and stuck at zero faults. DAL H parity checking can be disable, reference by reference, by deasserting the external pin DPE L.

The action following the detection of a DAL H parity error depends on the reference type: during a demand D-stream reference, the cache entry is invalidated, the cause of the error is logged in MSER<7,3:0>, and a machine check abort is initiated. During an request D-stream reference, the cache entry is invalidated, the cause of the error is logged in MSER<7,3:0>, but no abort occurs. During a request I-stream reference, the cache entry is invalidated, the cause of the error is logged in MSER<7,3:0>, but no abort occurs.

# 4.0 INTERFACE

This section details the pinouts for the CVAX CPU chip.

## 4.1 Pinouts

### 4.1.1 Summary - The CVAX CPU chip has the following pinouts:

| Signals | Signal Type | Number of Pins | Running Total |
|---|---|---|---|
| Data and Address | IO | 32 | 32 |
| Cycle status/parity | IO | 4 | 36 |
| Data parity enable | IO | 1 | 37 |
| Address strobe | IO | 1 | 38 |
| Data strobe | O | 1 | 39 |
| Byte mask | O | 4 | 43 |
| Write | O | 1 | 44 |
| Data buffer enable | O | 1 | 45 |
| Ready | I | 1 | 46 |
| Error | I | 1 | 47 |
| Reset | I | 1 | 48 |
| Halt | I | 1 | 49 |
| Interrupt request | I | 4 | 53 |
| Power fail | I | 1 | 54 |
| Corrected read data | I | 1 | 55 |
| Interval timer | I | 1 | 56 |
| DMA request | I | 1 | 57 |
| DMA grant | O | 1 | 58 |
| Cache control | I | 1 | 59 |
| Coprocessor data | IO | 6 | 65 |
| Coprocessor status | IO | 2 | 67 |
| Power | I | 5 | 72 |
| Ground | I | 5 | 77 |
| Clock in | I | 2 | 79 |
| Test | IO | 2 | 81 |
| Bus Request | O | 1 | 82 |
| Console mode | O | 1 | 83 |
| Memory Error | I | 1 | 84 |

### 4.1.2 Data And Address Bus -

#### 4.1.2.1 Data And Address Lines (DAL<31:00>) -

The Data and Address Bus (DAL<31:00>) is a bi-directional time-multiplexed bus. During the first part of a CPU read cycle or CPU write cycle, DAL<31:30> indicate the length of the memory operand (00 = reserved, 01 = longword, 10 = quadword, 11 = reserved for DMA octaword transfers), and DAL<29:02> contain the LONGWORD address of the memory operand (DAL<29> distinguishes memory space from I/O space). DAL<01:00> are reserved (see the Memory Access Protocol) and may differ from the address implied by BM<3:0> L in the following circumstances:

- During an instruction prefetch (always an aligned longword)
- During a character string data prefetch (always an aligned longword)
- During a PTE read (always an aligned longword)
- During the second cycle of an unaligned operation.

During the first part of an interrupt acknowledge cycle, DAL<06:02> contain the IPL of the interrupt being acknowledged and DAL<31:7,1:0> are 0. During the first part of an external processor register read or write cycle, DAL<7:2> contain the IPR number of the register that is being accessed and DAL<31:8,1:0> are 0. During the second part of a CPU read, external processor register read or interrupt acknowledge cycle, DAL<31:00> is used to receive incoming information. During the second part of a CPU write or external processor register write cycle, DAL<31:00> is used to transmit outgoing information. The DAL bus is also used to exchange information with an external floating point processor.

Control of DAL<31:0> H is relinquished whenever DMG L is asserted.

#### 4.1.2.2 Cycle Status/Data Parity (CS/DP<3:0> L) -

CS/DP<3:0> are time-multiplexed signals. During the first part of IO cycles, CS/DP<3:0>, in conjunction with the WRITE signal (WR L), provide status about the current bus cycle. Specifically, WR L and CS/DP<2:0> mean the following when AS L is asserted:

| WR L | CS/DP<2:0> | Bus Cycle Type |
|---|---|---|
| H | LLL | Request D-stream read |
| H | LLH | reserved |
| H | LHL | external IPR read |
| H | LHH | interrupt acknowledge |
| H | HLL | request I-stream read |
| H | HLH | demand D-stream read (lock) |
| H | HHL | demand D-stream read (modify intent) |

| | | |
|---|---|---|
| H | HHH | demand D-stream read (no lock or modify intent) |
| L | LLL | reserved |
| L | LLH | reserved |
| L | LHL | external IPR write |
| L | LHH | reserved for DMA device use |
| L | HLL | reserved |
| L | HLH | write unlock |
| L | HHL | reserved |
| L | HHH | write no unlock |

NOTE

EXTERNAL IPRs ARE ACCESSED WITH THE SAME PROTOCOL AS MEMORY. THEREFORE, RDY L OR ERR L MUST BE ASSERTED TO TERMINATE THESE CYCLES; this is a change from the MicroVAX CPU chip.

During the first part of a cacheable read cycle, CS/DP<3> provides status about which cache set is being allocated. CS/DP<3> is undefined during all other cycles. This signal allows a memory system to build a data coherent external cache. CS/DP<3> means the following when AS L is asserted:

| CS/DP<3> | Cache set information |
|---|---|
| 0 | set 1 is being allocated |
| 1 | set 2 is being allocated |

During the second part of IO cycles, CS/DP<3:0> L provide byte parity for the DAL bus data during a CPU read, CPU write, or external processor write cycle. No parity checking is done on a CVAX/coprocessor data transfer, external processor register read, or interrupt acknowledge cycle. Even parity is checked/generated on even bytes; odd parity on odd bytes. Even parity will drive a 0 when there are an even number of 1s in the byte's data; odd parity will drive a 0 for an odd number of 1s. CS/DP<3> L is the parity signal for DAL<31:24>, CS/DP<2> L for DAL<23:16>, CS/DP<1> L for DAL<15:8>, CS/DP<0> L for DAL<7:0>. On a CPU read, the CPU reads and checks data parity for the bytes specified by BM<3:0> L if Data Parity Enable (DPE L) is asserted. On a CPU write or external processor register write cycle, the CPU generates data parity for all bytes, irrespective of BM<3:0> L.

NOTE

DURING READ CYCLES, CS/DP<3:0> L MUST BE ASSERTED SYNCHRONOUSLY WITH RESPECT TO THE CHIP'S TIMING SAMPLING POINT AND THEREFORE MUST NOT CHANGE DURING THE SAMPLE WINDOW.

Control of CS/DP<3:0> L is relinquished whenever DMG L is asserted.

4.1.2.3 Data Parity Enable (DPE L) -

Data Parity Enable (DPE L) is a bi-directional signal.

During a CPU read and interrupt acknowledge cycle, DPE L is asserted by external logic in conjunction with the DAL data in order to enable parity checking on the incoming DAL data. When deasserted, the DAL Parity lines are ignored. DPE L must be externally pulled up by an external resistor to the unasserted state, and therefore any interface which wants CVAX to check DAL H parity must actively assert DPE L.

During a CPU write or external processor register write cycle, DPE L will be asserted by CVAX in conjunction with the DAL data in order to indicate that valid parity information is present.

NOTE

DURING A CPU READ CYCLE, DPE L MUST BE ASSERTED SYNCHRONOUSLY WITH RESPECT TO THE CHIP'S TIMING SAMPLING POINT AND THEREFORE MUST NOT CHANGE DURING THE SAMPLE WINDOW.

Control of DPE L is relinquished whenever DMG L is asserted.

4.1.3 Bus Control -

4.1.3.1 Address Strobe (AS L) -

Address Strobe signal (AS L) is a bi-directional signal. CVAX drives AS L to provide timing and control information to external logic. During a CPU read, CPU write, external processor register read, external processor register write, or interrupt acknowledge cycle, the chip asserts AS L when the initial information on DAL<31:00> and CS/DP<3:0> L is valid. The chip deasserts AS L at the conclusion of the bus cycle.

External logic drives AS L to provide an asynchronous address timing strobe. During a DMA cache invalidate cycle, AS L is asserted to latch the DMA address into the CPU.

Control of AS L is relinquished whenever DMG L is asserted.

#### 4.1.3.2 Data Strobe (DS L) -

The Data Strobe signal (DS L) provides timing information for data transfers. During a CPU read (single or multiple transfers), external processor register read, or interrupt acknowledge cycle, the chip asserts DS L to indicate that DAL<31:00> and CS<3:0>/DP L are free to receive incoming data, and deasserts DS L to indicate that it has received and latched the incoming data. During a CPU write or external processor register write cycle, the chip asserts DS L to indicate that DAL<31:00> and CS/DP<3:0> L contain valid outgoing data, and deasserts DS L to indicate that the data is about to be removed.

Control of DS L is relinquished whenever DMG L is asserted.

#### 4.1.3.3 Byte Masks (BM<3:0> L) -

The byte mask signals (BM<3:0> L) specify which bytes of the DAL bus contain valid information during the second part of an IO cycle. If BM<3> L is asserted, then DAL<31:24> contain valid data; if BM<2> L, then DAL<23:16>; if BM<1> L, then DAL<15:8>; if BM<0> L, then DAL<7:0>. During CPU read cycles, the byte masks indicate which bytes of data, and which bits of parity, must be placed on the DAL and Data Parity lines; if this amounts to less than 32 bits, the other bytes of the DAL and bits of Data Parity are ignored. During a CPU write cycle, the byte masks specify which bytes of the DAL bus, and which Data Parity bits, contain valid data. External processor register cycles always read and write four bytes. Therefore, BM<3:0> L will be asserted during these cycles. BM<3:0> L are valid when AS L is asserted.

Control of BM<3:0> L is relinquished whenever DMG L is asserted.

#### 4.1.3.4 Write (WR L) -

The Write signal (WR L) specifies the direction of data transfer on the DAL bus. If WR L is asserted, then the chip is driving data onto the DAL. If WR L is not asserted, the chip is not driving data onto the DAL. WR L can be used to control the direction input of the DAL transceivers. WR L is valid when AS L is asserted.

Control of WR L is relinquished whenever DMG L is asserted.

#### 4.1.3.5 Data Buffer Enable (DBE L) -

The Data Buffer Enable signal (DBE L) is used in conjunction with the WR L signal to control the external DAL transceivers. The chip asserts DBE L to enable the DAL transceivers, and deasserts it to disable them.

Control of DBE L is relinquished whenever DMG L is asserted.

4.1.3.6 Ready (RDY L) -

External logic asserts the Ready signal (RDY L) to signal normal termination of the current CPU read, CPU write, external processor register read, external processor register write, or interrupt acknowledge cycle. During a CPU read, external processor read or interrupt acknowledge cycle, this indicates that external logic has placed the required input data on the DAL bus in time for the next timing sampling point (see the timing diagrams). During a CPU write or external processor register write cycle, this indicates that the information on the DAL bus has been received and can be removed following the next timing sampling point. Upon assertion of RDY L, the chip terminates the current bus cycle and proceeds. External logic then deasserts RDY L.

NOTE

NOTE THAT READY L MUST BE ASSERTED SYNCHRONOUSLY WITH RESPECT TO THE CHIP'S TIMING SAMPLING POINT AND THEREFORE MUST NOT CHANGE DURING THE SAMPLE WINDOW; this is a change from the MicroVAX CPU chip.

4.1.3.7 Error (ERR L) -

External logic asserts the Error signal (ERR L) to signal abnormal termination of the current CPU read, external processor read, external processor write, CPU write cycle, or interrupt acknowledge cycle. The interpretation of ERR L depends on whether RDY L is also asserted.

- ERR L asserted, RDY L asserted. This causes the chip to RETRY the current bus cycle.

- ERR L asserted, RDY L not asserted. This causes the chip to ABORT the current bus cycle.

During a CPU demand read or CPU write cycle, an abort causes a machine check. During a CPU request read, an abort causes the prefetched data to be discarded. During an external processor read cycle, an abort will cause the read data to appear as 0. During an external processor write cycle, an abort is ignored. During an interrupt acknowledge cycle, an abort causes an interrupt through SCB vector 0. The abort action will only be taken if RDY L is deasserted for two consecutive ERR L/RDY sample points. In fact, if the abort response (ERR L asserted, RDY L deasserted) is detected at the first sample point, but RDY L is asserted at the second sample point, the cycle will be terminated and retried. This action eliminates a timing hazard that is possible when issuing a RETRY in systems that externally synchronize RDY L and ERR L.

When ever any cycle is retried, DMG L will be asserted in response to DMR

L prior to retrying the cycle. When CVAX regains mastership of the DAL, the retried cycle is guaranteed to be immediately repeated unless it was the second longword read (non-prefered data in a quadword block) used to fill a cache entry. In this case, the cycle is never repeated and the partially allocated cache entry is invalidated. This is true irrespective of whether or not multiple transfers are enable in CADR.

NOTE

NOTE THAT ERR L MUST BE ASSERTED SYNCHRONOUSLY WITH RESPECT TO THE CHIP'S TIMING SAMPLING POINT AND THEREFORE MUST NOT CHANGE DURING THE SAMPLE WINDOW; this is a change from the MicroVAX CPU chip.

#### 4.1.4 System Control -

##### 4.1.4.1 Reset (RESET L) -

External logic asynchronously asserts the Reset signal (RESET L) to force the chip to its initial power up state.

NOTE

THE DEASSERTION OF RESET L MUST BE EXTERNALLY SYNCHRONIZED SO THAT THE FIRST RISING EDGE OF CLKA FOLLOWING THE DEASSERTION OF RESET CORRESPONDS TO P1.

When RESET L is asserted, the DAL lines are tri-stated, and all control line outputs are driven to the deasserted state. When RESET L is deasserted, the chip enters the restart process with the restart code = 3 (RESET L asserted).

##### 4.1.4.2 Halt (HALT L) -

External logic asserts the Halt signal (HALT L) to transfer control to console macrocode. At the conclusion of the current macroinstruction, the chip enters the restart process with the restart code = 2 (HALT L asserted). HALT L is edge- rather than level-sensitive, is sampled during P2 of every microcycle, and is internally used during P1 (internal

synchronizer settling time is two clock phases - nominally 50ns).

### 4.1.5 Interrupt Control -

#### 4.1.5.1 Interrupt Request (IRQ<3:0> L) -

The Interrupt Request signals (IRQ<3:0> L) allow external logic to input interrupt requests to the chip. IRQ<3> L corresponds to BR7 and interrupts at IPL17; IRQ<2> L to BR6, IPL16; IRQ<1> L to BR5, IPL15; IRQ<0> L to BR4, IPL14. When taken, interrupt requests are acknowledged by an interrupt acknowledge cycle. IRQ<3:0> L are level-sensitive, are sampled during P2 of every microcycle, and are internally used during P1 (synchronizer settling time is two clock phases - nominally 50ns).

#### 4.1.5.2 Power Fail (PWRFL L) -

The Power Fail signal (PWRFL L) allows external logic to signal a power fail condition to the chip. PWRFL L interrupts at IPL1E (SCB vector 0C hex). A power fail interrupt is NOT acknowledged by the chip. PWRFL L is edge- rather than level-sensitive, is sampled during P2 of every microcycle, and is internally used during P1 (synchronizer settling time is two clock phases - nominally 50ns).

#### 4.1.5.3 Corrected Read Data (CRD L) -

The Corrected Read Data signal (CRD L) allows external logic to signal an ECC error to the chip. CRD L interrupts at IPL1A (SCB vector 54 hex). A corrected read data interrupt is NOT acknowledged by the chip. CRD L is edge- rather than level-sensitive, is sampled during P2 of every microcycle, and is internally used during P1 (synchronizer settling time is two clock phases - nominally 50ns).

#### 4.1.5.4 Interval Timer (INTTIM L) -

The Interval Timer signal (INTTIM L) allows external logic to signal an interval timer rollover to the chip. INTTIM L interrupts at IPL16 (SCB vector C0 hex). An interval timer interrupt is NOT acknowledged by the chip. INTTIM L is edge- rather than level-sensitive, is sampled during P2 of every microcycle, and is internally used during P1 (synchronizer settling time is two clock phases - nominally 50ns).

4.1.5.5 Memory Error (MEMERR L) -

The Memory Error signal (MEMERR L) allows external logic to signal an memory error to the chip. MEMERR L permits the implementation of a memory subsystem with multiple write buffers or delayed writes. When the CPU writes, this type of memory subsystem latches the data and address and asserts RDY immediately. Then, if an error occurs, it is reported via the MEMERR interrupt. MEMERR L interrupts at IPL1D (SCB vector 60 hex). A Memory Error interrupt is NOT acknowledged by the chip. MEMERR L is edge- rather than level-sensitive, is sampled during P2 of every microcycle, and is internally used during P1 (synchronizer settling time is two clock phases - nominally 50ns).

4.1.6 DMA Control -

4.1.6.1 DMA Request (DMR L) -

The DMA Request signal (DMR L) is asserted by external logic which wishes to take control of the DAL bus and related control signals for DMA or other purposes. DMR L is level-sensitive, is sampled during P4 of every microcycle, and is internally used during P3 (synchronizer settling time is two clock phases - nominally 50ns).

4.1.6.2 DMA Grant (DMG L) -

The DMA Grant signal (DMG L) is asserted by the chip to grant control of the DAL bus and related control signals to external logic. The chip tristates the DAL bus and the following strobe signals: AS L, DS L, DBE L, CS/DP<3:0> L, and WR L. When external logic deasserts DMR L, the chip responds by deasserting DMG L and then starts the next bus cycle.

4.1.7 Cache Control (CCTL L) -

CCTL L has two functions: during DMA write operation, it is used to start a conditional cache invalidate operation; during CVAX memory reads, it is used to prevent data caching.

