SIEMENS

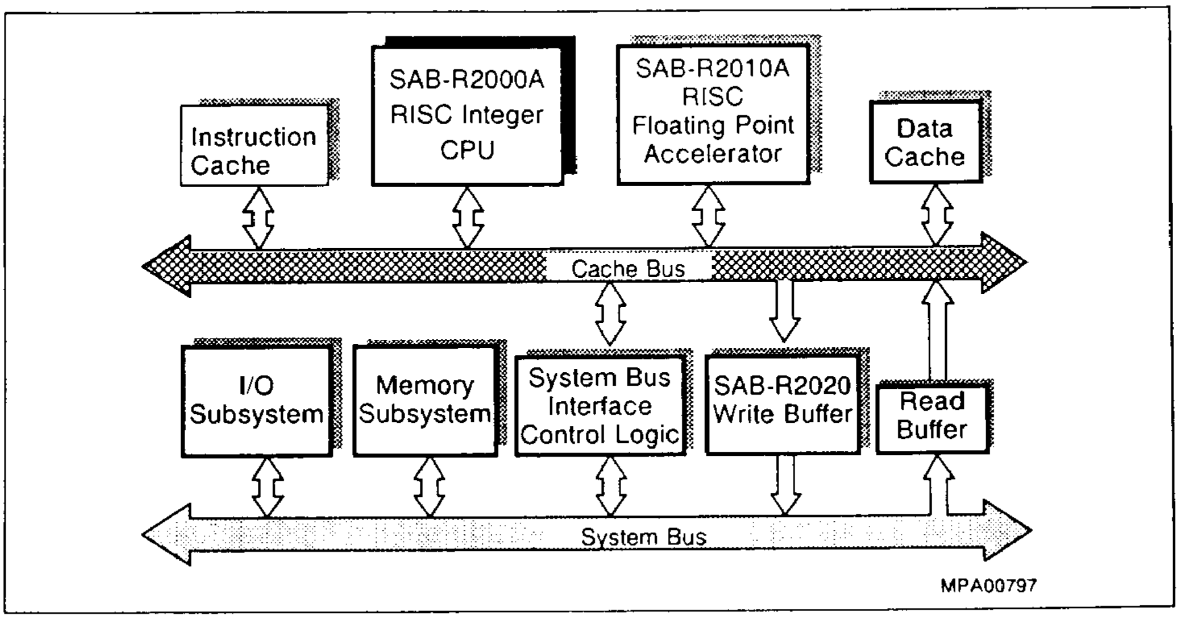
High Performance 32-Bit RISC Microprocessor

SAB-R2000A

Including on-chip memory management and cache control with support for up to three external coprocessors including the SAB-R2010A floating point accelerator.

Advance Information

- Two tightly-coupled 16 MHz units on a single chip
  - full 32-bit RISC CPU
  - system control processor (CP0)
- Load / store architecture
  - support for loading misaligned data
  - configurable endianness
- Full 32-bit operation
  - thirty two general purpose 32-bit registers
  - all instructions and addresses are 32-bits
- On-chip cache control
- On-chip memory management unit
- High Performance
  - 12 VAX 11/780 mips average at 16 MHz
- Extensive software and development support
- Instruction set compatible to R3000 processors
- Fully pin and functionally compatible to all R2000A processors of other manufacturers
- Ceramic package: C-PGA-145



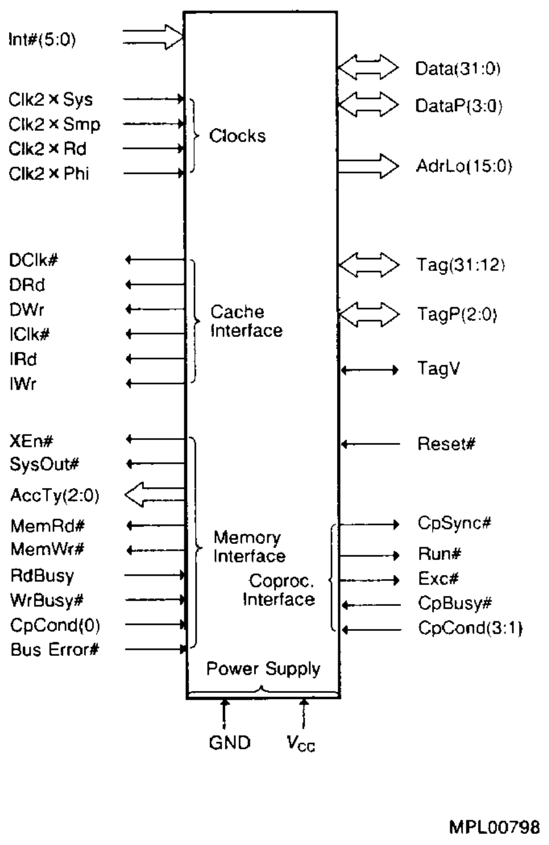
Ordering Information

Type	Ordering code	Package	Description
SAB-R2000A-12-A	Q67120-C552	C-PGA-145	32-bit RISC Processor, 12.5 MHz
SAB-R2000A-16-A	Q67120-C494	C-PGA-145	32-bit RISC Processor, 16.67 MHz
SAB-R2000A-20-A	Q67120-C517	C-PGA-145	32-bit RISC Processor, 20 MHz

Introduction

The SAB-R2000A is a high-performance microprocessor architecture implemented as a full-custom CMOS VLSI chip which achieves 20 VAX 11/780 mips average at 16 MHz. It is a single chip microprocessor that consists of two tightly-coupled 16 MHz units. The first is a full 32-bit RISC CPU. The second unit is a System Control Processor (CP0) that integrates the functions needed to keep the CPU from idling for memory access (Memory Management) and/or for Interrupt and Exception handling. The System Control Processor contains a Translation Lookaside Buffer (TLB) and control registers to efficiently support a virtual memory system, as well as all the control logic to realize separate caches for instructions and data. This architecture permits a dual-cache bandwidth of up to 200 Mbytes/second at 16 MHz using standard SRAM devices. It is possible for up to three external coprocessors (including the SAB-R2010A floating point accelerator) to be coupled with the SAB-R2000A. The synchronous coprocessor interface generates all the addresses and manages the memory interface control.