4.1.7.1 Conditional Cache Invalidate -

Since CVAX has an internal cache, it must be able to monitor external (DMA) write traffic in order to prevent cache data from becoming stale. In such a situation, a DMA device writes a memory location which is also stored in the CVAX cache. Only memory is updated. Therefore, in order to

guarantee that the cache is free of stale data, this address "collision" must be detected and the corresponding cache entry must be invalidated.

Each conditional invalidate operation detects a collision on a quadword cache entry. Two consecutive conditional invalidate cycles can be used to detect a collision on a naturally aligned octaword. A DMA device asynchronously drives an address on the DAL and then asserts AS L in order to provide an asynchronous address latch control for CVAX. The DMA device initiates a conditional invalidate operation by asserting CCTL L. The alternate quadword defined by inverting address bit <3> can also be conditionally invalidated if CCTL L is asserted twice during the DMA operation. A DMA device keeps AS L asserted throughout the conditional invalidate operation and deasserts it to end the quadword or octaword DAL transfer.

When used to initiate conditional invalidate cycles, CCTL L is level-sensitive, is sampled during P4 of every microcycle, and is internally used during P3 (synchronizer settling time is two clock phases - nominally 50ns).

#### 4.1.7.2 Prevent Data Caching -

External logic asserts CCTL L to prevent storing the result of the current CPU read cycle in the internal cache. CCTL L must be asserted coincident with the first transfer of data during a multiple transfer read operation if either transfer is to be prevented for storing a result in the cache. IO space and read lock references are never cached, irrespective of state of CCTL L.

NOTE

WHEN USED TO PREVENT DATA CACHING, CCTL L MUST BE ASSERTED SYNCHRONOUSLY WITH RESPECT TO THE CHIP'S TIMING SAMPLING POINT.

### 4.1.8 Coprocessor Control -

An optional coprocessor can be attached to CVAX in order to accelerate certain integer and floating point instructions.

4.1.8.1 Coprocessor Data Lines (CPDAT<5:0> H) -

The coprocessor Data Lines carry opcode and control information to the floating point coprocessor and return condition code and exception status to CVAX. CVAX drives these lines until it is waiting for return data. The coprocessor drives these lines when it is returning status. In both cases, CPDAT<5:0> H is sampled synchronously by the destination at the beginning of P1.

4.1.8.2 Coprocessor Status Lines (CPSTA<1:0> H) -

The Coprocessor Status Lines are sampled synchronously at the beginning of P1 and informs the CPDAT<5:0> H destination how to interpret the CPDAT data as is indicated below:

CVAX drives coprocessor lines

| CPSTA<1:0> H | Function | CPDAT<5:0> H |
|---|---|---|
| 00 | Operation encoded on CPDAT<5:0> H | <5:4> Address alignment code<br><3> = 0 no action<br><3> = 1 CVAX ready for result<br><2> = 0 no action<br><2> = 1 reserved<br><1> = 0 no action<br><1> = 1 DAL<31:0> contains floating point operand<br><0> = 0 no action<br><0> = 1 DAL<5:0> contains floating point short literal; DAL<31:6> are 0's |
| 01 | F/D floating point opcode on CPDAT<5:0> | <5:0> Floating point opcode |
| 10 | G floating point opcode on CPDAT<5:0> | <5:0> Floating point opcode |
| 11 | Integer opcode on CPDAT<5:0> | <5:0> Integer opcode |

Coprocessor drives coprocessor lines

| CPSTA<1:0> H | Function | CPDAT<5:0> H |
|---|---|---|
| 00 | Result not ready | reserved |
| 01 | Condition codes ready | <5> = 0 the result clears the PSL N bit<br><5> = 1 the result sets the PSL N bit<br><br><4> = 0 the result clears the PSL Z bit<br><4> = 1 the result sets the PSL Z bit<br><br><3> = 0 the result clears the PSL V bit<br><3> = 1 the result sets the PSL V bit<br>(integer overflow/ACB condition met)<br><br><2:0> Status<br>000 protocol error<br>001 illegal opcode<br>010 reserved operand trap<br>011 divide by zero<br>100 floating point overflow<br>101 floating point underflow<br>110 reserved<br>111 no error |
| 10 | result ready on DAL | <0> = 1 last data on DAL<br><0> = 0 not last data<br><br><1> = 1 error on POLY step occured on odd coefficient<br><1> = 0 no error occured, or error on POLY step occurred on even coefficient<br><br><5:3> reserved |
| 11 | reserved | <5:0> reserved |

4.1.9 Miscellaneous -

4.1.9.1 Power -

These inputs supply +5V to the chip. Extreme care must be taken to connect the power pins together, i.e., use very short wires or a power plane.

4.1.9.2 Ground -

These inputs supply ground to the chip. Extreme care must be taken to connect the ground pins together, i.e., use very short wires or a power plane.

4.1.9.3 Clock In (CLKA,CLKB) -

This input supplies basic clock timing to the chip. CLKA and CLKB are nominally 20 Mhz MOS level square wave signals that are 180 degrees phase shifted.

4.1.9.4 Test (TEST0 H, TEST1 H) -

TEST0 H and TEST1 H are used to control the internal CVAX test logic. The test logic enables test hardware to read internal CVAX state that is not normally observable on the external pins. The operation of the test pins is documented in section 4.5.

4.1.9.5 Bus Request (BR L) -

BR L will be asserted whenever CVAX has granted the use of the DAL (DMG L asserted) and has internally stalled because it needs to use the DAL. Note that once CVAX grants the use of the DAL, it can internally execute instructions until it needs to access DAL data (read miss, write when write buffer is full, etc.) or transfer information to and from the coprocessor.

BR L will be deasserted when DMG L is deasserted.

4.1.9.6 Console Mode (CM L) -

CM L is a time-multiplexed signal. During the first part of a cycle, CM L is asserted to indicate that CVAX is near the end of executing an REI macroinstruction. CM L will never be asserted during the first part of consecutive cycles. During the second part of a cycle, CM L is asserted to indicate that execution of a new macroinstruction is beginning. NOP is the only macroinstruction in which CM L will be asserted during the second part of consecutive cycles.

## 4.2 Bus Cycle Descriptions

The CVAX CPU chip supports the following types of bus cycles: idle; single transfer CPU read; multiple transfer CPU read; single transfer CPU write; external processor register read; external processor register write; interrupt acknowledge; DMA grant; and cache invalidate.

### 4.2.1 Idle Cycle -

An idle cycle requires two clock phases (nominally 100 ns). The DAL bus is undefined. The bus control signals are unasserted.

### 4.2.2 Single Transfer CPU Read Cycle -

In a single transfer CPU read cycle, the chip reads at most one longword from main memory or an I/O device. A single transfer CPU read cycle requires a minimum of four clock phases (nominally 200 ns) and may last longer, in increments of two clock phases (nominally 100 ns). The chip drives the physical LONGWORD address onto DAL<29:02>. DAL<31:30> are asserted to 01 to indicate single longword transfer, BM<3:0> L are asserted as required, and WR L is unasserted. The chip asserts AS L, indicating that the physical address is valid. The chip then asserts DS L, indicating that the DAL bus is free to receive incoming data and then DBE L, enabling the external interface to drive the DAL lines. The chip then samples for cycle complete once every two clock phases, starting at the next possible P1 edge.

If no error occurs, external logic responds by placing the required data on DAL<31:00> and CS/DP<3:0> L, asserting DPE L if DAL parity is to be checked, asserting CCTL L if data caching is to be prevented, asserting RDY L, and deasserting ERR L. The chip reads the data from the DAL bus and corresponding byte parity information from CS/DPE<3:0> L. Parity is checked if DPE L is asserted. If a parity error occurs, the appropriate error information is logged in MSER, the chip ignores the data on DAL<31:00>, and generates a machine check if the cycle was a demand read cycle.

If an error occurs, external logic responds by asserting ERR L with RDY L deasserted. The chip ignores the data on DAL<31:00> and generates a machine check if the cycle was a demand read cycle.

NOTE

An error will only be recognized if RDY L is deasserted for two consecutive P1 sample points. If the error response (ERR L asserted, RDY L deasserted) is detected at the first P1 sample point, but RDY L is

asserted at the second P1 sample point, the cycle will terminate according to the retry protocol detailed below.

To request a retry, external logic must assert both RDY L and ERR L. Retrying a read cycle can eliminate DAL deadlocks because CVAX guarantees that DAL arbitration occurs before the cycle is restarted (DMG L will be granted if DMR L is asserted). Note that certain request read cycles will not reissue a bus cycle if they are retried. Specifically, if the retry occurs during the second longword read of a cache allocation operation, the bus cycle is not reissued and the partially allocated cache entry is invalidated. All retries that occur on the first longword read will reissue a read bus cycle. In all read cycles, DAL arbitration occurs after the read cycle is terminated.

Irrespective of how a read cycle is terminated, the chip finishes the cycle by deasserting AS L, DBE L and DS L.

4.2.3 Multiple Transfer CPU Read Cycle -

In a multiple transfer CPU read cycle, the chip reads two (quadword) longwords from main memory. A multiple transfer CPU read cycle requires a minimum of six clock phases (nominally 300 ns) and may last longer. Each longword transfer may be independently stretched in increments of two clock phases (nominally 100 ns). Note that I/O space read references always occur as single transfer read cycles.

The chip drives the physical address of the preferred LONGWORD that is to be accessed onto DAL<29:02>. Note this address can be aligned to either longword address within the quadword block. DAL<31:30> are asserted to 10 to indicate a quadword transfer. BM<3:0> L are asserted, and WR L is unasserted. The chip asserts AS L, indicating that the physical address is valid and DBE L, indicating that the external interface can drive information on the DAL. For each of the multiple transfers, the chip asserts DS L to indicate that the DAL bus is free to receive incoming data and then samples for individual transfer complete once every two clock phases, starting at the next possible P1 edge.

If no error occurs, external logic responds on each transfer by placing the required data on DAL<31:00> and CS/DP<3:0> L, asserting DPE L if DAL parity is to be checked, asserting CCTL L if data caching is to be prevented, asserting RDY L, and deasserting ERR L. The chip reads the data from the DAL bus and corresponding byte parity information from CS/DPE<3:0> L, and deasserts DS L. Parity is checked if DPE L is asserted. If a parity error occurs, the appropriate error information is logged in MSER, the chip ignores the data on DAL<31:00>, and generates a machine check if the cycle was a demand read cycle. If data caching was not prevented, the CPU then continues on to read the additional data by reasserting DS L irrespective of a DAL parity error. The second data is ignored if a DAL parity error was detected on the first transfer. If data

caching was prevented, the cycle immediately terminates without reading the second longword of data.

If an error occurs during either transfer, external logic responds by asserting ERR L with RDY L deasserted. The chip ignores the data on DAL<31:00>, terminates the cycle without reading any additional data, and generates a machine check if the cycle was a demand read cycle. Note that only the first transfer can be a demand cycle.

NOTE

An error will only be recognized if RDY L is deasserted for two consecutive P1 sample points. If the abort response (ERR L asserted, RDY L deasserted) is detected at the first P1 sample point, but RDY L is asserted at the second P1 sample point, the cycle will terminate according to the retry protocol detailed below.

To request a retry, external logic must assert both RDY L and ERR L. Retrying a read cycle can eliminate DAL deadlocks because CVAX guarantees that DAL arbitration occurs before the cycle is restarted (DMG L will be granted if DMR L is asserted). Note that if the retry occurs during the second longword transfer, the read will not be reissued.

Irrespective of how a read cycle is terminated, the chip finishes the cycle by deasserting AS L, DBE L and DS L. Note that DBE L is not deasserted in between each transfer.

CVAX sends out an address only on the initial longword (prefered) transfer of a multiple transfer read cycle. The address associated with the second (cache fill) transfer is implied and therefore is not driven out of the chip. The implied address is generated by inverting address bit<2> of the prefered address. All references therefore stay within a quadword block. For example, if the initial longword address generated by CVAX on an quadword transfer is 0007FB36, the subsequent implied adresses is 0007FB32.

Normally, a multiple transfer cycle reads two longwords of data. However, the cycle terminates after the first data transfer if ERR L is asserted and RDY L is deasserted (memory error), or CCTL L is asserted (prevent data caching). The cycle does not terminate early if a DAL parity error is detected on the first transfer.

A summary of all possible multiple transfer cycle responses follows:

| Condition | | | | Action | |
|---|---|---|---|---|---|
| 1=asserted | | | | | |
| 0=deasserted | | | | | |
| X=don't care | | | | | |
| CCTL | RDY | ERR | DAL Parity ERROR | On First Reference | On Second Reference |
| X | 0 | 0 | X | wait for data | wait for data |
| X | 0 | 1 | X | machine check if demand<br>invalidate cache entry<br>no second reference | no machine check<br>invalidate cache entry |
| 0 | 1 | 0 | 0 | no machine check<br>update cache<br>proceed to second reference | no machine check<br>update cache |
| 1 | 1 | 0 | 0 | no machine check<br>no cache change<br>no second reference | no machine check<br>update cache |
| 0 | 1 | 0 | 1 | machine check if demand<br>invalidate cache entry<br>log error in MSER<br>proceed to second reference | no machine check<br>invalidate cache entry<br>log error in MSER |
| 1 | 1 | 0 | 1 | machine check if demand<br>invalidate cache entry<br>log error in MSER<br>no second reference | no machine check<br>invalidate cache entry<br>log error in MSER |
| X | 1 | 1 | X | no machine check<br>no cache change<br>no second reference - retry | no machine check<br>invalidate cache entry<br>no retry |

4.2.4 CPU Write Cycle -

In a CPU write cycle, the chip writes information to main memory or I/O devices. A CPU write cycle requires a minimum of four clock phases (nominally 200 ns) and may last longer, in increments of two clock phases (nominally 100 ns). The chip drives the physical LONGWORD address onto DAL<29:02>. BM<3:0> L are asserted as required, DAL<31:30> are driven as 01 to indicate longword transfer, and WR L is asserted. The chip asserts AS L, indicating that the physical address is valid and then DBE L, indicating that the write data can be driven on an external bus. The chip then drives the output data onto DAL<31:00>, drives the byte parity

information on CS/DP<3:0> L, asserts DPE L indicating that valid parity information is available, and asserts DS L, indicating the Data bus contains valid data. The chip then samples for cycle complete once every two clock phases, starting at the next possible P1.

If no error occurs, external logic responds by reading the data from the DAL bus, asserting RDY L and deasserting ERR L. If an error occurs, external logic responds by asserting ERR L with RDY L deasserted. Aborting a write cycle will generate a machine check. Note that a DAL parity error will be reported back to the CPU by asserting ERR L and deasserting RDY L.

NOTE

An error will only be recognized if RDY L is deasserted for two consecutive P1 sample points. If the error response (ERR L asserted, RDY L deasserted) is detected at the first P1 sample point, but RDY L is asserted at the second P1 sample point, the cycle will terminate according to the retry protocol detailed below.

To request a retry, external logic must assert both RDY L and ERR L. DAL arbitration occurs after the write operation is terminated.

Irrespective of how a write cycle is terminated, the chip finishes the cycle by deasserting AS L, DBE L, and DS L.

### 4.2.5 External Processor Register Read Cycle -

An external processor register read cycle is initiated whenever a category 3 processor register (see section 2.8) is read using a MFPR instruction. This cycle requires a minimum of four clock phases (nominally 200 ns) and may last longer, in increments of two clock phases (nominally 100 ns). The chip drives the processor register number onto DAL<7:2>. DAL<31:30> are asserted to 01 to indicate longword transfer, BM<3:0> L are all asserted, and WR L is unasserted. The chip asserts AS L, indicating that the register number is valid and then DBE L, indicating that read data can be driven on the DAL. The chip then asserts DS L, indicating that the DAL bus is free to receive incoming data. The chip then samples for cycle complete once every two clock phases, at the next possible P1.

If the processor register is implemented, external logic responds by placing the required data on DAL<31:00>, asserting RDY L, and deasserting ERR L. The chip reads the data from the DAL bus. If the processor register is not implemented, external logic responds by asserting ERR L with RDY L deasserted. The chip ignores the data on DAL<31:00> and

internally forces the result to zero. No parity checking is done during external processor read cycles.

NOTE

The not implemented response will only be recognized if RDY L is deasserted for two consecutive P1 sample points. If this response (ERR L asserted, RDY L deasserted) is detected at the first P1 sample point, but RDY L is asserted at the second P1 sample point, the cycle will terminate according to the retry protocol detailed below.

To request a retry, external logic must assert both RDY L and ERR L. DAL arbitration occurs after the initial read cycle is terminated.

Irrespective of how an external processor read cycle is terminated, the chip finishes the cycle by deasserting AS L, DBE L and DS L.

4.2.6 External Processor Register Write Cycle -

An external processor register write cycle is initiated whenever a category 3 processor register (see section 2.8) is written using a MTPR instruction. This cycle requires a minimum of four clock phases (nominally 200 ns) and may last longer, in increments of two clock phases (nominally 100 ns). The chip drives the processor register number onto DAL<7:2>. BM<3:0> L are all asserted, DAL<31:30> are driven as 01 to indicate longword transfer, and WR L is asserted. The chip asserts AS L, indicating that the register number is valid, and then asserts DBE L, indicating that the write data can be driven on an external bus. The chip then drives the write data onto DAL<31:00> and asserts DS L, indicating the DAL contains valid data. The chip then samples for cycle complete once every two clock phases, starting at the next possible P1.

If the processor register is implemented, external logic responds by reading the data from the DAL bus, asserting RDY L, and deasserting ERR L. If the processor register is not implemented, external logic either responds as is indicated above or asserts ERR L and deasserts RDY L. Both responses have the same effect; no special action is taken.