Figure 1
Logic Symbol



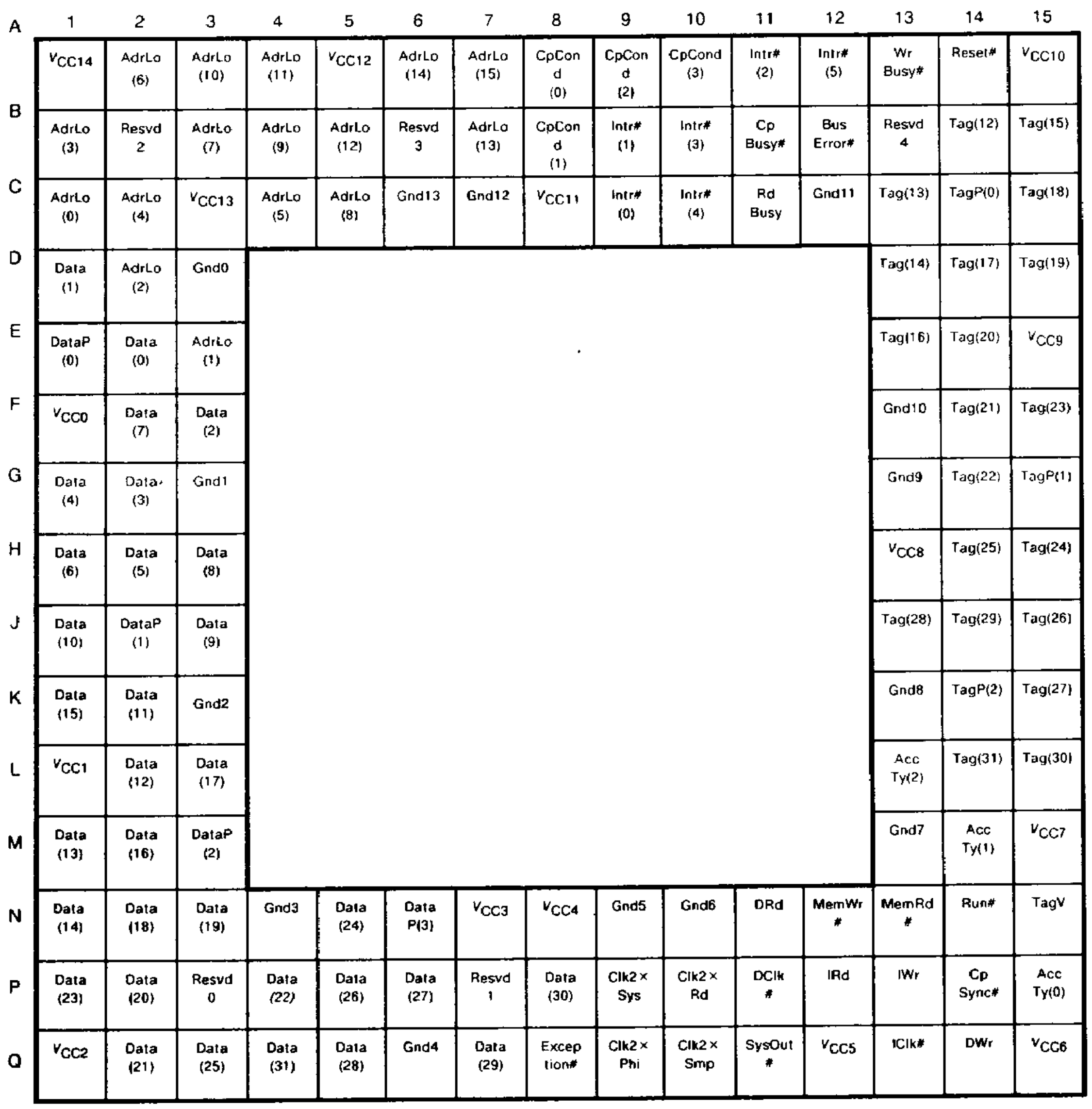
Note: "#" signifies an active low signal.

Pin Names

Data(31:0)	Data Bus
DataP(3:0)	Even Parity for Data Bus
AdrLo(15:0)	Low Adress Bus
Tag(31:12)	Cache Tag and High Adress Bus
TagP(2:0)	Even Parity over TagV and Tag Bus
TagV	Tag Validity Indicator
Reset#	Synchronous Initialization
CpSync#	Coprocessor Synchronization
Run#	Processor in Run or Stall State
Exc#	Exception
CpBusy#	Coprocessor Busy
CpCond(3:0)	Coprocessor Condition
BusError#	Bus Error
WrBusy#	Write Busy for Main Memory
RdBusy#	Read Busy for Main Memory
MemWr#	Main Memory Write
MemRd#	Main Memory Read
AccTy(2:0)	Access Type
SysOut#	System Clock Out
XEn#	Read Enable (Read Buffer)
DWr	Data Cache Write Enable
IWr	Instruction Cache Write Enable
DRd	Data Cache Read Enable
IRd	Instruction Cache Read Enable
DClk#	Data Cache Latch Enable
IClk#	Instruction Cache Latch Enable
Int#(5:0)	Interrupt Bus

Pin Configurations

Figure 2
C-PGA-145 (Top View)



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
A	V_{CC14}	AdrLo (6)	AdrLo (10)	AdrLo (11)	V_{CC12}	AdrLo (14)	AdrLo (15)	CpCon d (0)	CpCon d (2)	CpCond (3)	Intr# (2)	Intr# (5)	Wr Busy#	Reset#	V_{CC10}
B	AdrLo (3)	Resvd 2	AdrLo (7)	AdrLo (9)	AdrLo (12)	Resvd 3	AdrLo (13)	CpCon d (1)	Intr# (1)	Intr# (3)	Cp Busy#	Bus Error#	Resvd 4	Tag(12)	Tag(15)
C	AdrLo (0)	AdrLo (4)	V_{CC13}	AdrLo (5)	AdrLo (8)	Gnd13	Gnd12	V_{CC11}	Intr# (0)	Intr# (4)	Rd Busy	Gnd11	Tag(13)	TagP(0)	Tag(18)
D	Data (1)	AdrLo (2)	Gnd0										Tag(14)	Tag(17)	Tag(19)
E	DataP (0)	Data (0)	AdrLo (1)										Tag(16)	Tag(20)	V_{CC9}
F	V_{CC0}	Data (7)	Data (2)										Gnd10	Tag(21)	Tag(23)
G	Data (4)	Data (3)	Gnd1										Gnd9	Tag(22)	TagP(1)
H	Data (6)	Data (5)	Data (8)										V_{CC8}	Tag(25)	Tag(24)
J	Data (10)	DataP (1)	Data (9)										Tag(28)	Tag(29)	Tag(26)
K	Data (15)	Data (11)	Gnd2										Gnd8	TagP(2)	Tag(27)
L	V_{CC1}	Data (12)	Data (17)										Acc Ty(2)	Tag(31)	Tag(30)
M	Data (13)	Data (16)	DataP (2)										Gnd7	Acc Ty(1)	V_{CC7}
N	Data (14)	Data (18)	Data (19)	Gnd3	Data (24)	Data P(3)	V_{CC3}	V_{CC4}	Gnd5	Gnd6	DRd	MemWr #	MemRd #	Run#	TagV
P	Data (23)	Data (20)	Resvd 0	Data (22)	Data (26)	Data (27)	Resvd 1	Data (30)	Clk2× Sys	Clk2× Rd	DClk #	IRd	IWr	Cp Sync#	Acc Ty(0)
Q	V_{CC2}	Data (21)	Data (25)	Data (31)	Data (28)	Gnd4	Data (29)	Excep tion#	Clk2× Phi	Clk2× Smp	SysOut #	V_{CC5}	IClk#	DWr	V_{CC6}

MPP00799

Pin Definitions and Functions

Symbol	Pin Number	Input (I) Output (O)	Function
Data(31:0)	Q4,P8,Q7,Q5, P6,P5,Q3,N5, P1,P4,Q2,P2, N3,N2,L3,M2, K1,N1,M1,L2, K2,J1,J3,H3, F2,H1,H2,G1, G2,F3,D1,E2	I/O	A 32-bit bus used for all instruction and data transmission among the processor, caches, memory interface, and coprocessors.
DataP(3:0)	N6,M3,J2,E1	I/O	A 4-bit bus containing even parity over the data bus.
Tag(31:12)	L14,L15,J14, J13,K15,J15, H14,H15,F15, G14,F14,E14, D15,C15,D14, E13,B15,D13, C13,B14	I/O	A 20-bit bus used for transferring cache tags and high addresses between the processor, caches, and memory interface.
TagV	N15	I/O	The tag validity indicator
TagP(2:0)	K14,G15,C14	I/O	A 3-bit bus containing even parity over the catenation of TagV and Tag31:12.
AdrLo (15:0)	A7,A6,B7,B5, A4,A3,B4,C5, B3,A2,C4,C2, B1,D2,E3,C1	O	A 16-bit bus containing byte addresses used for transferring low addresses from the processor to the caches and memory interface.
IRd	P12	O	Read enable for the instruction cache.
IWr	P13	O	Write enable for the instruction cache.
IClk#	Q13	O	The instruction cache address latch clock. This clock runs continuously.
DRd	N11	O	The read enable for the data cache.
DWr	Q14	O	The write enable for the data cache.
DClk#	P11	O	The data cache address latch clock. This clock runs continuously.

Pin Definitions and Functions (cont'd)

Symbol	Pin Number	Input (I) Output (O)	Function
AccTy(2:0)	L13,M14,P15	O	A 3-bit bus used to indicate the size of data being transferred on the data bus, whether or not a data transfer is occurring, and the purpose of the transfer. The run encoding of the Access Type is illustrated in the table below.
MemWr#	N12	O	Signals the occurrence of a main memory write.
MemRd#	N13	O	Signals the occurrence of a main memory read.
BusError#	B12	I	Signals the occurrence of a bus error during a main memory read or write.
Run#	N14	O	Indicates whether the processor is in the run or stall state.
Exc#	Q8	O	Indicates that the instruction about to commit state should be aborted.
SysOut#	Q11	O	A reflection of the internal processor clock used to generate the system clock.
CpSync#	P14	O	A clock which is identical to SysOut# and used by coprocessors for timing synchronization with the CPU.
RdBusy	C11	I	The main memory read stall termination signal. In most system designs RdBusy is normally asserted and is deasserted only to indicate the successful completion of a memory read. RdBusy is sampled by the processor only during memory read stalls.
WrBusy#	A13	I	The main memory write stall initiation/termination signal.

AccTy(2)	AccTy(1:0)	size
1	XX	no transaction
0	00	byte
0	01	half word
0	10	tribyte
0	11	word

Pin Definitions and Functions (cont'd)

Symbol	Pin Number 144-Pin	Input (I) Output (O)	Function
CpBusy#	B11	I	The coprocessor busy stall initiation/termination signal.
CpCond (3:0)	A10,A9,B8,A8	I	A 4-bit bus used to transfer conditional branch status from the coprocessors to the main processor.
Int#(5:0)	A12,C10,B10, A11,B9,C9	I	A 6-bit bus used by the memory interface and coprocessor to signal maskable interrupts to the processor. It is also used to specify the processor's mode during Reset. The Table below summarizes the mode selectable features.

Inter-	Y Cycle Modes		Z Cycle Modes	
rupt#	Phase 1	Phase 2	Phase 1	Phase 2
Int#(0)	Reserved	Reserved	Reserved	BigEndian#
Int#(1)	Reserved	Reserved	Reserved	Tristate#
Int#(2)	Reserved	Reserved	Reserved	NoCache#
Int#(3)	Reserved	Reserved	Reserved	Bus DriveOn
Int#(4)	Phase DelayOn#	Phase DelayOn#	Asserted#	Phase DelayOn#
Int#(5)	Reserved	Reserved	Asserted#	R2000Md

Symbol	Pin Number 144-Pin	Input (I) Output (O)	Function
Clk2 × Sys	P9	I	The master double frequency input clock used for generating SysOut#.
Clk2 × Smp	Q10	I	A double frequency clock input used to determine the sample point for data coming into the processor and coprocessors.
Clk2 × Rd	P10	I	A double frequency clock input used to determine the enable time of the cache RAMs.
Clk2 × Phi	Q9	I	A double frequency clock input used to determine the position of the internal phases, phase1 and phase2.
Reset#	A14	I	Synchronous initialization input used to force execution starting from the reset memory address. Reset# must be deasserted synchronously but asserted asynchronously. The deassertion of Reset# must be synchronized by the leading edge of SysOut.

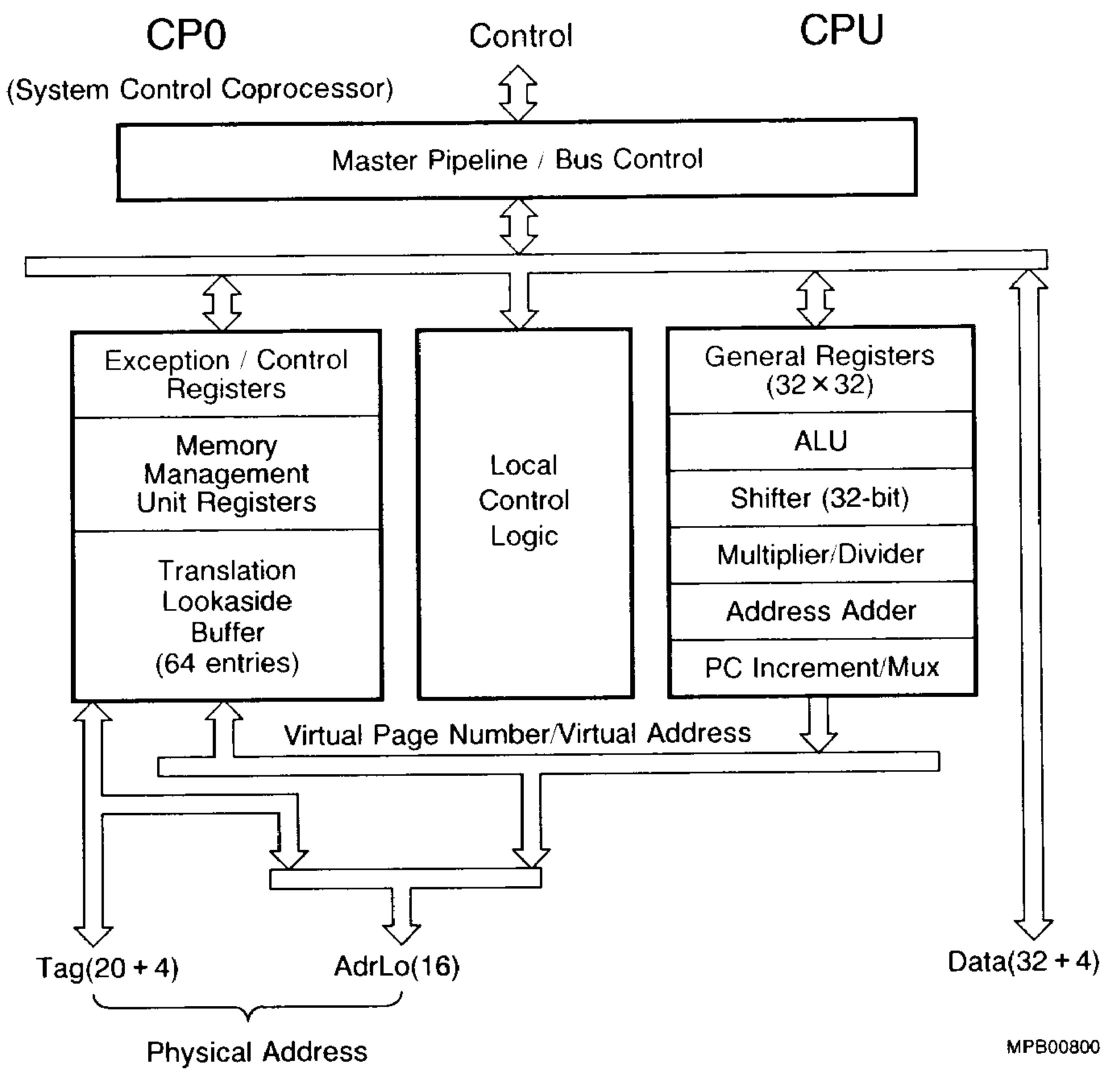
Pin Definitions and Functions (cont'd)