NOTE

The not implemented response (ERR L asserted, RDY L deasserted) will take no

special action only if RDY L is deasserted for two consecutive P1 sample points. If this response is detected at the first P1 sample point, but RDY L is asserted at the second P1 sample point, the cycle will terminate according to the retry protocol detailed below.

To request a retry, external logic must assert both RDY L and ERR L. DAL arbitration occurs after the initial write cycle is terminated.

Irrespective of how an external processor write cycle is terminated, the chip finishes the cycle by deasserting AS L, DBE L and DS L.

### 4.2.7 Interrupt Acknowledge Cycle -

An interrupt acknowledge cycle has the same structure as a single transfer CPU read cycle. DAL<6:2> is driven out with the IPL level of the interrupt being acknowledged (IRQ<3> L is IPL 17, IRQ<2> L is IPL 16, IRQ<1> L is IPL 15, IRQ<0> L is IPL 14) and DAL<31:7,1:0> are driven with zeros. The data read in is used to generate the vector and new IPL for the interrupt sequence. Bits <09:02> of the incoming data are used to create the vector offset within the System Control Block. The new PSL priority level is determined by either the external interrupt request level that caused the interrupt or by bit <0> of the value supplied by external hardware. If bit<0> is 0, the new IPL level is determined by the interrupt request level being serviced. IRQ<3> sets the IPL to 17 (hex); IRQ<2>, 16 (hex); IRQ<1>, 15 (hex); and IRQ<0>, 14 (hex). If bit<0> of the value suppled by external hardware is 1, then the new IPL is forced to 17 (hex). Bits <31:10,01> of the incoming data are ignored. Assertion of ERR L in proper combination with RDY L causes the bus cycle to be retried or aborted. An abort causes an interrupt through SCB vector 0.

### 4.2.8 DMA Grant Cycle -

The chip can relinquish its control of the DAL bus and related control signals upon request from a DMA device or another CPU. The external device requests control of the bus by asserting DMR L. At the conclusion of the current bus cycle, the chip responds by tristating DAL<31:00>, AS L, DS L, WR L, and DBE L, BM<3:0> L and DP/CS<3:0> L. The external device may now use the DAL bus to transfer data. To return control of the DAL bus to the CPU, the external device and deasserts DMR L. The chip responds by deasserting DMG L and starting the next bus cycle.

### 4.2.9 Cache Invalidate Cycles -

External logic initiates a conditional invalidate operation to detect and invalidate stale data that is stored in the cache. A conditional invalidate cycle uses a minimum of six clock phases (nominally 300 ns).

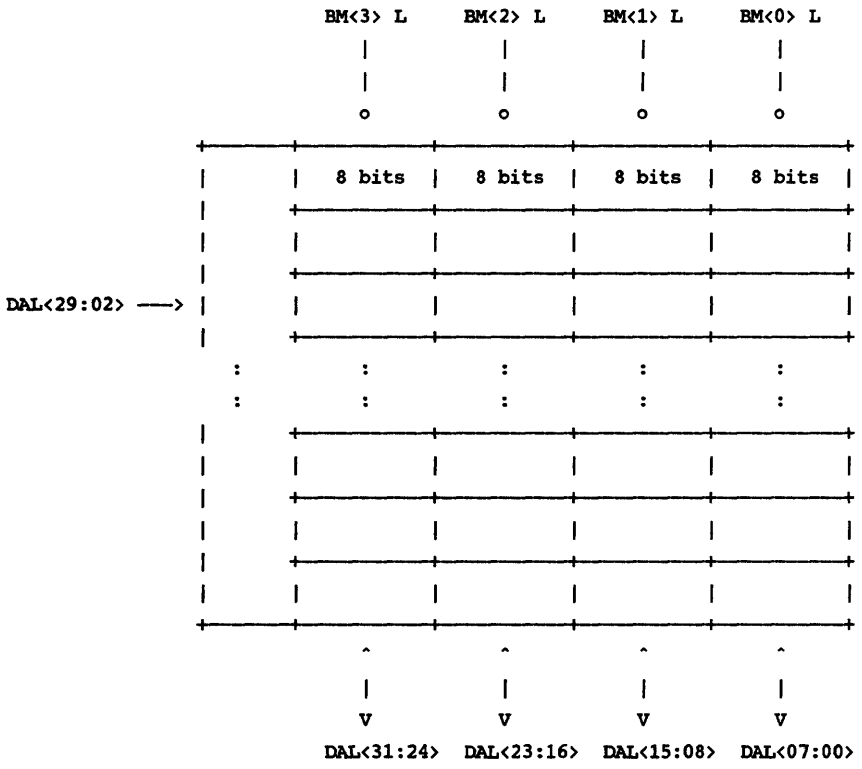
Once DMG L is asserted by the CPU, external logic asynchronously drives the physical address onto DAL<31:0>, asynchronously asserts AS L to latch the address into the CPU, and then asynchronously asserts CCTL L to start a conditional invalidate cycle. The CPU then invalidates the quadword cache entry selected by the DMA address if the location is stored in the cache. External logic deasserts CCTL L and then optionally reasserts CCTL L to conditionally invalidate the alternate quadword formed by inverting address bit <3>. This allows external logic to detect and invalidate stale data stored in any naturally aligned octaword. The cycle ends when external logic deasserts both AS L and CCTL L.

The CPU detects and invalidates quadword stale data in six clock phases. Therefore, the maximum cache invalidate rate can not exceed 8 byte/six clock phases (nominally 26.6Mb/sec).

## 4.3 Memory Access Protocol

The 28-bit address provided by the CVAX CPU chip on DAL<29:02> is a LONGWORD address which uniquely identifies one of up to 268,435,456 32-bit memory locations. The chip provides four byte masks, BM<3:0> L, to facilitate byte accesses within 32-bit memory locations. The chip imposes no restrictions on data alignment. Any data item, regardless of size, may be placed starting at any memory address (except for the aligned operands of ADAWI and the interlocked queue instructions).

Memory is viewed as four parallel eight-bit banks, each of which receives the longword address DAL<29:02> in parallel. Each bank reads or writes one byte of the data bus (DAL<31:00>), provided that its byte mask signal is asserted. This is illustrated in the following diagram:




Any single transfer CPU read or CPU write falls into one of the following categories: byte access, word access within a longword, word access across longwords, aligned longword access, unaligned longword access. (Quadword data is accesses with two successive longword accesses, with no optimization.) Byte accesses, word accesses within a longword, and aligned longword accesses require one bus cycle. Word accesses which cross a longword boundary, and unaligned longword accesses, require two bus cycles. The exact signal usage is shown in the following chart:

Single Transfer

| Access Type | Cycle | DAL<31:30> | DAL<29:02> | BM<3> L | BM<2> L | BM<1> L | BM<0> L |
|---|---|---|---|---|---|---|---|
| byte | 1 | 01 | A<29:02> | if A<1:0>=11 | if A<1:0>=10 | if A<1:0>=01 | if A<1:0>=00 |
| word within longword | 1 | 01 | A<29:02> [A<1:0> ne 11] | if A<1:0>=10 | if A<1:0>=10 or A<1:0>=01 | if A<1:0>=0X | if A<1:0>=00 |
| aligned longword | 1 | 01 | A<29:02> [A<1:0> = 00] | L | L | L | L |
| word across longwords | 1 | 01 | A<29:02> | L | H | H | H |
| | 2 | | A+4<29:02> [A<1:0> = 11] | H | H | H | L |
| unaligned longword | 1 | 01 | A<29:02> | L | if A<1:0>=01 or A<1:0>=10 | if A<1:0>=01 | H |
| | 2 | 01 | A+4<29:02> [A<1:0> ne 00] | H | if A<1:0>=11 | if A<1:0>=10 or A<1:0>=11 | L |

Accesses requiring more than one bus cycle are performed sequentially, with no computation in between. However, DMA grants may occur between the bus cycles of an unaligned reference.

All multiple transfer CPU read cycles read exactly two aligned longwords. Therefore, BM<3:0> L are asserted for each data transfer, DAL<31:30> H are driven as 10 and DAL<1:0> are not specified. DMA grants can not occur between the individual transfers.

### 4.3.1 I-stream Prefetching -

CVAX contains a twelve byte I-stream prefetch buffer organized as three aligned longwords. A request I-stream prefetch cycle will be generated only when an aligned longword is empty. Up to six bytes at a time can be retired from the prefetch buffer.

## 4.4 Coprocessor Protocols

### 4.4.1 Passing Opcode Information To The Coprocessor -

Opcode information must be passed to the coprocessor whenever it is going to execute or accelerate any of the CVAX instructions. All floating point and some integer instructions pass opcode information when coprocessor activity is desired (assuming the coprocessor is present). In either case, only the six lower order opcode bits are passed to the coprocessor. CVAX drives an f/d_floating point opcode on CPDAT when CPSTA<1:0> H=01; a g_floating point opcode when CPSTA<1:0>=10; and an integer opcode when CPSTA<1:0> H=11. Note that the integer instructions that are accelerated

by the coprocessor are DIVL2, DIVL3, MULL2, MULL3, and EMUL.

### 4.4.2 Passing Operands To The Coprocessor -

Operands that are to be passed to the coprocessor can come from three sources: memory, the internal cache, or the general purpose registers. CVAX drives CPSTA<1:0> H=00 and CPDAT<1> H=1 when the next coprocessor operand data is on DAL<31:0>. If the source of the operand is either memory or the internal cache, CPDAT<5:4> are driven with the two low order address bits of the reference; otherwise the source is the general purpose registers and CPDAT<5:4> are driven as 00. The coprocessor must align all unaligned data. If the data is coming from memory (AS L is asserted), the coprocessor reads the DALs according to the full memory read protocol (RDY L and/or ERR L asserted); otherwise, the data is coming from CVAX (internal cache or the general purpose registers), is driven on the DALs at P3 and is sampled by the coprocessor at the next P1. If the source of the operand is either memory or the internal cache and a parity error is detected by CVAX, the chip aborts the coprocessor operation, and never signals the coprocessor for the current result. The coprocessor is reset when CVAX sends a new coprocessor opcode.

In summary, CPDAT<5:0> H are encoded as follows while operands are being passed to the coprocessor:

| CPDAT<5:0> | |
|---|---|
| <5:4> | Address alignment code |
| <3:2> | not specified |
| <1> = 0 | no action |
| = 1 | DAL<31:0> is floating point operand |
| <0> = 0 | no action |
| = 1 | DAL<5:0> is short literal; DAL<31:6> are 0's. |

### 4.4.3 Passing Results Back From The Coprocessor -

CVAX informs the coprocessor when it is ready for a result by asserting CPSTA<1:0> H=00 and CPDAT<3> H=1 at a P3 edge. CVAX then gives up ownership of the CPDAT, CPSTA, and DAL by tri-stating these lines at the next P2 edge; the coprocessor gains ownership of CPDAT and CPSTA by driving them at the next P3 edge. The DAL remains tri-stated until either the coprocessor returns a result or CVAX regains CPDAT and CPSTA ownership. CVAX then waits for the coprocessor result. Note that during a POLY STEP operation, CVAX will assert CPDAT<2> H at the same time it is signaling a CPDAT/CPSTA ownership transfer when a VAX POLY instruction is going to be suspended.

While waiting, CVAX can grant DMG L on a P4 edge. If no DMG L is granted, CVAX continuously samples the coprocessor CPSTA and CPDAT lines on each P1 edge; if granted, the CPSTA and CPDAT lines are ignored. The coprocessor asserts CPSTA<1:0> H=00 at a P3 edge to indicate that the result is not ready; and CPSTA<1:0> H=01, to indicate that they are ready. If the coprocessor indicates that the condition codes are ready (P3) at the same time that CVAX grants DMG L (P4), the coprocessor repeats the response until DMG L is deasserted.

Once CVAX detects that the condition code results are ready, the CPDAT lines are used to determine the response, and DMG L will not be granted until the end of the operation. CPDAT<5:0> H are encoded as follow:

CPDAT<5> = 0 the result clears the PSL N bit.
= 1 the result sets the PSL N bit.

CPDAT<4> = 0 the result clears the PSL Z bit.
= 1 the result sets the PSL Z bit.

CPDAT<3> = 0 the result clears the PSL V bit
= 1 the result sets the PSL V bit
(integer overflow/ACB condition met)

| CPDAT<2:0> | Status | Data Transfer |
|---|---|---|
| 000 | protocol error | aborted |
| 001 | illegal opcode | aborted |
| 010 | reserved operand trap | aborted |
| 011 | divide by zero | aborted |
| 100 | floating point overflow | aborted |
| 101 | floating point underflow | continue |
| 110 | reserved | |
| 111 | no error | continue |

If CPDAT<2:0> H indicates protocol error, illegal opcode, reserved operand trap, divide by zero, or floating point overflow, no data is transfered and the coprocessor gives up CPDAT and CPSTA ownership by tri-stating these lines at the next P3 edge; otherwise, at the next P3 edge, the coprocessor drives the first (possible only) result on the DAL<31:0> H, and CPSTA<1:0>=10. A second transfer is used if the coprocessor is returning a double precision result. The coprocessor drives the second (always final) longword on DAL<31:0> and CPSTA<1:0>=10 on the following P3 edge. In parallel with each data transfer, the coprocessor drives CPDAT<0>=1 to indicate the final transfer; CPDAT<0>=0, to indicate the first of two transfers; CPDAT<1>=1 to indicate that an error occurred on the odd coefficient of a POLY step instruction; and CPDAT<1>=0 to indicate that no error occurred or that the error occurred on the even coefficient of a POLY step. After the final transfer, the coprocessor relinquishes ownership of CPDAT, CPSTA, and DAL by tristating these lines at the next P2 edge.

A single unaligned longword is transfered for a single precison result (F), and two unaligned longwords are transfered for a double precision

result (D or G). CVAX aligns the data and performs the final transfer if the ultimate destination of the coprocessor data is memory.

#### 4.4.4 Coprocessor Reset -

There are severals conditions that are detected in CVAX which reset the coprocessor.

- RESET L is asserted
- coprocessor operand generates a reserved addressing mode fault
- coprocessor operand generates an address translation fault
- coprocessor operand access causes a machine check abort
- coprocessor operand access causes a cache or DAL parity error
- coprocessor operation must be suspended, i.e., interrupt during POLY

The coprocessor resets whenever RESET L is asserted or CVAX issues a new coprocessor opcode.

### 4.5 Test Logic

The test logic enables test hardware to read internal CVAX state that is not normally observable on the external pins. It also allows a tester to redefine fourteen pins in order to gain control of internal logic. Some knowledge of the internal organization of the CVAX CPU is necessary in order to understand the significance of the test logic output. It is beyond the scope of this specification to describe the internal organization of the chip. For further information, the applicable documents listed in section 1.2 should be consulted.

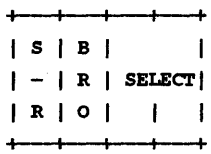
The Test Logic can be conceptually divided into logic that aids observability and logic that controls the test operation.

#### 4.5.1 Observablitiy Logic -

The observability logic consists of three parallel-in serial-out test registers, a main data reducer register, logic to parallel load internal signals into these registers, and logic that allows the registers to be externally observed. The contents of the test registers can be observed using either scan mode or reduce mode. In scan mode, the selected test register is simply serially shifted out to the TEST1 H pin. In reduce mode, the test registers become linear feedback shift registers and the selected output is serially shifted out to the TEST1 H pin. The three test registers feed a fourth reducer register known as the main reducer. The main reducer is a linear feedback shift register output can also be serially shifted out to the TEST1 H pin.

4.5.2 Control Logic -

The control logic consists of the configuration register. This register selects which test registers or main reduce drives TEST1 H, the mode of operation for the test registers (scan or reduce), and if a broadcast should be forced.




| Bit | Function | |
|---|---|---|
| S-R | Scan/Reduce select | 0=reduce<br>1=scan |
| BRO | force broadcast | 0=no broadcast<br>1=force broadcast |
| SELECT | select which register to observe | 00=main reducer<br>01=test register #1<br>10=test register #2<br>11=test register #3 |

When TEST0 H is deasserted, configuration register is reset to 0000.

4.5.3 Normal State -

The Test Logic is in the 'normal' state when TEST0 H is deasserted. No pins are redefined and TEST1 H is driving the output of the main reducer. The configuration latch is cleared.

4.5.4 Test State -

4.5.4.1 Internal MAB -

The Test Logic is in the 'test' state when TEST0 H is asserted. HALT L is now redefined to EXTERNAL H which controls whether the internal control store microaddress bus (MAB) is driven by external pins or the normal internal paths; and PWRFL L is redefined to LOAD H which is used to control the parallel loading of the test registers and main reducer register. When EXTERNAL H (HALT L) is deasserted, MAB is controlled by the normal internal paths.

4.5.4.2 External MAB -

When TEST0 H and EXTERNAL H (HALT L) are both asserted, IRQ<1:3> L, CPDAT<0:5> H and CPSTA<0:1> H are redefined to EXTERNAL MAB<10:0> H, respectively; and, IRQ<0> L is redefined to CONFIGURE H. The MAB<10:0> is driven by EXTERNAL MAB<10:0> H (IRQ<1:3> L, CPDAT<5:0> H and CPSTA<0:1> H) which are latched at the beginning of each cycle. If CONFIGURE H (IRQ<0> L) is asserted, IRQ<1:3> L and CPDAT<0> H are loaded into the configuration register in the middle of each cycle.

4.5.4.3 Force Broadcast -

If the broadcast bit in the configuration latch is set, DAL<31:0> H are driven with the contents of the internal W bus on every cycle.

4.5.5 Test Registers -

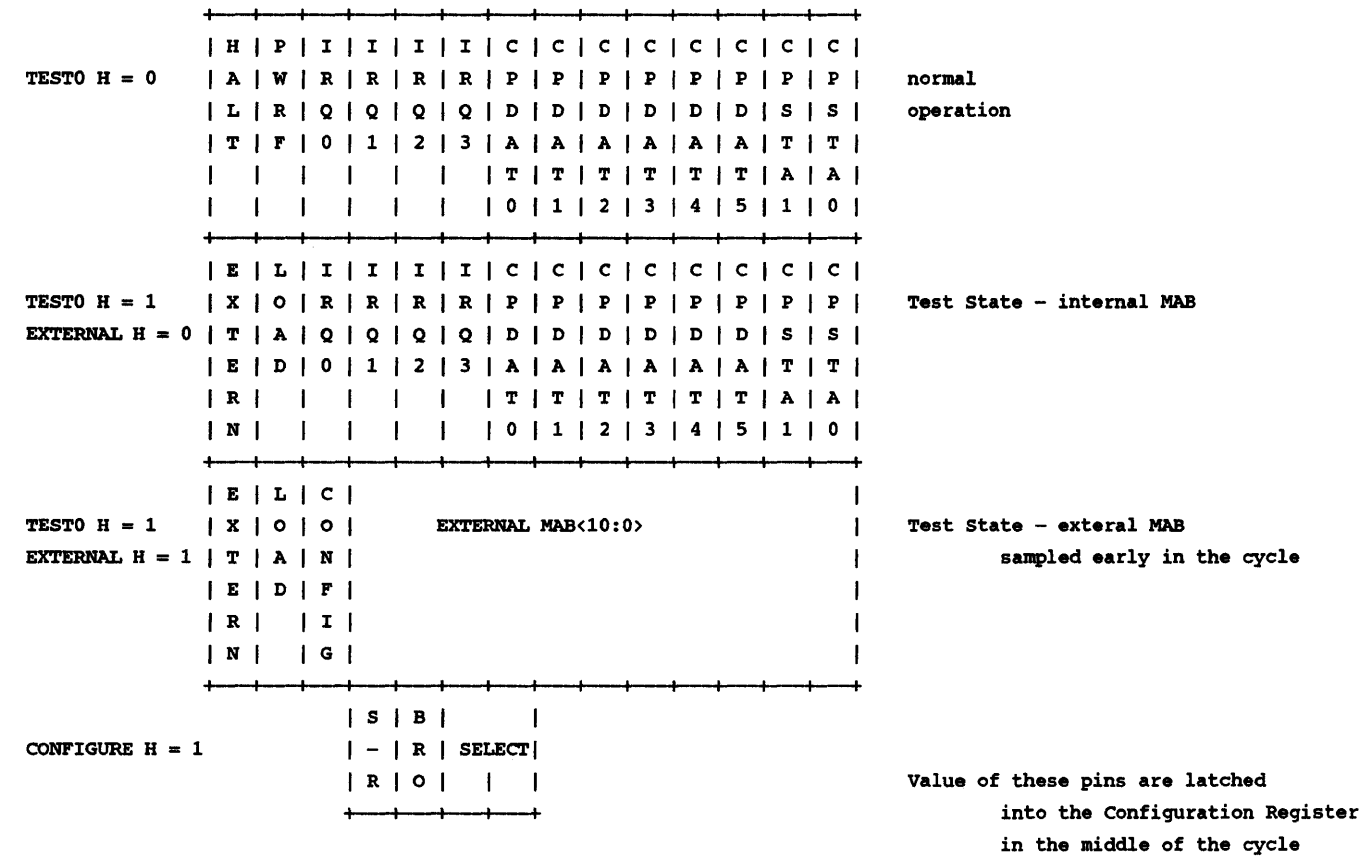
The three test registers capture the following information:

| Test Register | Information |
|---|---|
| Register #1 | micro-instruction bus (MIB<40:0>)<br>micro-test bus (UTEST<2:0) |
| Register #2 | MAB_H<10:0> |
| Register #3 | Instruction box multiplex output (IMUX<28:0>) |

4.5.6 Main Reducer -

The main reducer is fed by the outputs of the three test registers. This register only operates in the reduce mode.

4.5.7 Test Control Pins Allocation -




| Signal | Function | |
|---|---|---|
| EXTERNAL H | External/Internal MAB | 0 = internal<br>1 = external |
| LOAD H | parallel load scan shift register with scan data. | 0 = no load<br>1 = load |
| CONFIGURE H | parallel load configuration latch from EXT_MAB<10:7>. | 0 = do not load Configuration Register<br>1 = load Configuration Register |

# 5.0 DC CHARACTERISTICS

## 5.1 Absolute Maximum Ratings

| | |
|---|---|
| Storage Temperature Range | -55 C to +125 C |
| Active Temperature Range | 0 C to +125 C |
| Supply Voltage | -0.5 V to +tbs V |
| Input or Output Voltage Applied | -1 V to tbs V |

## 5.2 Electrical Characteristics

| | |
|---|---|
| Specified Temperature Range | 0 C to +70 C |
| Specified Supply Voltage Range | +4.50 V to +5.50 V |
| Test Conditions | Temperature = +70 C |
| | Vss = 0 V |
| | Vcc = +4.75 V (except as noted) |

| Symbol | Parameter | Min | Max | Units | Test Condition |
|---|---|---|---|---|---|
| Vih | High level input voltage (TTL) | 2.0 | | V | |
| Vil | Low level input voltage (TTL) | | 0.8 | V | |
| Vihm | High level input voltage (MOS) | 70% Vdd | | V | |
| Vilm | Low level input voltage (MOS) | | 30% Vdd | V | |
| Voh | High level output voltage | 2.4 | | V | Ioh = - 400 uA |
| Vol | Low level output voltage | | 0.4 | V | Iol = 2.0 mA |
| Iil | Input leakage current | -10 | 10 | uA | 0 < Vin < 5.25 V |
| Iol | Output leakage current | -10 | 10 | uA | 0 < Vin < 5.25 V |
| Icc | Active supply current | | tbs | mA | Iout = 0, Ta = 0 C |
| Cin | Input capacitance | | tbs | pF | |
| Cout | Output capacitance | | tbs | pF | |

## 5.3 Signal Summary

| Signal Name | Signal Type | Pin Number | Applicable Tests | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | Vih | Vil | Voh | Vol | Iil | Iol | Vihm | Vilm |
| DAL<31> H | IO | | X | X | X | X | X | X | | |
| DAL<30> H | IO | | X | X | X | X | X | X | | |
| DAL<29> H | IO | | X | X | X | X | X | X | | |
| DAL<28> H | IO | | X | X | X | X | X | X | | |
| DAL<27> H | IO | | X | X | X | X | X | X | | |
| DAL<26> H | IO | | X | X | X | X | X | X | | |
| DAL<25> H | IO | | X | X | X | X | X | X | | |
| DAL<24> H | IO | | X | X | X | X | X | X | | |
| DAL<23> H | IO | | X | X | X | X | X | X | | |
| DAL<22> H | IO | | X | X | X | X | X | X | | |
| DAL<21> H | IO | | X | X | X | X | X | X | | |
| DAL<20> H | IO | | X | X | X | X | X | X | | |
| DAL<19> H | IO | | X | X | X | X | X | X | | |
| DAL<18> H | IO | | X | X | X | X | X | X | | |
| DAL<17> H | IO | | X | X | X | X | X | X | | |
| DAL<16> H | IO | | X | X | X | X | X | X | | |
| DAL<15> H | IO | | X | X | X | X | X | X | | |
| DAL<14> H | IO | | X | X | X | X | X | X | | |
| DAL<13> H | IO | | X | X | X | X | X | X | | |
| DAL<12> H | IO | | X | X | X | X | X | X | | |
| DAL<11> H | IO | | X | X | X | X | X | X | | |
| DAL<10> H | IO | | X | X | X | X | X | X | | |
| DAL<09> H | IO | | X | X | X | X | X | X | | |
| DAL<08> H | IO | | X | X | X | X | X | X | | |
| DAL<07> H | IO | | X | X | X | X | X | X | | |
| DAL<06> H | IO | | X | X | X | X | X | X | | |
| DAL<05> H | IO | | X | X | X | X | X | X | | |
| DAL<04> H | IO | | X | X | X | X | X | X | | |

DC CHARACTERISTICS

| Signal Name | Signal Type | Pin Number | Applicable Tests | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | Vih | Vil | Voh | Vol | Iil | Iol | Vihm | Vilm |
| DAL<03> H | IO | | X | X | X | X | X | X | | |
| DAL<02> H | IO | | X | X | X | X | X | X | | |
| DAL<01> H | IO | | X | X | X | X | X | X | | |
| DAL<00> H | IO | | X | X | X | X | X | X | | |
| CS/DP<3> H | IO | | X | X | X | X | X | X | | |
| CS/DP<2> H | IO | | X | X | X | X | X | X | | |
| CS/DP<1> H | IO | | X | X | X | X | X | X | | |
| CS/DP<0> H | IO | | X | X | X | X | X | X | | |
| DPE L | IO | | X | X | X | X | X | X | | |
| AS L | IO | | X | X | X | X | X | X | | |
| CPDAT<5> H | IO | | X | X | X | X | X | X | | |
| CPDAT<4> H | IO | | X | X | X | X | X | X | | |
| CPDAT<3> H | IO | | X | X | X | X | X | X | | |
| CPDAT<2> H | IO | | X | X | X | X | X | X | | |
| CPDAT<1> H | IO | | X | X | X | X | X | X | | |
| CPDAT<0> H | IO | | X | X | X | X | X | X | | |
| CPSTA<1> H | IO | | X | X | X | X | X | X | | |
| CPSTA<0> H | IO | | X | X | X | X | X | X | | |
| DS L | O | | | | X | X | | X | | |
| BM<3> H | O | | | | X | X | | X | | |
| BM<2> H | O | | | | X | X | | X | | |
| BM<1> H | O | | | | X | X | | X | | |
| BM<0> H | O | | | | X | X | | X | | |
| WR L | O | | | | X | X | | X | | |
| DBE L | O | | | | X | X | | X | | |
| DMG L | O | | | | X | X | | X | | |
| BR L | O | | | | X | X | | X | | |
| CM L | O | | | | X | X | | X | | |
| RDY L | I | | X | X | | | X | | | |
| ERR L | I | | X | X | | | X | | | |

| Signal Name | Signal Type | Pin Number | Applicable Tests | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | Vih | Vil | Voh | Vol | Iil | Iol | Vihm | Vilm |
| RESET L | I | | X | X | | | X | | | |
| HALT L | I | | X | X | | | X | | | |
| IRQ<3> L | I | | X | X | | | X | | | |
| IRQ<2> L | I | | X | X | | | X | | | |
| IRQ<1> L | I | | X | X | | | X | | | |
| IRQ<0> L | I | | X | X | | | X | | | |
| RWRFL L | I | | X | X | | | X | | | |
| CRD L | I | | X | X | | | X | | | |
| INTTIM L | I | | X | X | | | X | | | |
| DMR L | I | | X | X | | | X | | | |
| CCTL L | I | | X | X | | | X | | | |
| MEMERR L | I | | X | X | | | X | | | |
| CLKA L | I | | | | | | X | | X | X |
| CLKB L | I | | | | | | X | | X | X |
| TEST0 L | I | | X | X | | | X | | | |
| TEST1 L | O | | | | X | X | | X | | |

## 6.0 AC CHARACTERISTICS

Test Conditions: (except as noted)

| | | |
|---|---|---|
| Temperature | = | +70 C |
| Vss | = | 0V |
| Vdd | = | +4.75V |
| Cload | = | 130pF (except CPDAT and CPSTA) |

### 6.1 Input Requirements

| Symbol | Parameter | Min | Max | Units | Remarks |
|---|---|---|---|---|---|
| Tclke | External clock edge rate | 0 | 10 | nS | |
| Tcycle | External clock cycle | 50 | TBS | nS | |
| Tclkh | External clock high | 5 | 25 | nS | |
| Tclkl | External clock low | 5 | 25 | nS | |
| Tclkdly | CLKA to CLKB delay | Tcycle/2 - 2 | Tcycle/2 + 2 | nS | |
| Tresetw | Reset input width | TBS | TBS | nS | |
| Tresets | Reset input setup prior to P1 | 20 | Tcycle/2-10 | nS | |
| Tsyns | Synchronizer input setup | TBS | | nS | |
| Tsynh | Synchronizer input hold | TBS | | nS | |
| Tds | DAL setup | 25 | | nS | |
| Tdps | parity setup | 20 | | nS | |
| Tdh | DAL hold | 5 | | nS | |
| Tsws | Sample window setup | 15 | | nS | |

| | | | | |
|---|---|---|---|---|
| Tswh | Sample window hold | 5 | nS | |
| Tcps | Coprocessor line setup | TBS | nS | Cload = 50pF |
| Tcph | Coprocessor line hold | TBS | | Cload = 50pF |
| Tcctlw | CCTL width during cache invalidates | 6*Tcycle | nS | |
| Tcctlh | CCTL L high between quadword invalidates | 20 | ns | |
| Tcctladrs | AS L set up during cache invalidates | 20 | ns | |
| Tash | AS L hold during cache invalidates | 4*Tcycle | nS | |
| Tasadrs | DAL setup during cache invalidates | 20 | nS | |
| Tasadrh | DAL hold during cache invalidates | 20 | nS | |

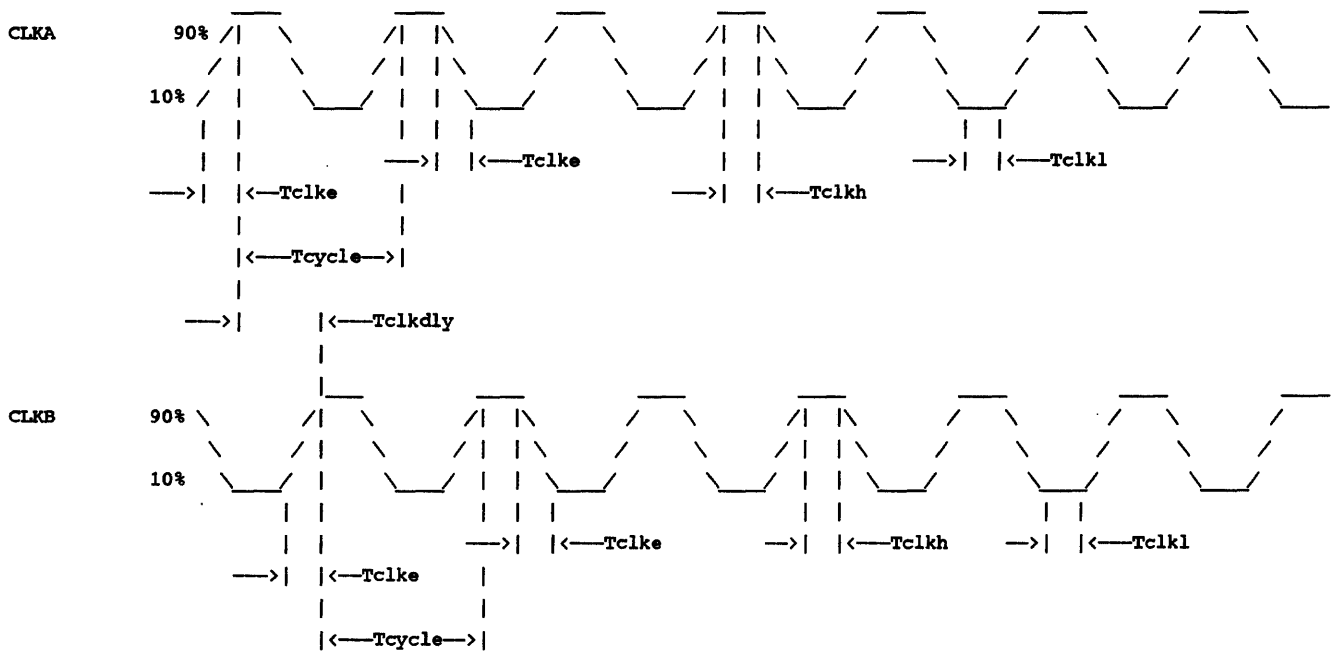
## 6.2 Output Responses

| Symbol | Parameter | Min | Max | Units | Remarks |
|---|---|---|---|---|---|
| Tsd | General strobe assertion delay | 0 | 20 | nS | |
| Tsid | strobe deassertion delay | 0 | 20 | nS | |
| Tasd | AS strobe assertion delay | 0 | 15 | nS | |
| Tasid | AS strobe deassertion delay | 0 | 15 | nS | |
| Tdsd | DS strobe assertion delay | 0 | 15 | nS | |
| Tdsid | DS strobe deassertion delay | 0 | 15 | nS | |
| Tdmgsd | DMG strobe assertion | | | | |