Symbol	Pin Number 144-Pin	Input (I) Output (O)	Function
GND13-0	C6,C7,C12, F13,G13, K13,M13, N10,N9, Q6,N4,K3, G3,D3		Ground
V_{CC}14-0	A1,C3,A5,C8, A15,E15,H13, M15,Q15, Q12,N8,N7, Q1,L1,F1		Power Supply (+ 5 V)
Resvd4-0	B13,B6,B2, P7,P3		Reserved

Functional Description

The SAB-R2000A consists of two integrated processors - a RISC CPU and a System Control Processor (CP0). Figure 3 is a block diagram of the SAB-R2000A which shows the functions incorporated within it.

Figure 3
Functional Block Diagram


CP0
(System Control Coprocessor)
Control
CPU
Master Pipeline / Bus Control
Exception / Control Registers
Memory Management Unit Registers
Translation Lookaside Buffer (64 entries)
Local Control Logic
General Registers (32 × 32)
ALU
Shifter (32-bit)
Multiplier/Divider
Address Adder
PC Increment/Mux
Virtual Page Number/Virtual Address
Tag(20 + 4)
AdrLo(16)
Physical Address
Data(32 + 4)
MPB00800

Basic Architecture

On the right hand side of figure 3 is the CPU datapath that implements the 5-stage pipeline, which will be explained shortly. The datapath consists of a stack of functional units including 32 General Registers, ALU, 32-bit Shifter and an autonomous Multiply Divide unit. An Address Adder and Increment MUX for the PC generate instruction and data addresses alternatively at double the basic clock rate. This is necessary so that the SAB-R2000A can access both the Instruction and Data caches in a single CPU cycle, due to the multiplexed Data bus.

On the left of figure 3 is the System Control Processor (CP0). It's major element is a fully associative 64-entry Translation Lookaside Buffer (TLB), which translates a 20-bit virtual page number into a physical page frame number in a clock phase. As well as address translation the System Control Coprocessor also manages the exception handling and error recovery, the external cache interface, the memory control interface and the external coprocessor interface. It also incorporates on-chip tag comparators, parity generators and checkers.

Integer CPU

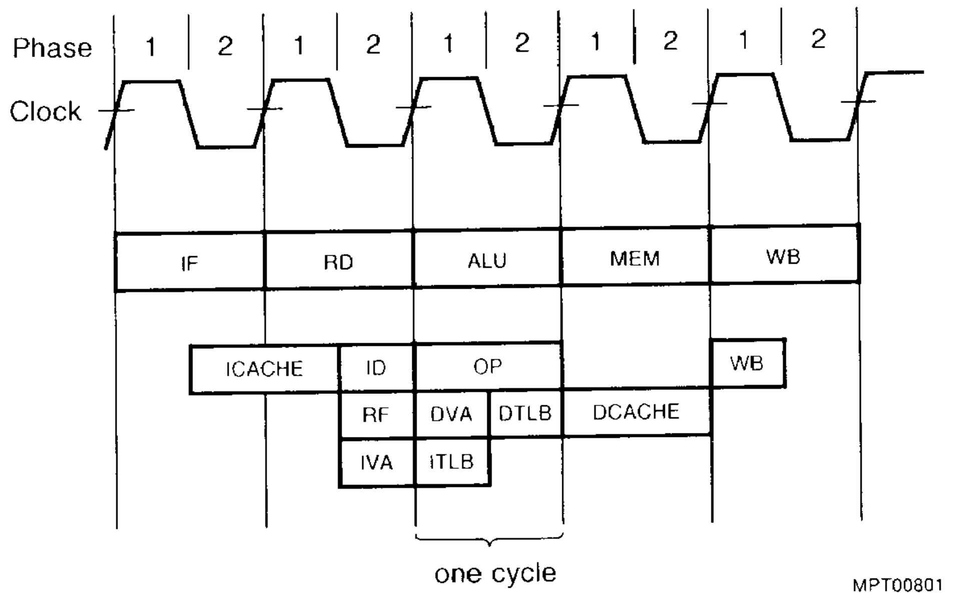
Pipeline Architecture

The execution of a single SAB-R2000A instruction consists of five primary steps or pipe stages.

(1) IF	Instruction Fetch: Access the TLB and translate the instructions virtual address into its physical address to read an instruction from the I-cache. Note that the instruction is not actually read into the processor until the beginning (phase 1) of the RD pipe stage. Refer to figure 4.
(2) RD	Register Fetch Instruction Decode: Read any required operands from the CPU registers while decoding the instruction.
(3) ALU	ALU Operation: Perform the required operation on instruction operands..
(4) MEM	Memory Access. Access memory (D-cache) if required (for a Load or Store instruction).
(5) WB	Writeback: Write back ALU results or value loaded from D-cache to the register file.

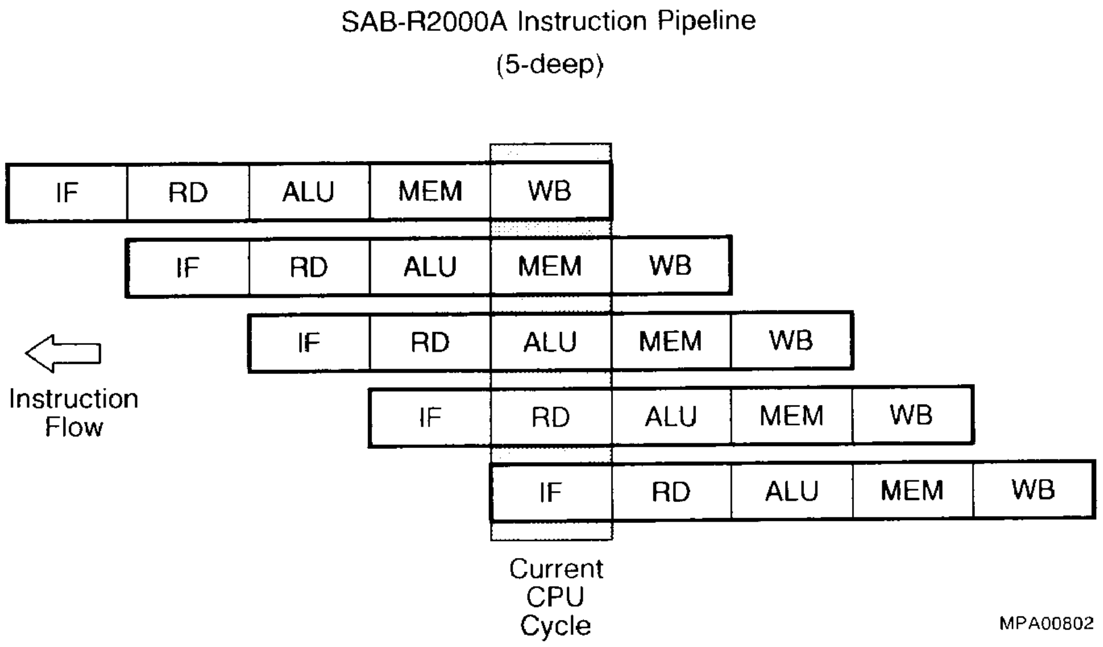
Each of these steps requires approximately one CPU cycle as shown in figure 4 (parts of some operations lap over into another cycle while other operations require only 1 2 cycle).