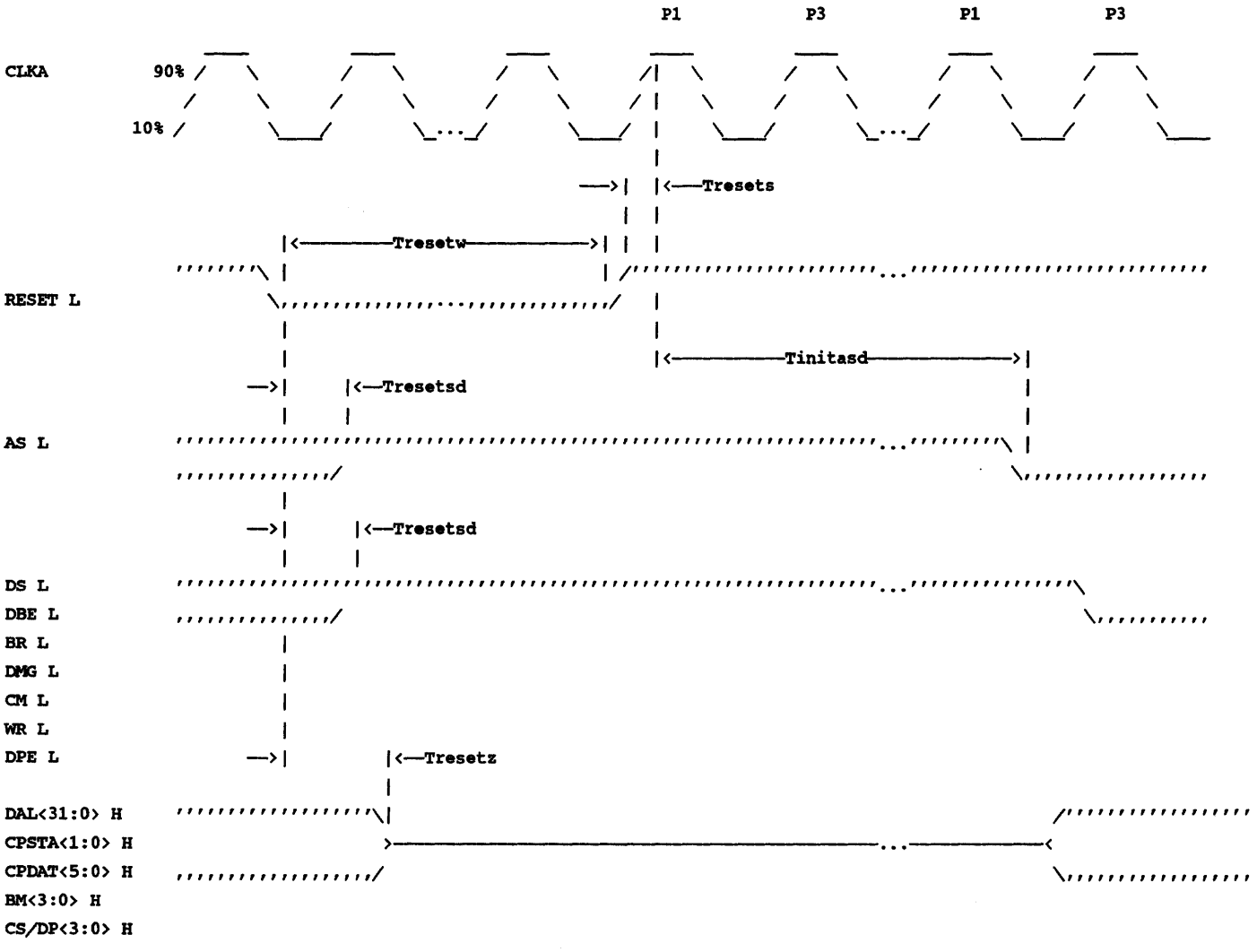
| | | | | | |
|---|---|---|---|---|---|
| | delay | 0 | 15 | nS | |
| Tdmgsid | DMG strobe deassertion delay | 0 | 15 | nS | |
| Tshlz | Strobe tri-state delay | 0 | 20 | nS | |
| Tszhl | Strobe active drive delay | 0 | 20 | nS | |
| Tdalhlz | DAL tri-state delay | 0 | 20 | nS | |
| Tdalzhl | DAL active drive delay | 0 | 20 | nS | |
| Tdald | DAL drive | 0 | 20 | nS | |
| Tdalh | DAL hold | 5 | | nS | |
| Tparityd | DP drive | 0 | 35 | nS | |
| Tparityh | DP hold | 0 | | nS | |
| Tbmh | BM and WR hold | 0 | | ns | |
| Tcpd | Coprocessor line drive | 0 | 15 | nS | Cload is 50pF |
| Tcpdh | Coprocessor line hold | 0 | | nS | Cload is 50pF |
| Tcphlz | Coprocessor tri-state delay | 0 | 15 | nS | |
| Tinitasd | First assertion of AS L after RESET L | TBS*Tcycle | TBS*Tcycle+Tasd | ns | |
| Tcmd | CM L drive | 0 | 20 | ns | |
| Tcmh | CM L hold | 0 | | ns | |
| Tresetd | Strobe inactive delay from reset | 0 | TBS | ns | |
| Tresetz | Bus tristate time from reset | 0 | TBS | ns | |

# 7.0 TIMING DIAGRAMS

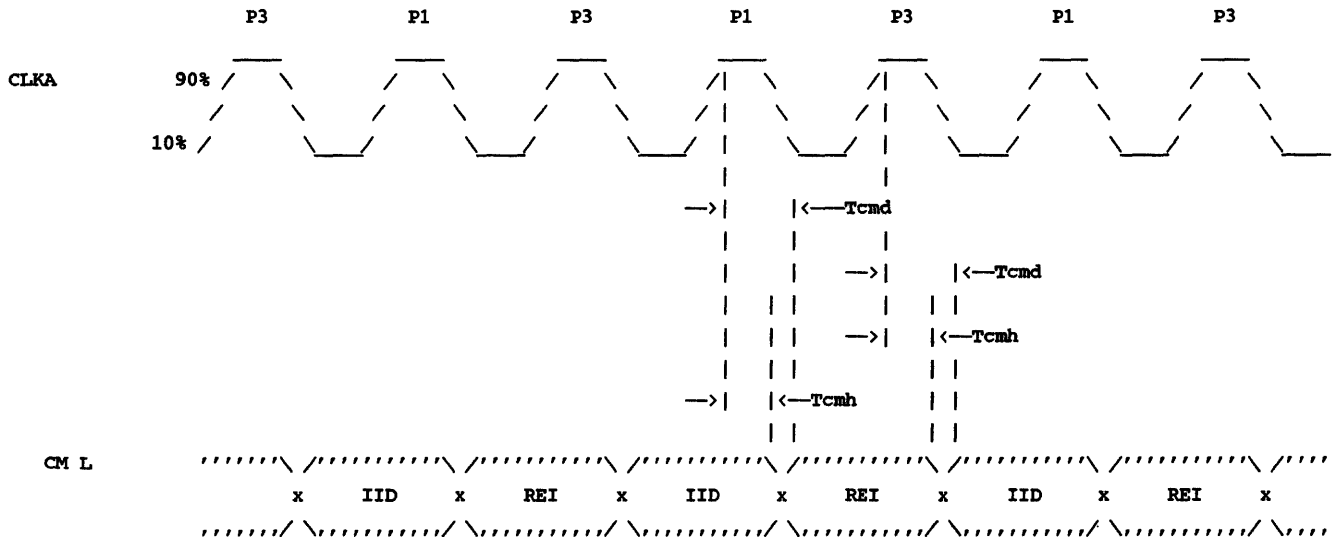
## 7.1 Clock Timing Requirements



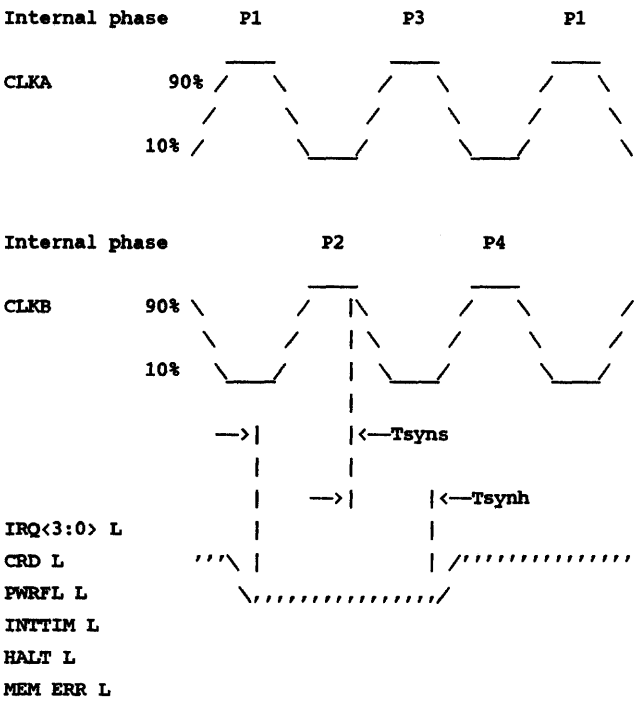
## 7.2 Initialization



## 7.3 CM L Timing

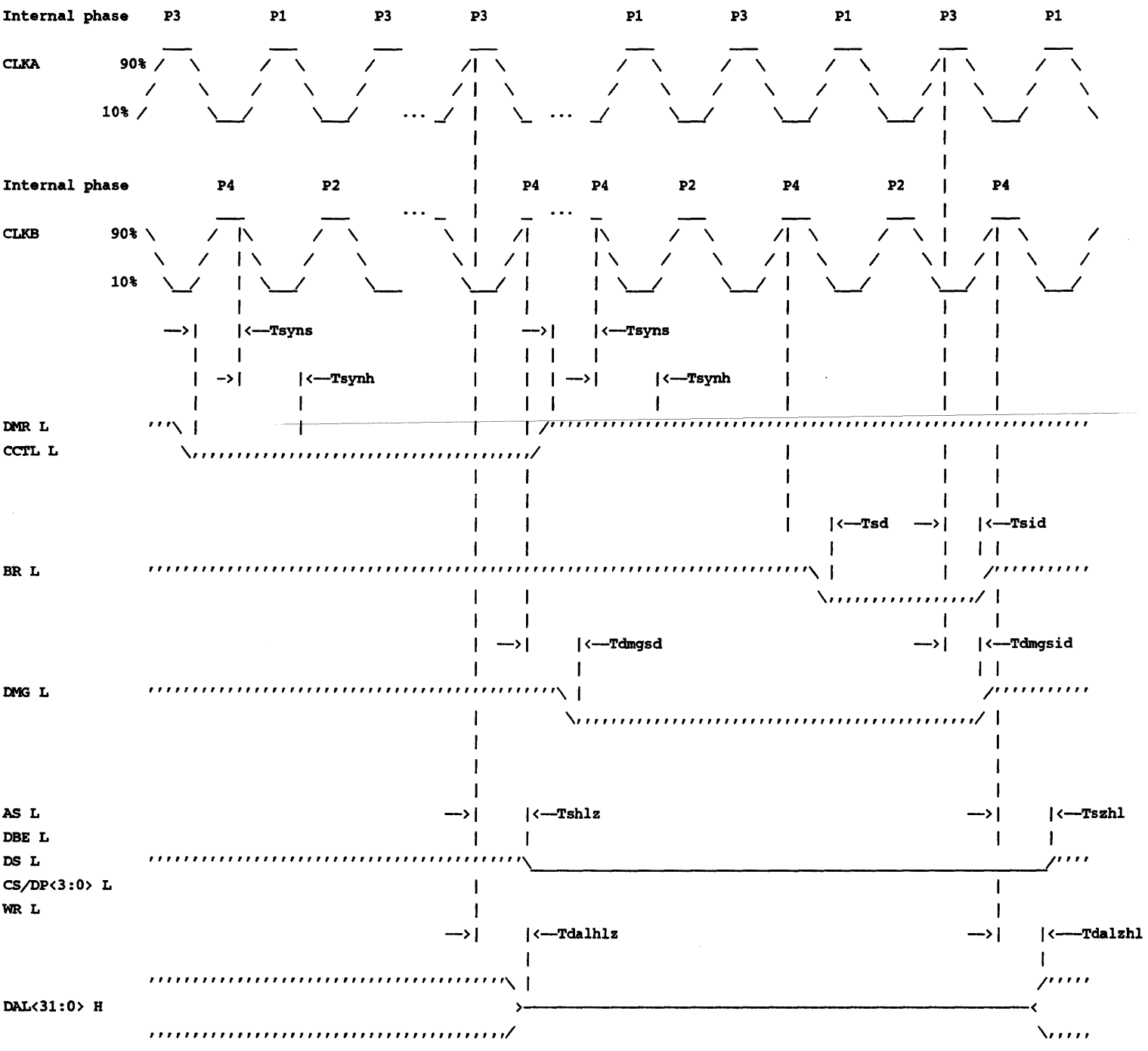


## 7.4 External Interrupt Timing



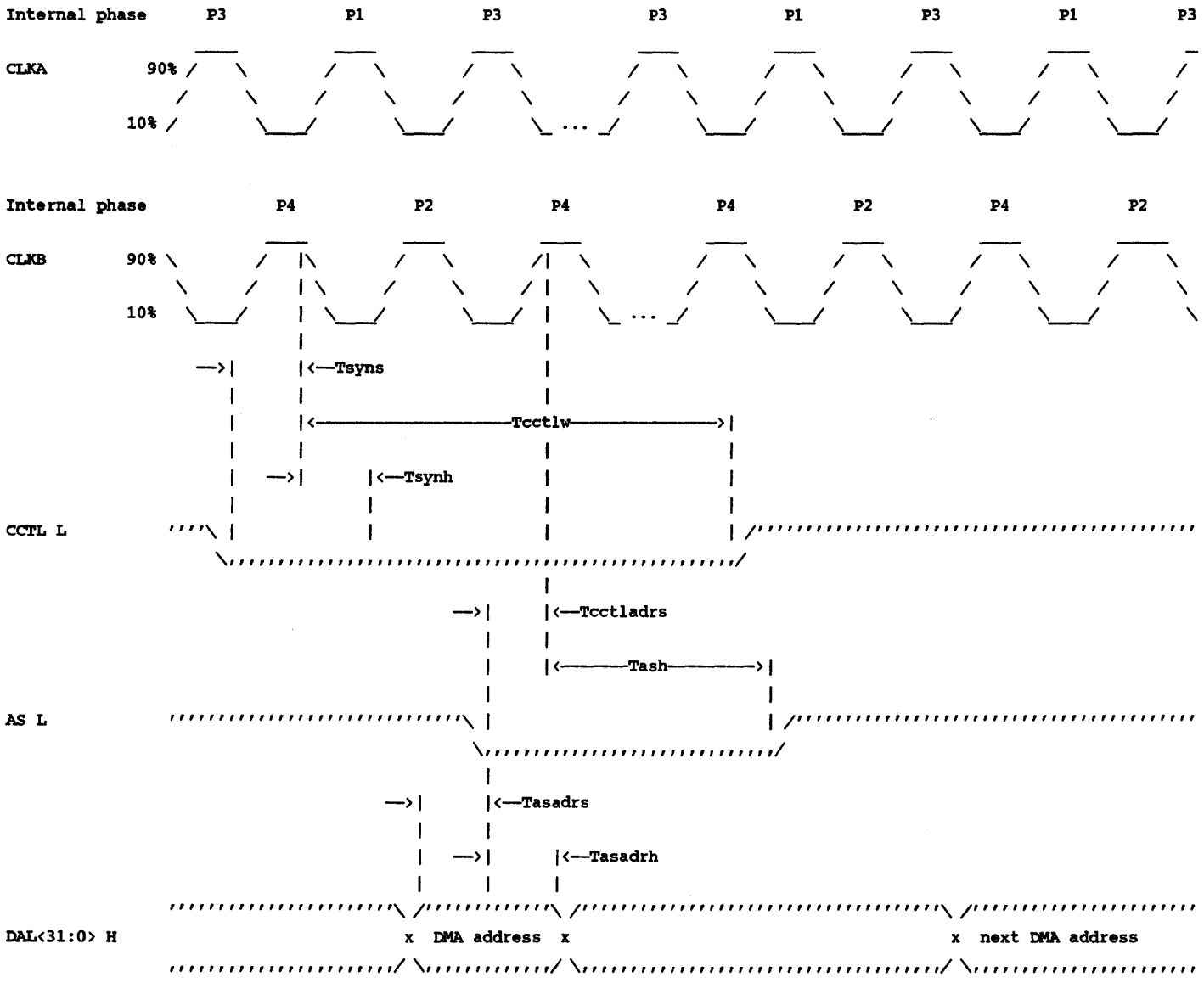

Tsyns and Tsynh are the setup and hold times needed at a synchronizer input to guarantee that the signal is recognized as expected.

## 7.5 External DMA Timing



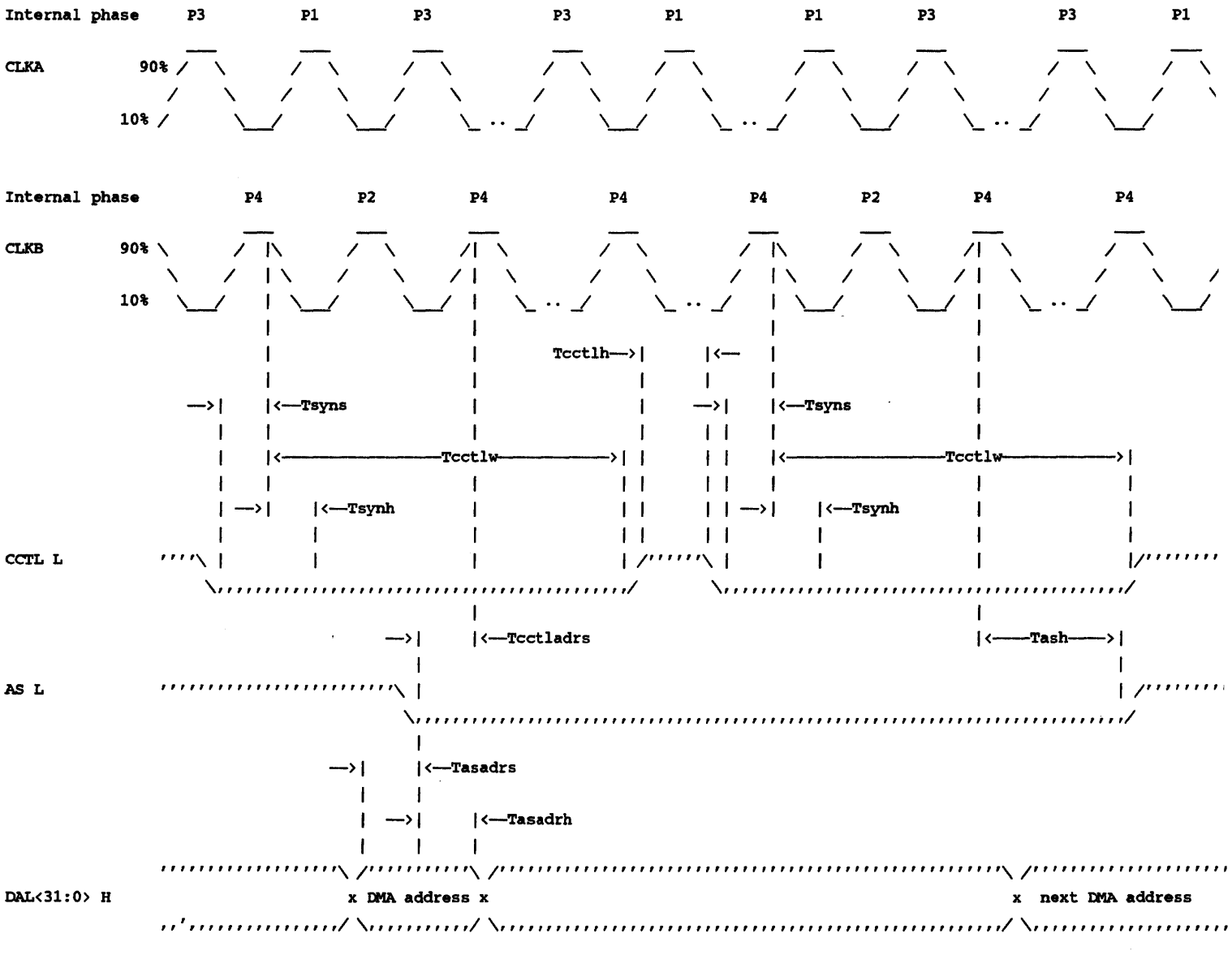

Tsyns and Tsynh are the setup and hold times needed at a synchronizer input to guarantee that the signal is recognized as expected. DMG L is asserted on P4 when DMR L is asserted eight phases earlier and no CPU IO cycle has started. DMG L is deasserted on P3 when DMR L is deasserted seven phases earlier. BR L asserts on P4 when DMG L is asserted if the CVAX is internally stalled because it needs to use the DAL. BR L deasserts when DMG L is deasserted.

## 7.6 Quadword Cache Invalidate Cycle



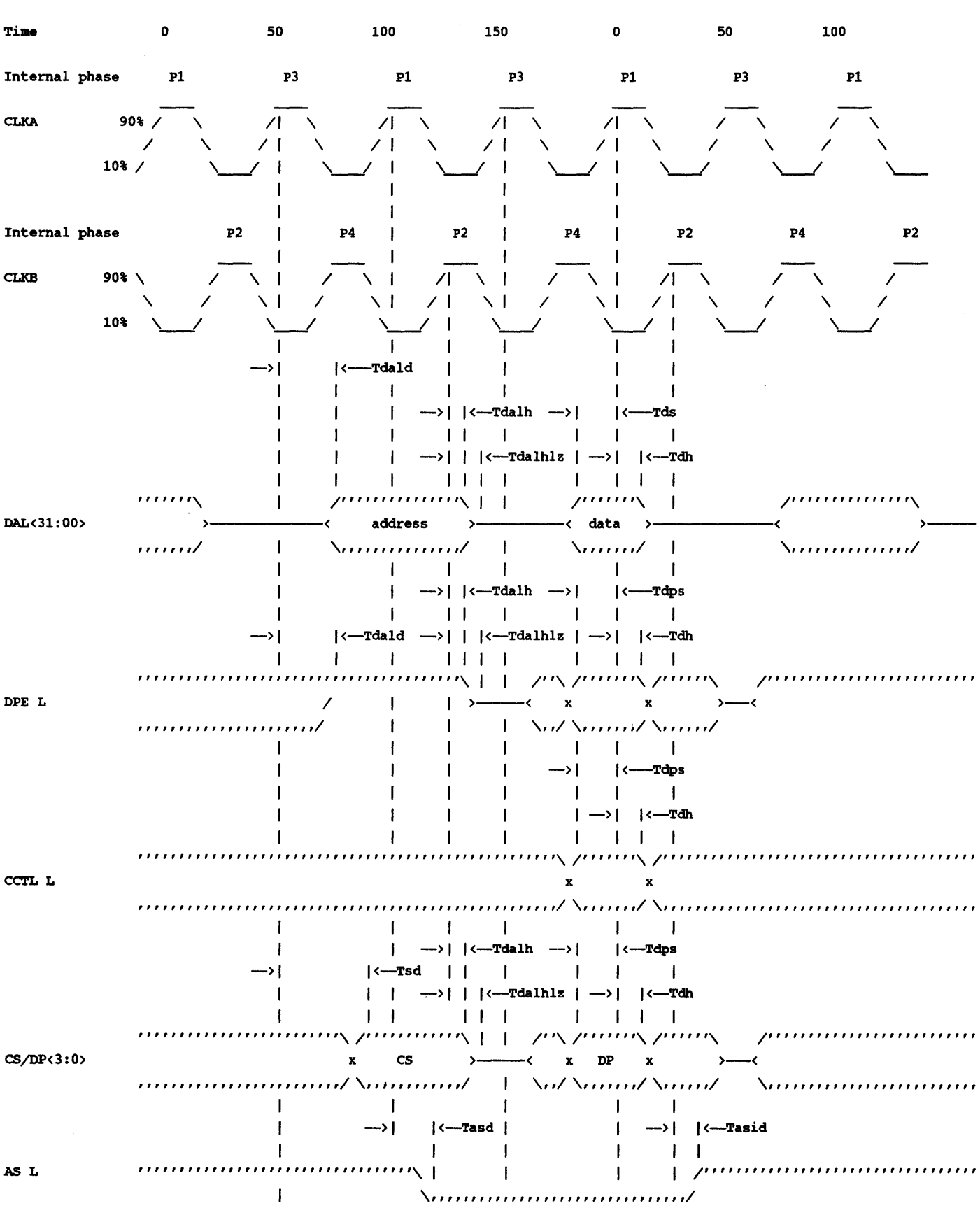

Tsyns and Tsynh are the setup and hold times needed at a synchronizer input to guarantee that the signal is recognized as expected. Tcctladrs and Tash are measured from the P4 that follows recognition of CCTL L.

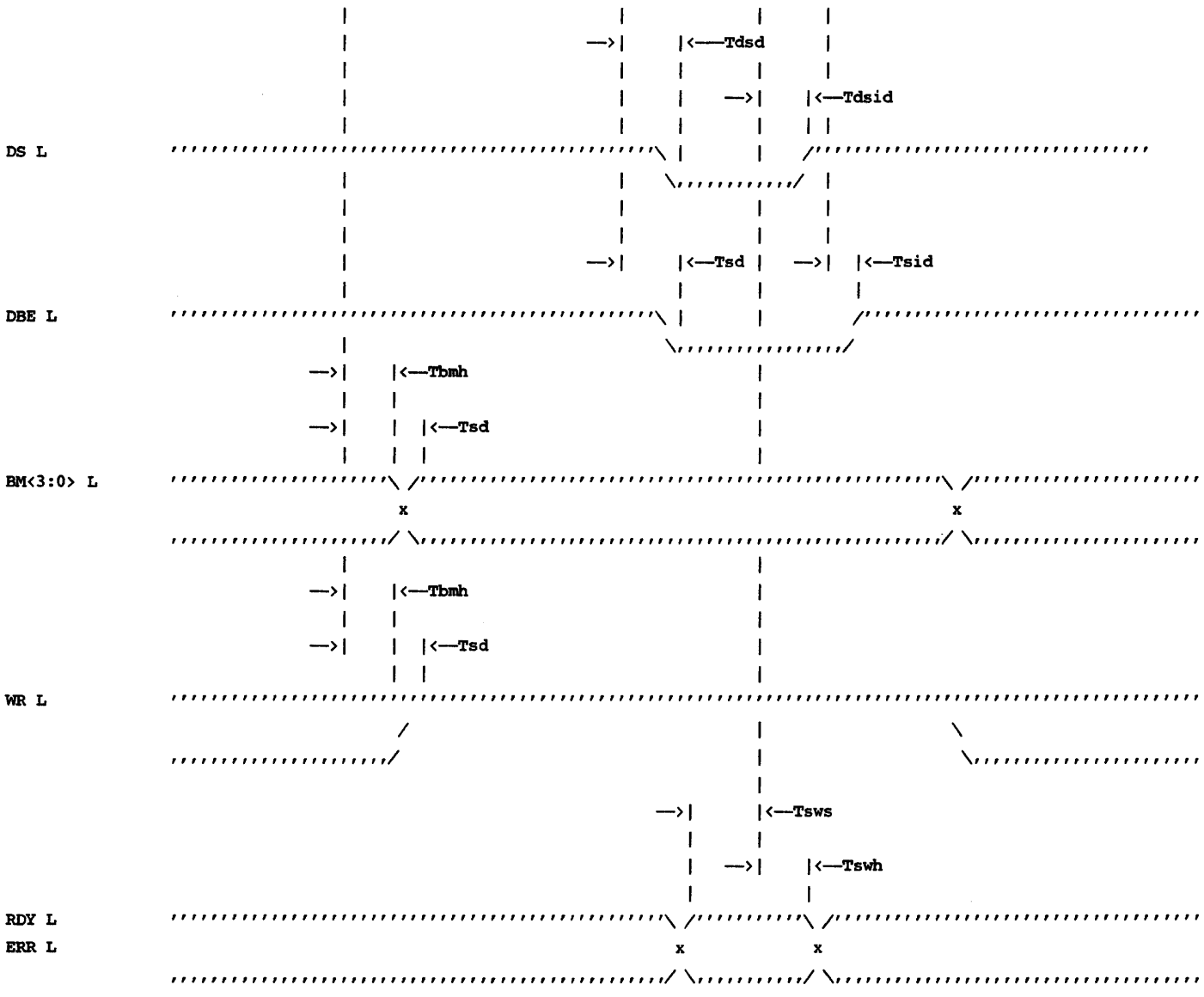
## 7.7 Octaword Cache Invalidate Cycle




Tsyns and Tsynh are the setup and hold times needed at a synchronizer input to guarantee that the signal is recognized as expected. Tcctladrs is measured from the P4 that follows recognition of the first CCTL L. Tash is measured from the P4 that follows recognition of the second CCTL.

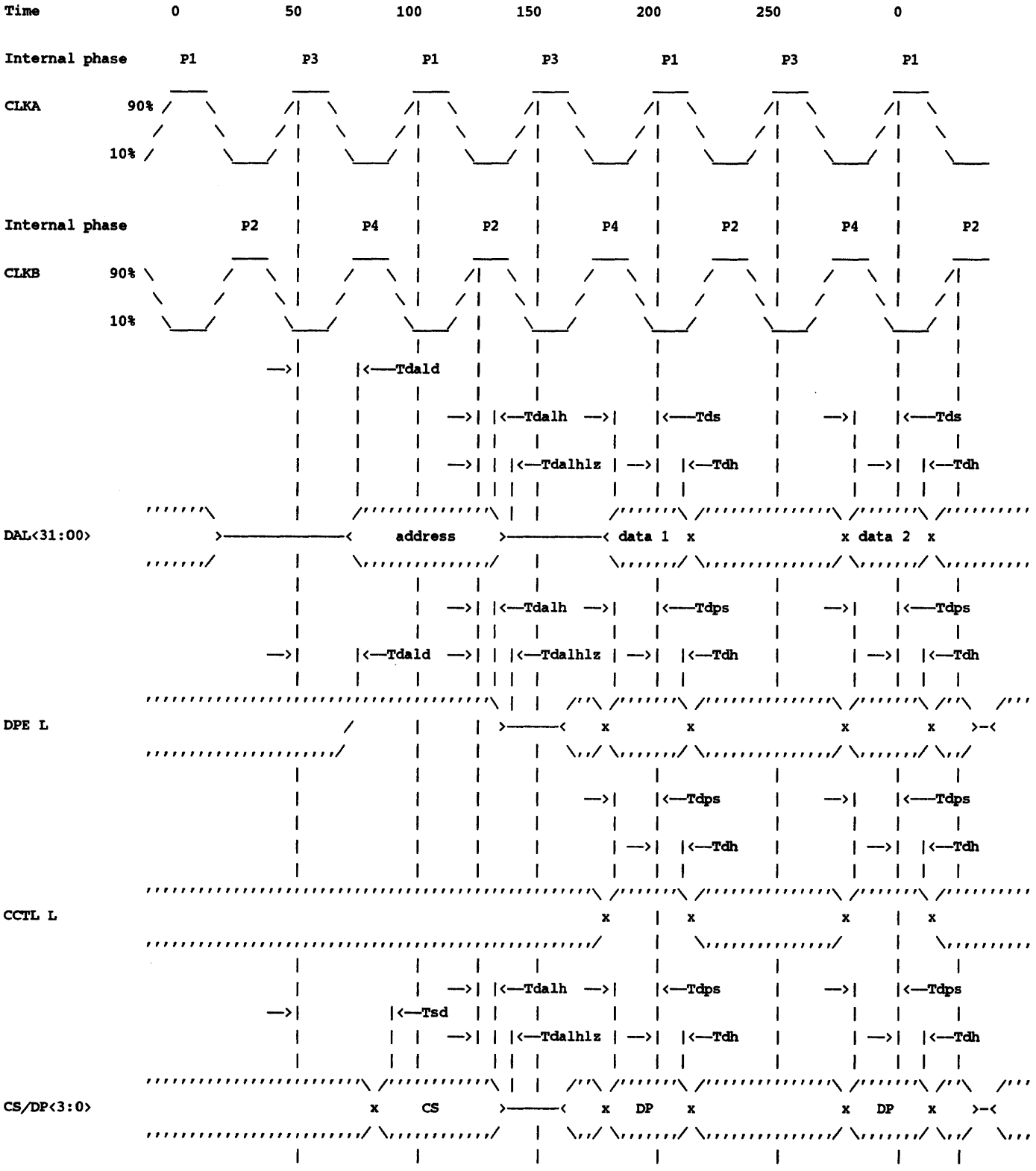
## 7.8 Single Transfer CPU Read Cycle, Interrupt Acknowledge Cycle

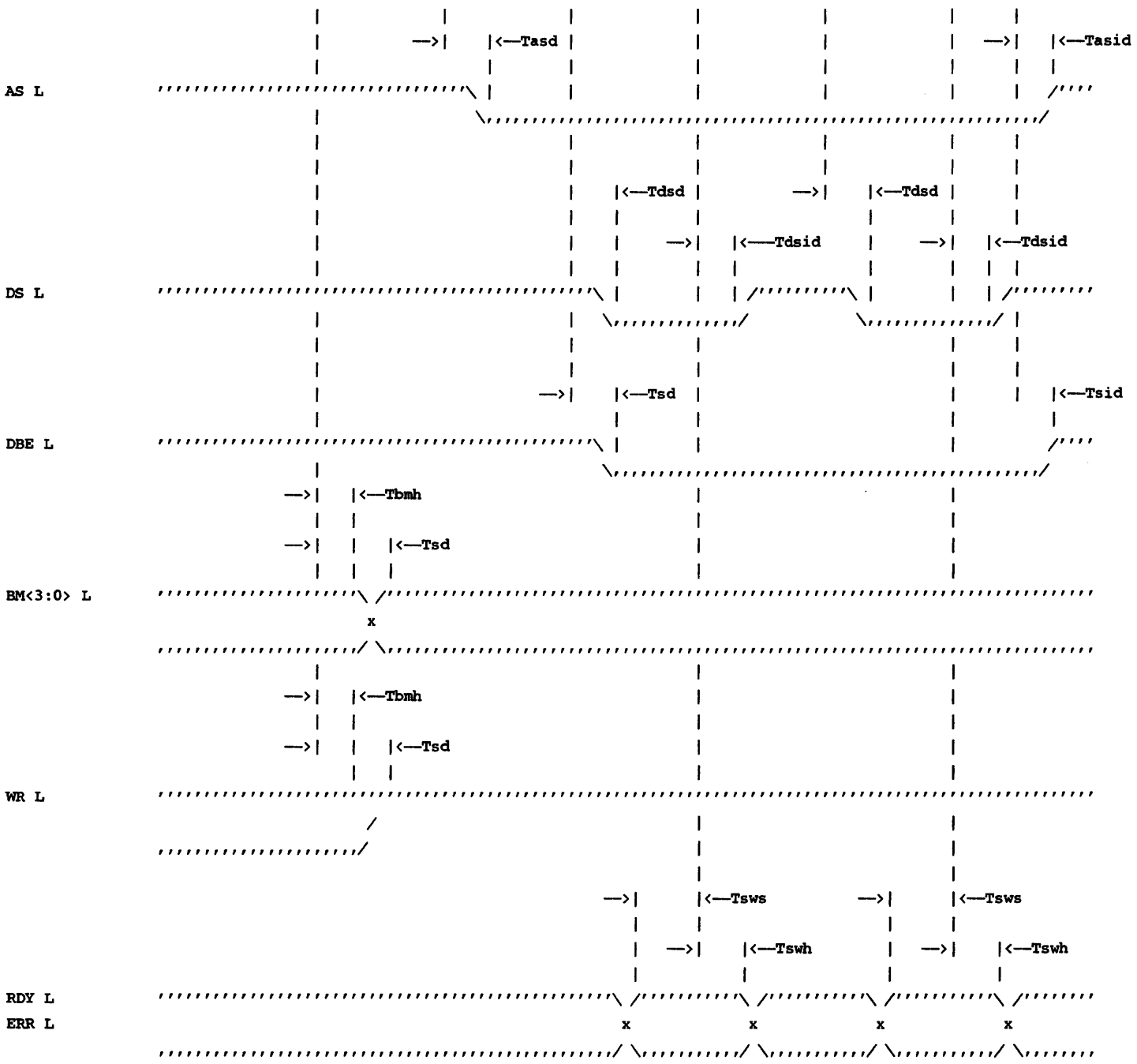





Ready slip timing: RDY L and ERR L are sampled internally coincident with data at T200. If either or both are asserted, the microcycle finishes up as shown. If neither is asserted, the chip strobes (AS L, DS L, DBE L, etc.) remain unchanged and P1 following P4 are restarted. Thus the granularity for RDY and ERR L slips is two clock cycles (nominally, 100 nsec).