Figure 5
Execution Sequence of an Instruction



ICACHE	:	Instruction Cache Access
ID	:	Instruction Decode
RF	:	Register Operand Fetch
OP	:	Operation (ALU/Shift)
IVA	:	Instruction Virtual Address Calculation
ITLB	:	TLB Access for Instruction
DVA	:	Data Virtual Address Calculation
DTLB	:	TLB Access for Data
DCACHE	:	Data Cache Access
WB	:	Write Back to Register File

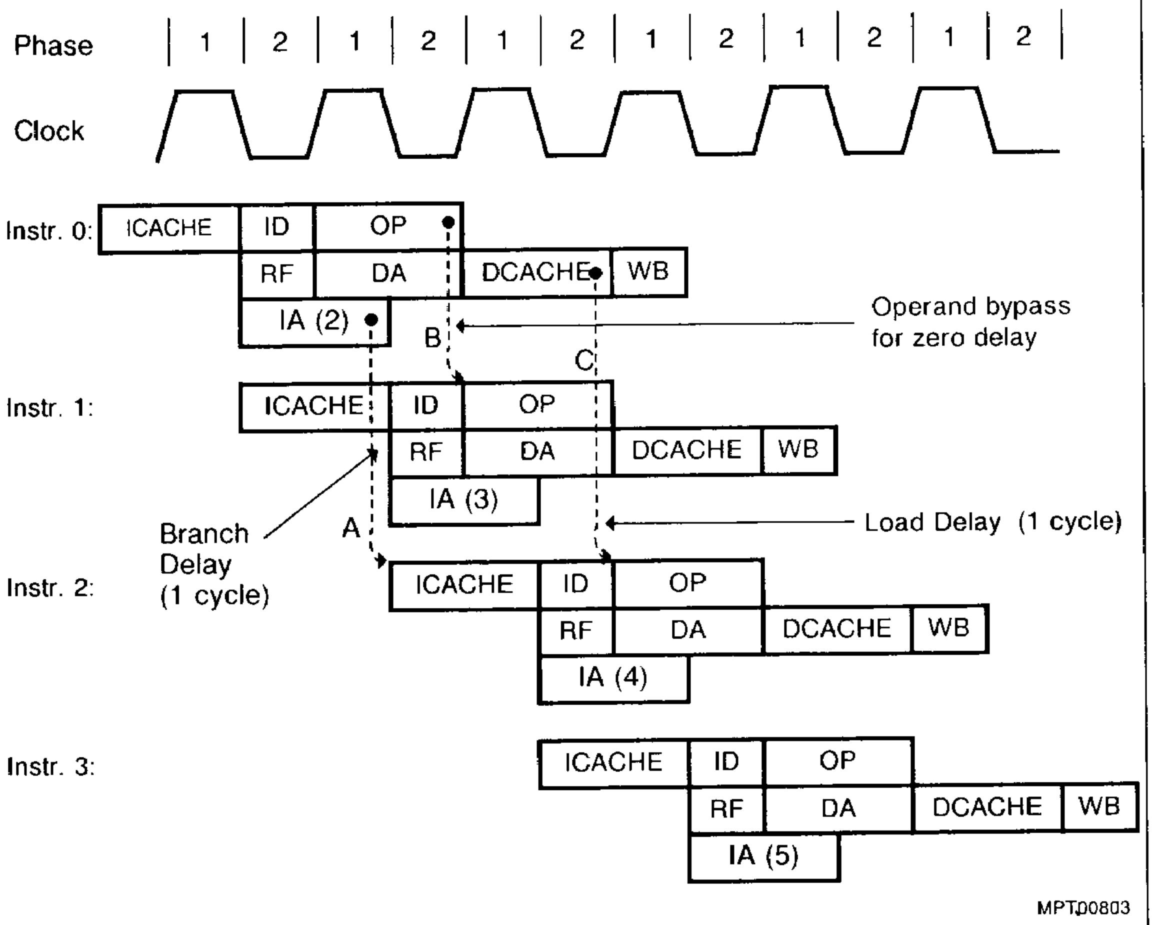
Figure 5
Overlapped execution of 5 instructions



The SAB-R2000A uses a 5-stage pipeline to achieve an instruction execution rate approaching one instruction per CPU cycle. Thus the executions of five instructions at a time are overlapped as shown in figure 5. This pipeline operates efficiently because different CPU resources (address and data bus accesses, ALU operations, register accesses and so on) are utilized on a non-interfering basis. As figure 6 illustrates there is not only parallelism due to pipelining but also parallelism within the execution of a single instruction.

The clock cycle is divided into two phases. To access the external instruction and data caches (ICACHE, DCACHE) requires 1 cycle, as do major internal operations (OP, DA, IA). Instruction Decode (ID) is simple enough to occur within one phase, overlapped with Register Fetch (RF). Instruction address calculation and translation (IA) also overlaps Instruction Decode and Register Fetch, it consists of instruction virtual address calculation (IVA see figure 4) and TLB access for instruction address translation (ITLB). The Instruction address calculation and translation that is performed in the present instruction is for the second instruction following the present instruction, as shown in figure 6 (i.e. IA in Instr. 0 is for Instr. 2).

Figure 6
Instruction Pipeline Dependencies



IA : Instruction address calculation and translation
DA : Data address calculation and translation

IA is performed here especially so that a branch, for example, at Instr. 0 in figure 6 can address the ICACHE access of Instr. 2 (see dotted line A), i.e. one cycle latency. What happens is that after the condition has been evaluated (i.e. for a conditional branch), either PC + 2 or the Branch Target address is MUXed to the IF stage of the second succeeding instruction (again dotted line A). This means that if the branch is taken the branch target address is the address of Instr. 2. On the other hand, if the branch is not taken, PC + 2 (i.e. two sequential instructions after the current PC) is the address of Instr. 2. Data address calculation and translation (DA) is similar to IA. Similarly, a load at Instr. 0 fetches data that are immediately used by the OP of Instr. 2 (dotted line C), while an ALU/Shift result gets passed directly into Instr. 1 with no delay (dotted line B). This tight coupling between instructions makes for a highly efficient pipeline. Note that the IA-ICACHE and DA-DCACHE cycles are displaced by one phase, so that the corresponding TLB and cache accesses can be interleaved on a single set of buses (see also figure 4).