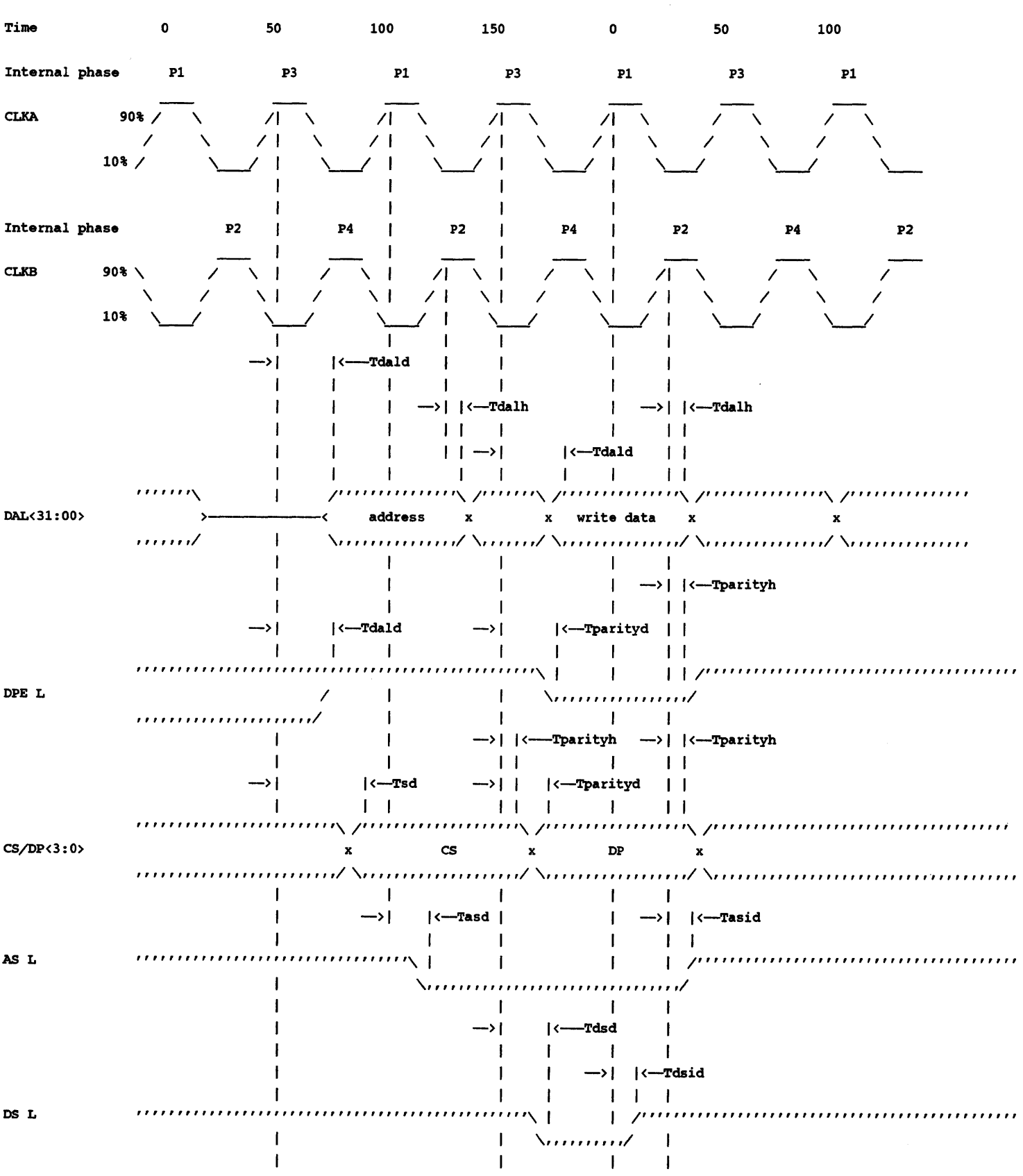
## 7.9 Multiple Transfer CPU Read Cycle

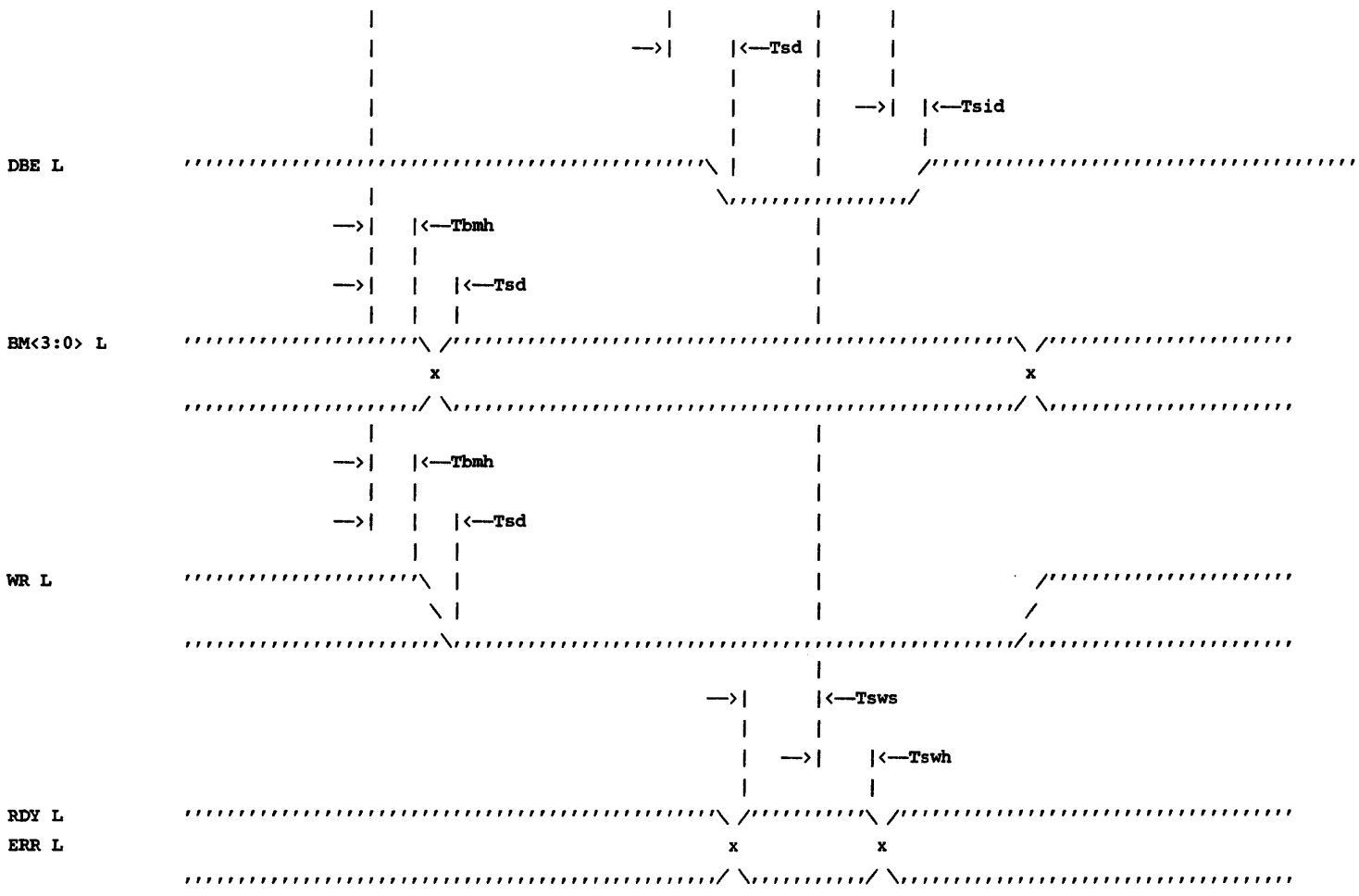





Ready slip timing: RDY L and ERR L are sampled internally coincident with data at T200 and T300. If either or both are asserted at T200 and T300, the microcycle finishes up as shown. If neither is asserted at sample point, the chip strobes (AS L, DS L, DBE L, etc.) remain unchanged and P1 following P4 are restarted. Thus the granularity at each sample point for RDY and ERR L slips is two clock cycles (nominally, 100 nsec).

## 7.10 CPU Write Cycle

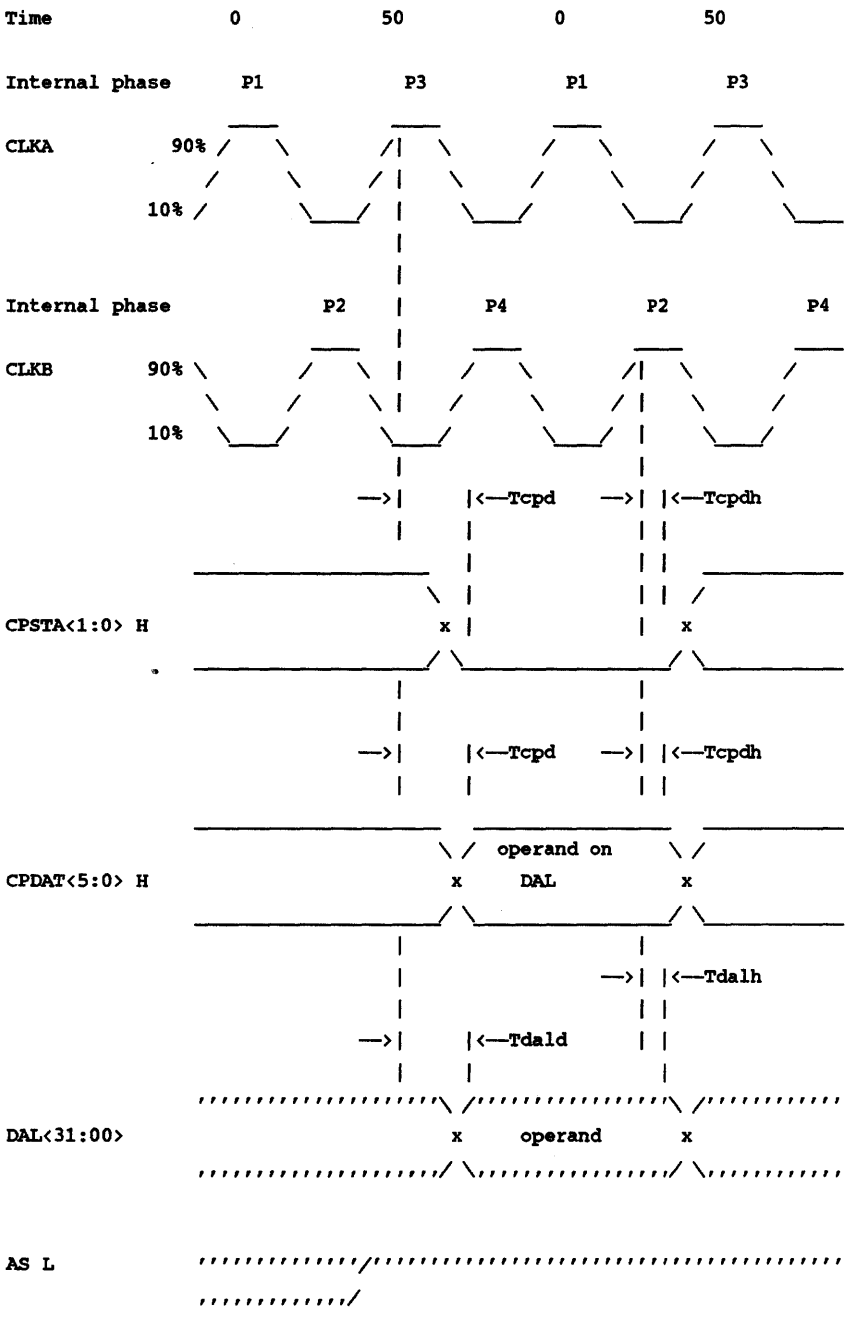





Ready slip timing: RDY L and ERR L are sampled internally at T200. If either or both are asserted, the microcycle finishes up as shown. If neither is asserted, the chip strobes (AS L, DS L, DBE L, etc.) remain unchanged and P1 following P4 are restarted. Thus the granularity for RDY and ERR L slips is two clock cycles (nominally, 100 nsec).