As shown the SAB-R2000A uses a number of techniques internally to enable execution of all instructions in a single cycle, such as bypassing between instructions in the pipeline to keep the latency of branches and memory references to 1 cycle and to allow ALU results to be used in the succeeding instruction. However, there are two categories of instructions whose special requirements could disturb the smooth flow of instructions through the pipeline:

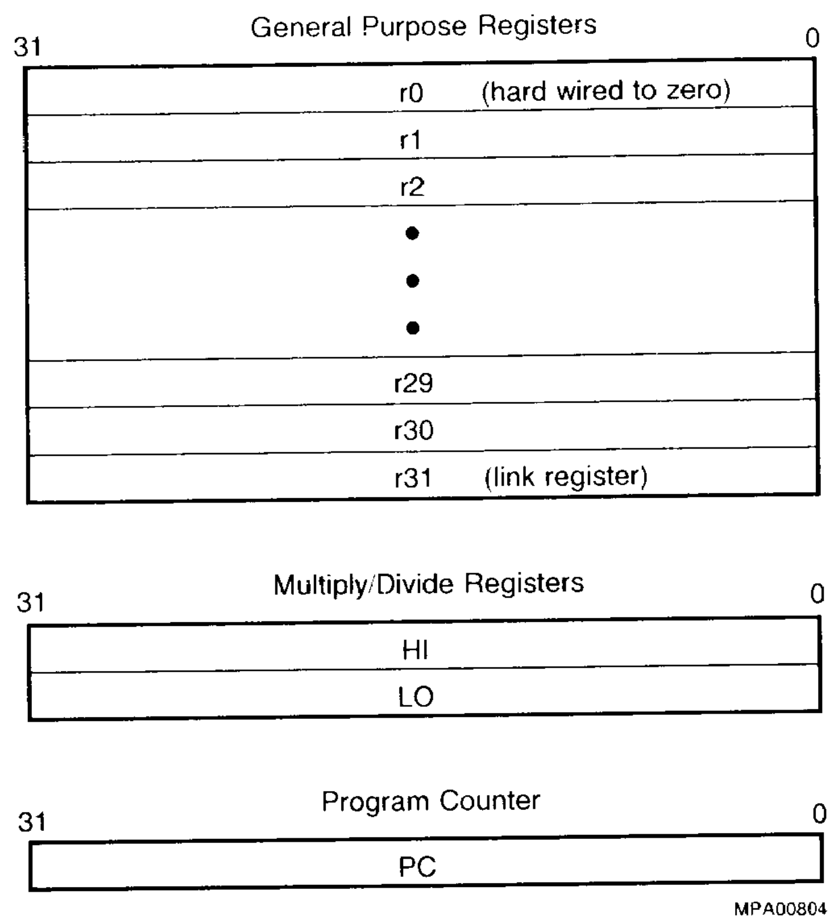
- **Load** instructions have a delay, or latency, of one cycle before the data being loaded is available to another instruction.
- **Jump and Branch** instructions also have a delay of one cycle while they fetch the instruction and target address if the branch is taken.

The SAB-R2000A continues execution despite the inherent 1-cycle delay. Loads, Jumps and Branches do not interrupt the normal flow of instructions through the pipeline, i.e. the pipeline is not stalled. The processor always executes the instruction immediately following one of these "delayed instructions". Instead of having the processor manage these pipeline delays, the SAB-R2000A turns over the responsibility for dealing with "delayed instructions" to software. Thus the assembler must insert an appropriate instruction immediately following a "delayed instruction". It also has the responsibility of ensuring that the inserted instruction has no dependencies relating to the "delayed instruction". In the SAB-R2000A architecture only multi-cycle delays (e.g. waiting for SAB-R2010A results, MUL/DIV results, etc.) are handled by hardware interlocks - 1-cycle delays, as described, are handled much more efficiently by software for all implementations (thus guaranteeing scalability).

CPU Registers

There are 32 general purpose 32-bit registers, two 32-bit registers that hold the results of integer multiply and divide operations, and a 32-bit program counter as shown in figure 7.

Figure 7
CPU Registers



The 32 General Purpose Registers are treated symmetrically, with two exceptions – r0 is hardwired to a zero value, and r31 is the link register for Jump And Link instructions. Register r0 may be specified as a target register for any instruction when the result of the operation is discarded. The register maintains a value of zero under all conditions when used as a source register.

The two Multiply/Divide registers (HI, LO) store the double-word, 64-bit result, of multiply operations or the quotient and remainder of divide operations.

The Program Counter contains the current virtual address of the next instruction to be executed.

There is no condition code register. If an instruction generates a condition, the corresponding flags are stored in a general purpose register. This means that the pipeline is freed of any special mechanisms to by-pass condition codes, interlock on them, or abort writing them on exceptions. Instead the methods already implemented to deal with register-value dependencies are employed. Further, conditions mapped onto the register file are subject to the same compile-time optimizations in allocation and reuse as other register variables.

There is also no Program Status Word (PSW) register - the functions traditionally provided by a PSW are instead provided in the Status and Cause registers incorporated within the CP0. CP0 has a number of special purpose registers that are used in conjunction with the memory management system and during exception processing. These will be explained in the CP0 section.

Data Formats

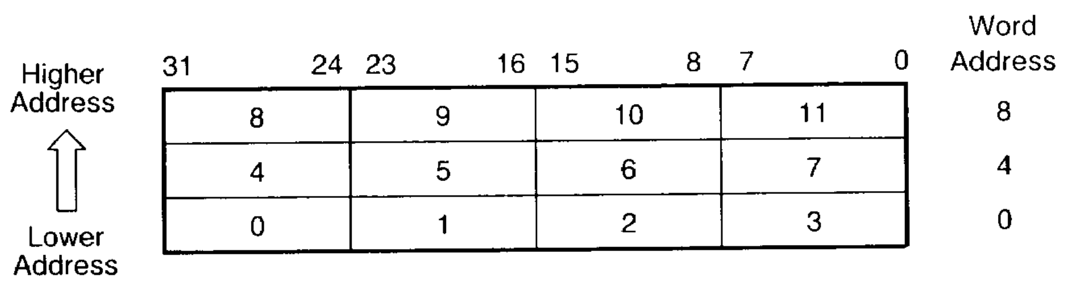
The SAB-R2000A defines signed/unsigned 32-bit words, 16-bit half-words and 8-bit bytes. The byte ordering is configurable either little-endian (iAPX x86®, NS32000®, DEC VAX®) or big-endian (MC680x0®, IBM 370®) byte ordering. Hence it is compatible with existing databases generated by machines that access bytes in either order.

Bit 0 is always the least significant (rightmost) bit, thus bit designations are always little-endian (although no instructions explicitly designate bit positions within words). Figures 8 and 9 show the ordering of bytes within words, and the ordering of words within multiple-word structures for the big-endian and little-endian conventions.

Special instructions are provided for addressing words that are not aligned on 4-byte (word) boundaries (Load/Store-Word-Left/Right; LWL, LWR, SWL, SWR). These instructions are used in pairs to provide addressing of misaligned words with one additional instruction cycle over that required for aligned words.

iAPX x86® is a trademark of INTEL
NS32000® is a trademark of National Semiconductor
DECVAX® is a trademark of Digital Equipment Cooperation
MC680x0® is a trademark of Motorola Inc.
IBM 370® is a trademark of IBM Cooperation

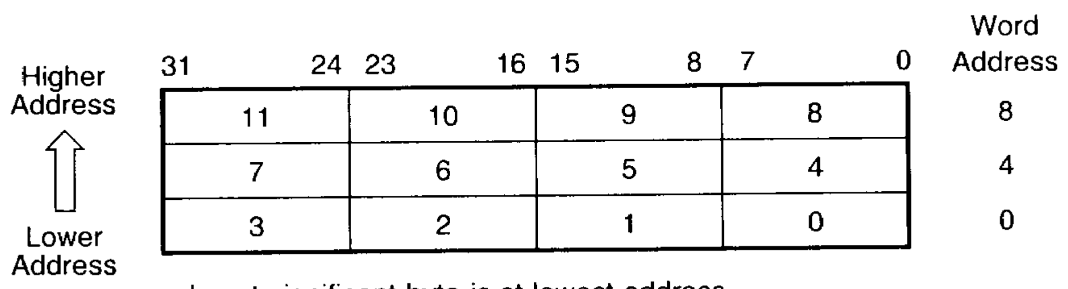
Figure 8
Addresses of Bytes within Words: Big Endian



- Most significant byte is at lowest address.
- Word is addressed by byte address of most significant byte.

MPA00805

Figure 9
Addresses of Bytes within Words: Little Endian



- Least significant byte is at lowest address.
- Word is addressed by byte address of least significant byte.

MPA00806

Addressing

The SAB-R2000A uses byte addressing, with alignment constraints for half-word and word accesses; half-word accesses must be aligned on an even byte boundary and word accesses must be aligned on a byte boundary divisible by four. Any attempt to address a data item that does not have the proper alignment causes an alignment exception. As said earlier special instructions are provided for addressing words that are not aligned.

The SAB-R2000A supports only one addressing mode – base register plus a signed 16-bit offset, which covers the most common case in High Level Languages and is very fast. The assembler, however, synthesizes some additional addressing modes to present more traditional addressing capabilities to the assembly language programmer.

Instruction Set Overview

All SAB-R2000A instructions consist of one 32-bit word. There are only three instruction formats (immediate, jump and register) as shown in figure 10.

The single instruction length simplifies instruction fetch and decode and eliminates the overhead for instructions crossing word and page boundaries within the memory hierarchy, thereby simplifying the interaction of instruction fetch with the virtual memory management unit. The three instruction formats ensure that opcodes and register descriptors are always found in the same bit locations. This enables register fetch to proceed in parallel with operation decode on all instructions, which is exactly what happens in the RD pipestage. More complicated (and less frequently used) operations can be synthesized by the compiler using sequences of simple instructions.

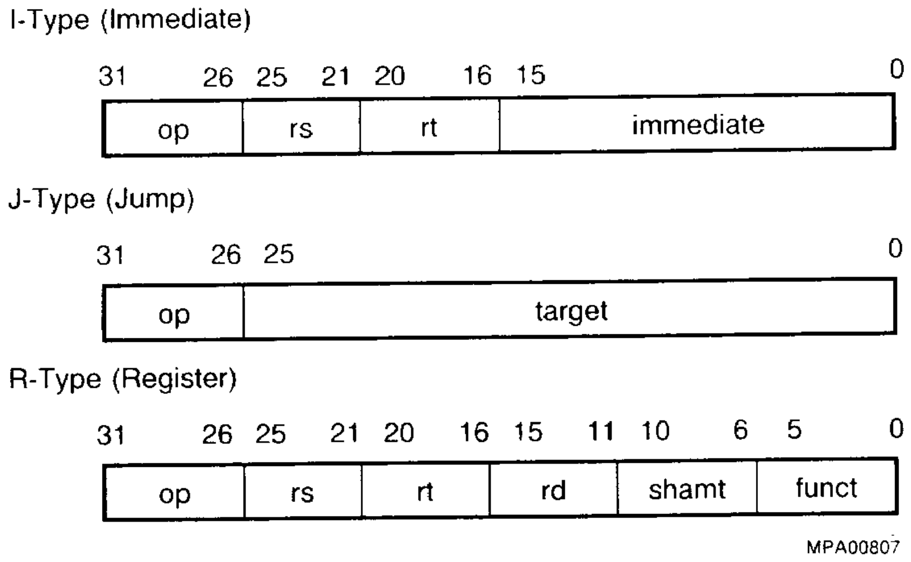
The SAB-R2000A instruction set can be divided into the following groups:

- **Load/Store** instructions move data between memory and general registers. They are all I-type instructions, since the only addressing mode supported is base register plus 16-bit, signed immediate offset. These are the only instructions which access memory.
- **Computational** instructions perform arithmetic, logical and shift operations on values in registers. They occur in both R-type (both operands and the result are registers) and I-type (one operand is a 16-bit immediate value) formats.
- **Jump and branch** instructions change the control flow of a program. Jumps are always to a paged absolute address formed by combining a 26-bit target with four bits of the Program Counter (J-type format, for subroutine calls) or 32-bit register addresses (R-type, for returns and dispatches). Branches have 16-bit offsets relative to the Program Counter (I-type). Jump and Link Instructions save a return address in Register 31.
- **Coprocessor** instructions perform operations in the coprocessors. Coprocessor Loads and Stores are I-type. Coprocessor computational instructions have coprocessor-dependent formats (see SAB-R2010A data sheet).

- **Coprocessor 0** instructions perform operations on the System Control Coprocessor (CP0) registers to manipulate the memory management and exception handling facilities of the processor.
- **Special** instructions perform a variety of tasks, including movement of data between special and general registers, system calls and breakpoint. They are always R-type.

A detailed summary of the instruction set is provided in the Instruction Set Summary section.

Figure 10
Instruction Formats



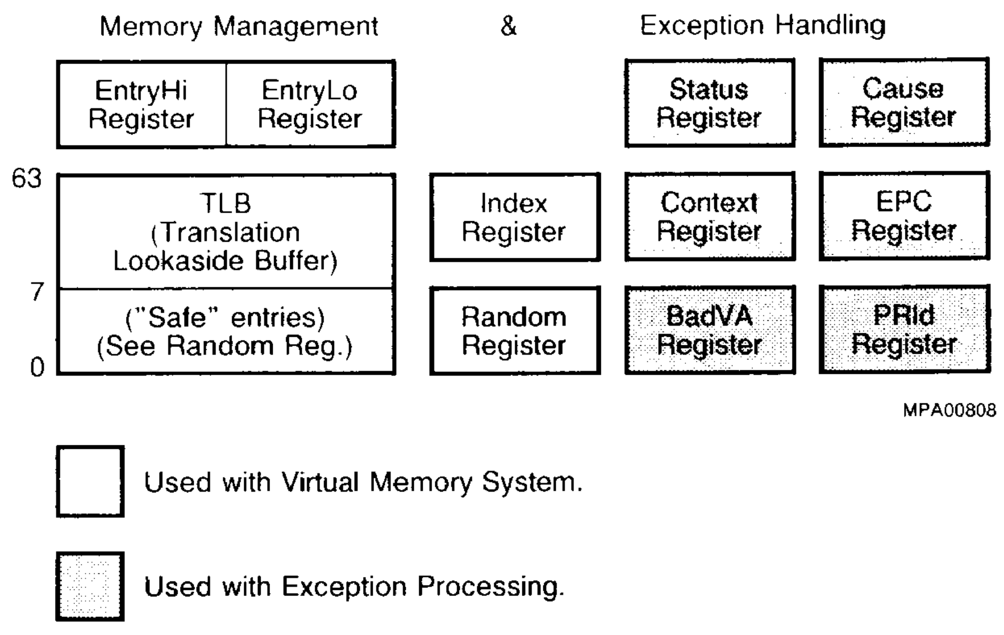
where:

op	:	is a 6-bit operation code
rs	:	is a 5-bit source register specifier
rt	:	is a 5-bit target (source/destination) register or branch condition
immediate	:	is a 16-bit immediate, branch displacement or address displacement
target	:	is a 26-bit jump target address
rd	:	is a 5-bit destination register specifier
shamt	:	is a 5-bit shift amount
funct	:	is a 6-bit function field

System Control Coprocessor

The SAB-R2000A CPU supports up to 4 coprocessors (designated CP0 through CP3), which are tightly coupled co-execution units that share a single instruction stream with the CPU. The System Control Coprocessor (CP0), is incorporated on the SAB-R2000A chip and supports the virtual memory system and exception handling functions (traps, interrupts, memory and internal operation faults) of the SAB-R2000A. CP0 incorporates a 64-entry TLB plus the registers shown in figure 11.