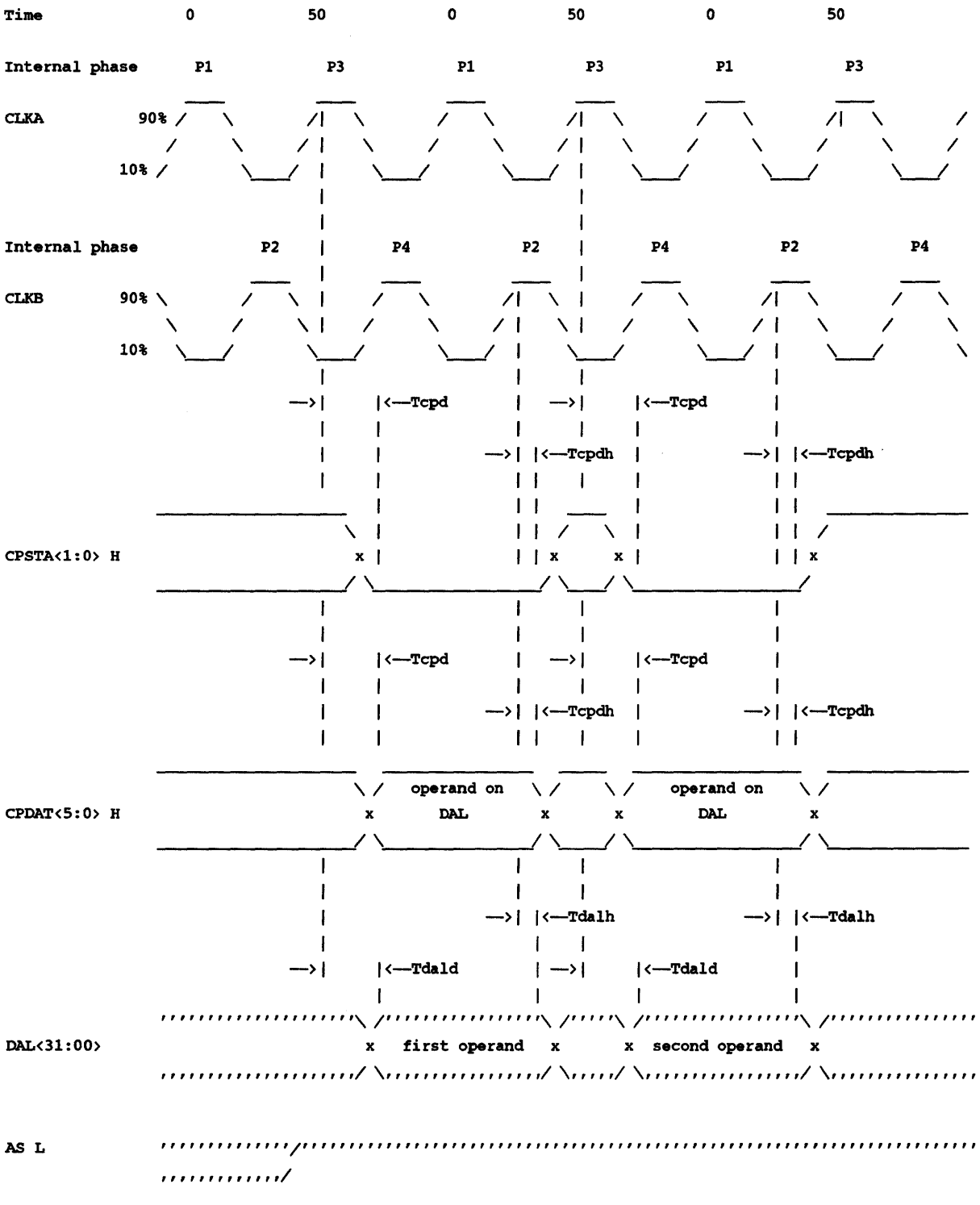
## 7.11 Coprocessor Timing

### 7.11.1 Single Precision CVAX To Coprocessor Transfer -



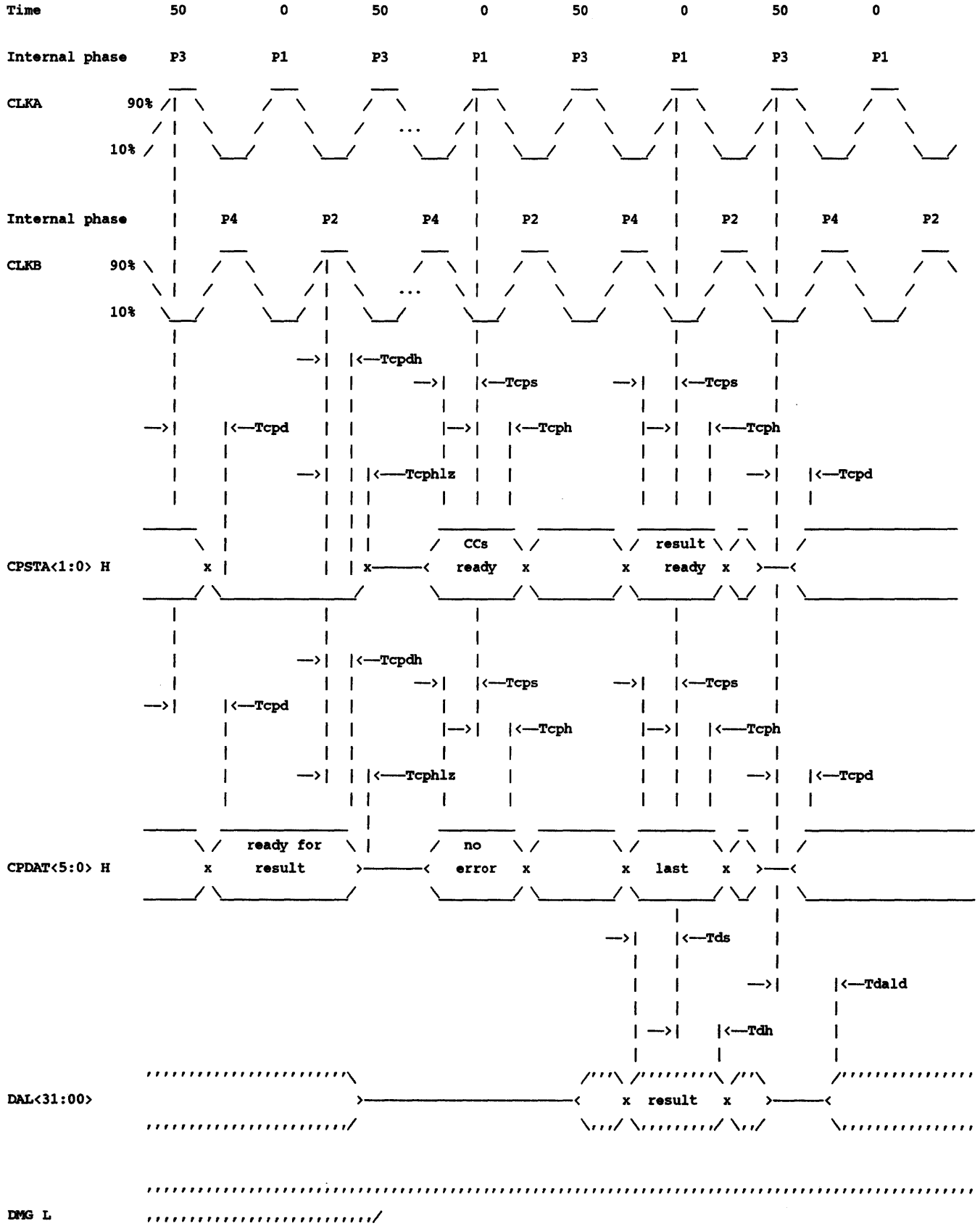

This timing is used when CVAX signals OPERAND ON DAL and AS L is not asserted. When AS L is asserted, the coprocessor reads the operand information off of the DAL according to the full memory read protocol.

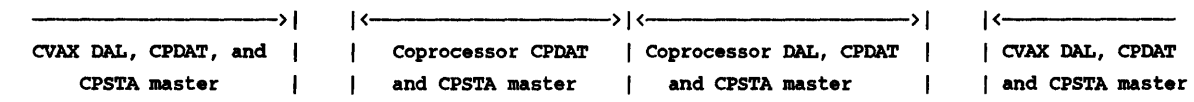
7.11.2 Double Precision CVAX To Coprocessor Transfer -




This timing is used when CVAX signals OPERAND ON DAL and AS L is not asserted. When AS L is asserted, the coprocessor reads the operand information off of the DAL according to the full memory read protocol. AS L may be asserted for either the first or second operand, or both operands.

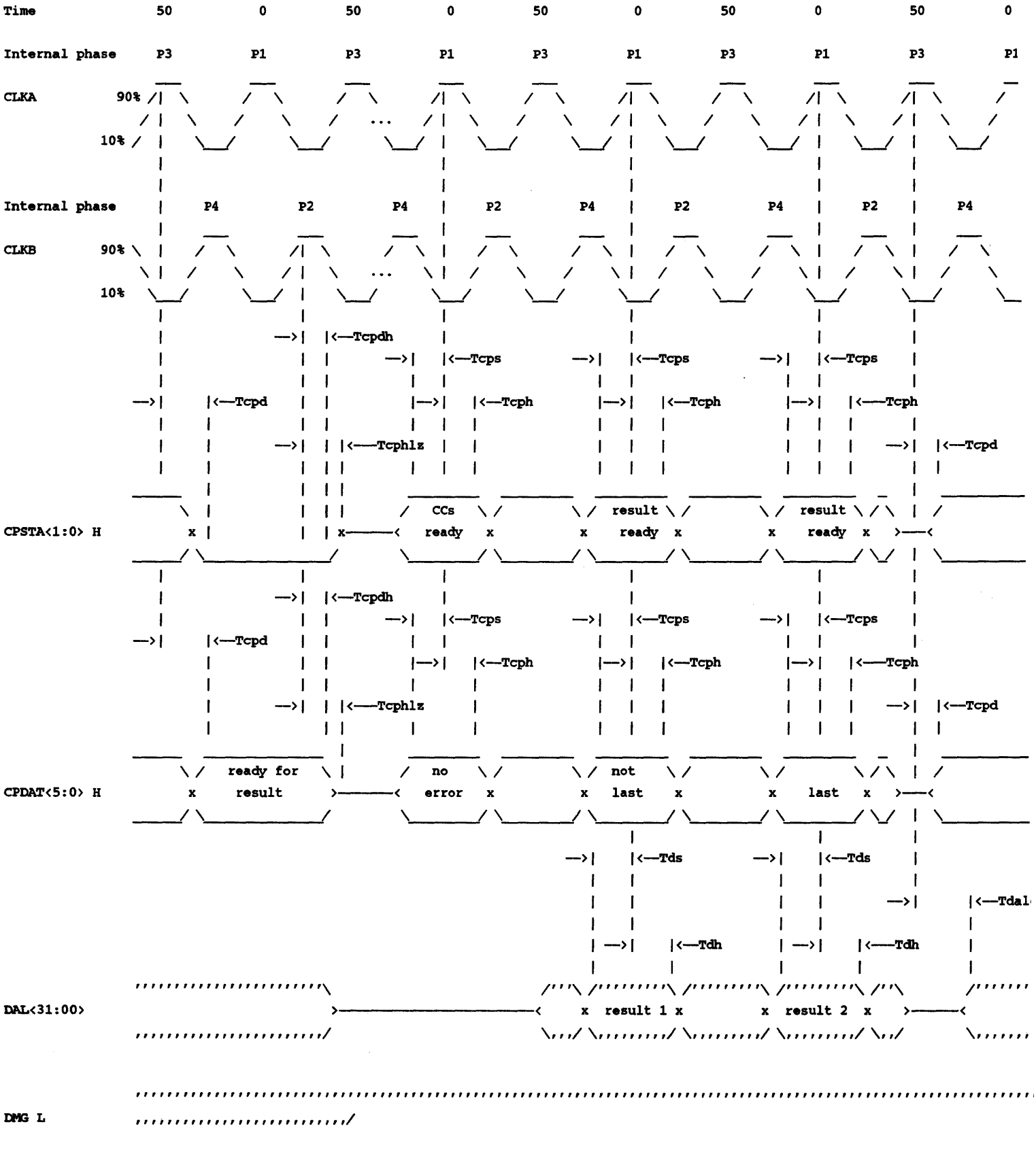
### 7.11.3 Single Precision Coprocessor To CVAX Transfers -

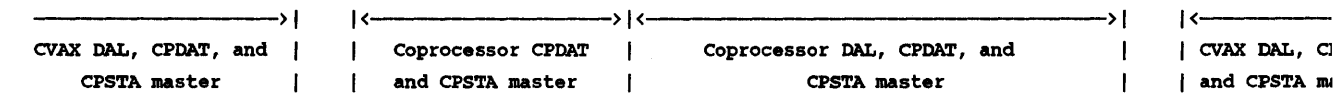





The coprocessor starts to drive CPDAT and CPSTA lines on a P3 edge after receiving a READY FOR RESULT command from CVAX. The coprocessor starts to drive DAL lines on the P3 edge after sending NO FATAL ERROR and CONDITION CODES READY status back to CVAX. while DMG L is not asserted. The coprocessor tri-states DAL, CPDAT, and CPSTA on the P2 edge after either sending back a FATAL ERROR status or the last longword result.

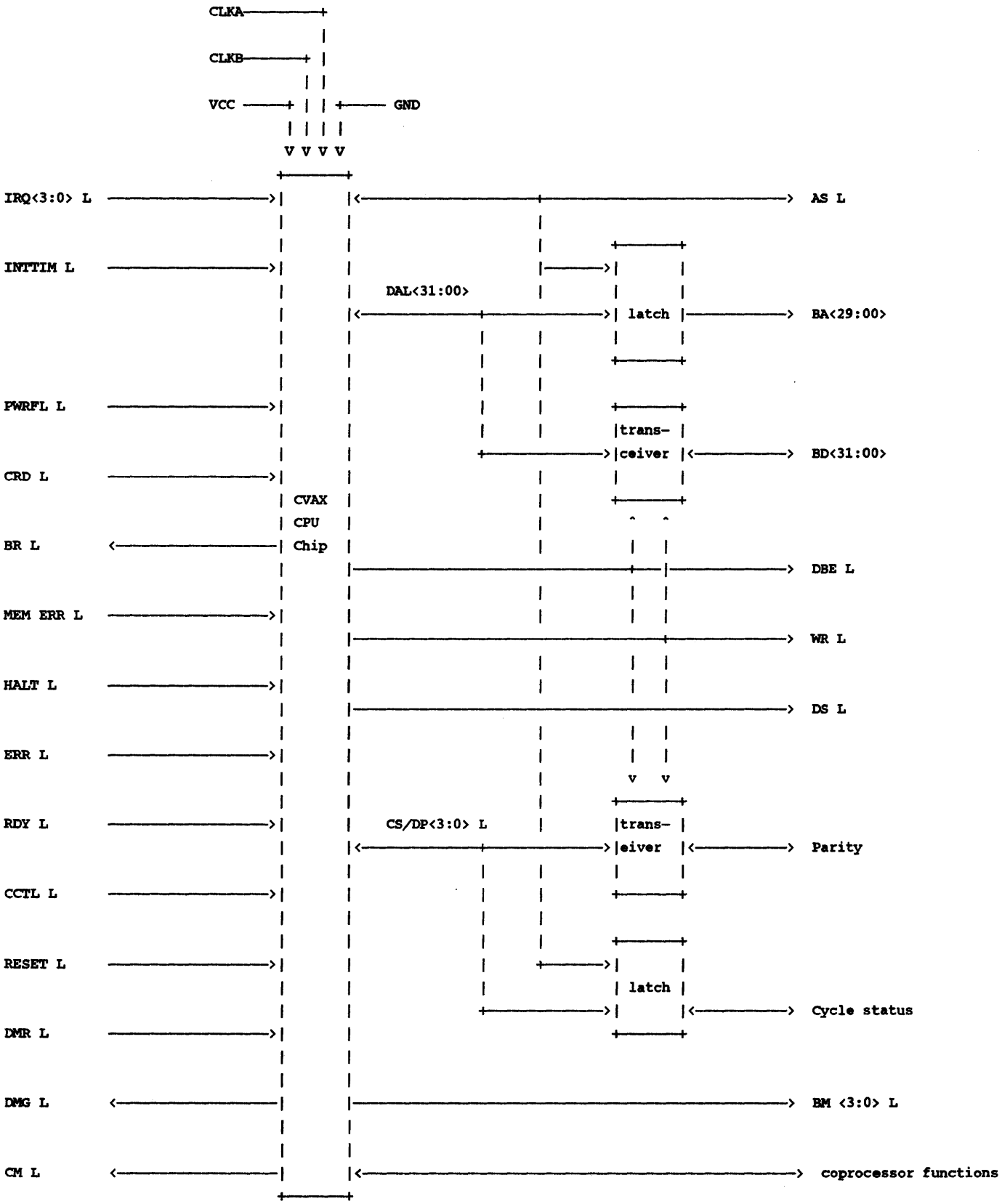
### 7.11.4 Double Precision Coprocessor To CVAX Transfers -






The coprocessor starts to drive CPDAT and CPSTA lines on a P3 edge after receiving a READY FOR RESULT command from CVAX. The coprocessor starts to drive DAL lines on the P3 edge after sending NO FATAL ERROR and CONDITION CODES READY status back to cvax. while DMG L is not asserted. The coprocessor tri-states DAL, CPDAT, and CPSTA on the P2 edge after either sending back a FATAL ERROR status or the last longword result.

## 8.0 CHIP INTERCONNECT DIAGRAM



# 9.0 DIFFERENCES BETWEEN CVAX AND UVAX

## 9.1 SOFTWARE DIFFERENCES

The following list highlights the specific software differences that exist between the two CPUs:

| Uvax | CVAX | | |
|---|---|---|---|
| | X | 1. | IPR 37 is defined as the CADR, and IPR 39 is defined as the MSER. MFPR/MTPR access these registers |
| X | | | IPR 37 and IPR 39 are not defined. MFPT/MTPR can not access these register unless they are externally defined. |
| | X | 2. | IPR 41 is defined as an accelerator (coprocessor) maintenance register. |
| X | | | IPR 41 is defined as the console saved interrupt stack pointer. |
| | X | 3. | Immediate index mode causes a reserve addressing error. |
| X | | | Immediate index mode does not cause a reserve addressing error. |
| | X | 4. | Passive release of an interrupt acknowledge cyce causes an interrupt though SCB vector 0. |
| X | | | Passive release of an interrupt acknowledge cycle cancels the entire interrupt transaction. |

## 9.2 HARDWARE DIFFERENCES

The intent of this section is provide an overview of the major hardware differences between the CVAX CPU and the uVAX CPU. In general, CVAX is targeted as a functional replacement for uVAX, and not as a pin for pin replacement. Although the basic sequencing and functionality of the strobe signals (AS L, DS L, and DBE L) is the same for both chips, the AC timing is quite different. Refer to the pin descriptions in each CPU Specification for the specific AC timing. In addition, the coprocessor protocol are very different. Refer to the coprocessor protocol descriptions in each CPU Specification for details. A uVAX coprocessor can not be used with CVAX.

The following list highlights the specific hardware differences that exist between the two CPUs:

| Uvax | CVAX | | |
|---|---|---|---|
| | X | 1. | The minimum IO cycle length is four clock phases (nominally 200 ns). |
| X | | | The minimum IO cycle length is eight clock phased (nominally 400 ns). |

| | | |
|---|---|---|
| | X | 2. The minimum IO cycle slip is two clock phases (nominally 100 ns). |
| X | | The minimum IO cycle slip is four clock phases (nominally 200 ns). |
| | X | 3. RDY L and ERR L are externally synchronized. The ready and/or error signals can be asserted with the read data. Error reporting during a minimum IO cycles is possible. |
| X | | RDY L and ERR L are internally synchronized. The ready or error signals must be asserted before the read data. Error reporting during a minimum IO cycles is not possible. |
| | X | 4. An IO cycle rety is requested when both RDY L and ERR L are asserted. |
| X | | Processor does not support IO cycle retries. |
| | X | 5. External processor protocol uses a normal IO cycle. RDY L or ERR L must be asserted to terminate the cycle. |
| X | | External processor protocol uses a ESP cycle. RDY L or ERR L are not asserted to terminate the cycle. |
| | X | 6. Parity protection is provided for DAL read and write data. DPE L allows external logic to select whether they support parity. A Memory System Error Register (MSER) is implemented which logs parity errors. |
| X | | No parity protection is provided. |
| | X | 7. Cycle status lines are time-multiplexed with parity information. This may require an external cycle status latch. |
| X | | Cycle status lines are driven on dedicated pins. No external cycle status latch is needed. |
| | X | 8. Interrupt acknowledge cycles send the interrupt grant level on DAL<6:2>. |
| X | | Interrupt acknowledge cycles send the interrupt grant level on DAL<4:0>. |
| | X | 9. Multiple transfer read cycles are supported. Two long words can be read in six cycles. |
| X | | Only single transfer read cycles are supported. |
| | X | 10. The CPU contains a 1Kb cache. Data caching can be prevented by asserting CCTL L, and DMA cache invalidates can be requested by asserting CCTL L. A cache disable register (CADR) is implemented. |
| X | | The CPU does not contain a cache. |
| | X | 11. A CPU bus request signal (BR L) is provided that informs external logic when the CPU stalls because it needs to use the DALs during DMA cycles. |
| X | | No CPU bus request signal is implemented. |
| | X | 12. A console mode signal (CM L) is provided that informs external logic when an REI instruction is executed and when execution of a new macroinstruction begins. |

| | | |
|---|---|---|
| X | | No console mode signal is implemented. |
| | X | 13. Operand read references for MOVC3/MOVC5 appear as demand D-stream reads. |
| X | | Operand read references for MOVC3/MOVC5 appear as I-stream reads. |
| | X | 14. HALT L interrupt is not acknowledged. |
| X | | HALT L interrupt is acknowledged by a ESP write to register 64. |
| | X | 15. IPR number driven out on DAL<7:2> during external processor cycles. |
| X | | IPR number driven out on DAL<5:0> during EPS cycles. |
| | X | 16. DAL<31:30> is driven with 01 for single longword transfers and 10 for quadword multiple transfers when an address is driven on DAL<29:0> (DAL<31:30> = 11 is reserved for DMA octaword transfers). |
| X | | DAL<31:30> is driven with 00 for byte, 01 for word, 10 for longword and 11 for quadword when an address is driven on DAL<29:0>. |