Figure 11
The CP0 Registers



The virtual memory system is implemented using a TLB and a group of programmable registers as shown in figure 11. The other registers shown are used to perform the exception handling capabilities. Table 1 provides a brief description of each register. In this table the number of each register is given. Logically the registers are numbered from 0 to 31. The numbers not contained in the table are unused.

Table 1
System Control Coprocessor (CP0) Registers

Register	Description	Number
EntryHi	High half of a TLB entry	10
EntryLo	Low half of a TLB entry	2
Index	Programmable pointer into TLB array	0
Random	Pseudo-random pointer into TLB array	1
Status	Mode, Interrupt enables, and diagnostic status info	12
Cause	Indicates nature of last exception	13
EPC	Exception Program Counter	14
Context	Pointer into kernel's virtual Page Table Entry array	4
BadVA	Most recent bad virtual address	8
PRId	Processor revision identification	15

Access to these registers and CP0 specific instructions (e.g. MTC0) may be restricted by setting CP0 to an "unusable" state (see status register). When the processor is executing in Kernel mode the usability of CP0 is ignored, i.e. it is always considered usable. This means that requests to manipulate CP0 from Kernel mode are always granted. However, it is possible for the Kernel to grant unrestricted access to CP0 registers by setting it to a usable state. Kernel and User modes are described in the next section.

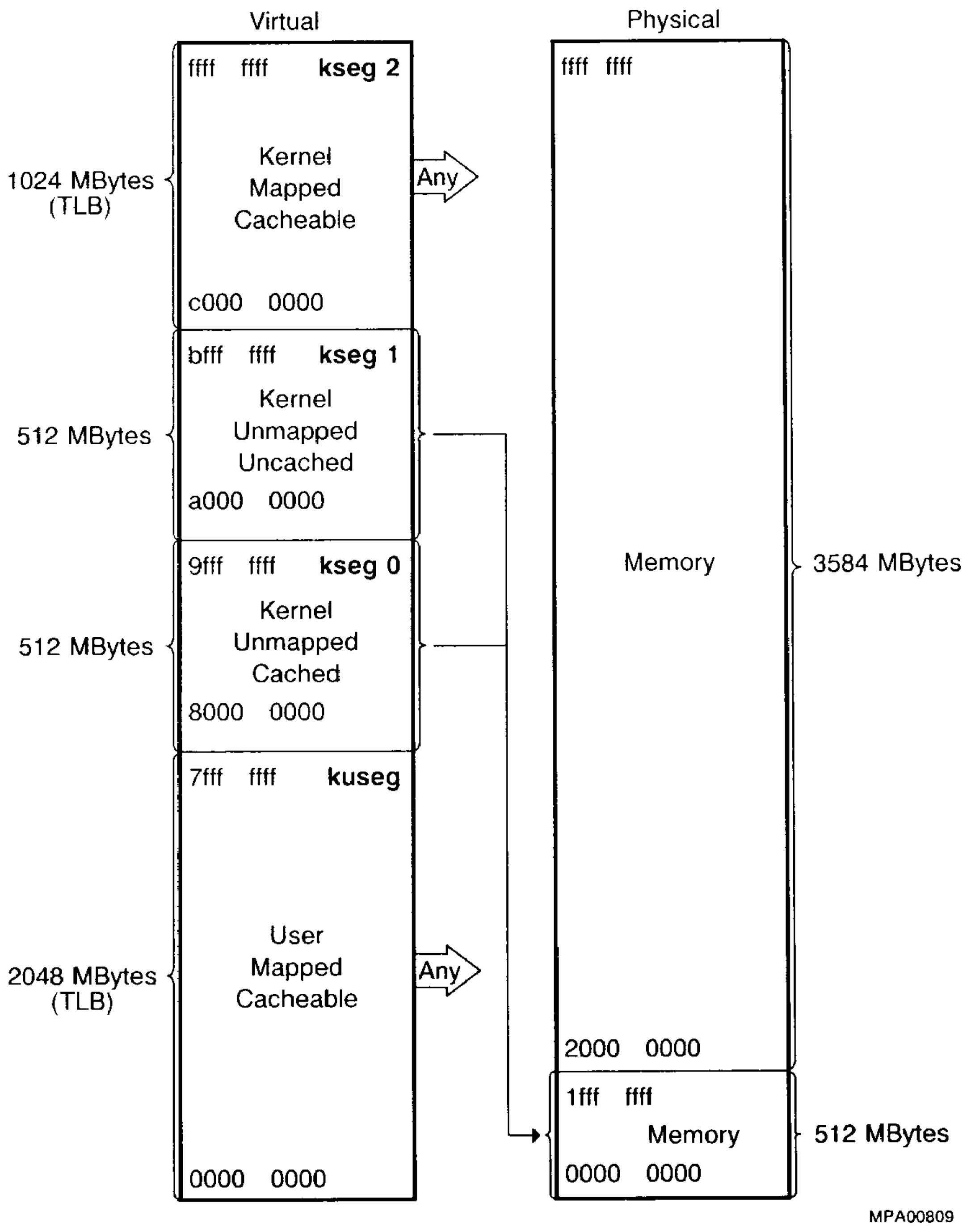
Memory Management System

The SAB-R2000A has an addressing range of 4 Gbytes. Since most systems implement physical memory sizes under 4 Gbytes, the SAB-R2000A's virtual memory system logically expands the available physical memory space by translating addresses composed in a large virtual address space into the physical memory space. All mapping is performed by the TLB in CP0. The TLB is fully associative and maps 64 4-Kbyte pages, sharable among user processes with minimal context-switch overhead, due to the PID (Process Identifier) number associated with each TLB entry.

Operating Modes

There are two operating modes in the SAB-R2000A, the Kernel mode and the User mode. The processor normally operates in User mode until an exception is detected forcing it into Kernel mode. It remains there until a Restore From Exception (rfe) instruction is executed.

Figure 12
Virtual Memory Map



Address mapping is different for Kernel and User modes. The User mode address space is a subset of the Kernel mode address space – i.e. the current User process owns a linear 2 Gbyte address space that is included in a 4 Gbyte Kernel address space. Figure 12 shows the virtual-to-physical memory map for both User mode and Kernel mode segments.

User Mode Virtual Addressing

When the processor is operating in User mode, a single, uniform virtual address space (kuseg) of 2 Gbytes is available to the current user process. All valid User mode virtual addresses have the most significant bit cleared to 0 – i.e. address references above 0x7FFFFFFF (0x means hex in a "C" type notation) cause an Address Error exception. All references to kuseg are mapped through the TLB, which also controls the cacheability of an access (the N-bit in a TLB entry). In Kernel mode, references to kuseg are treated just like User mode references, streamlining Kernel access to User data. Kuseg is typically used to hold all User code and data, and the current User process typically resides here, plus shared libraries in systems that have them.

Kernel Mode Virtual Addressing

In Kernel mode, three distinct virtual address spaces (in addition to kuseg) are available.

Kernel Cached, Unmapped – kseg0:

This segment is 512 Mbytes long starting at 0x80000000. References within kseg0 are direct-mapped onto the first 512 Mbytes of physical address space, use cache memory, but do not use TLB entries. That is to say that the physical address selected is defined by subtracting 0x80000000 from the virtual address (in order to directly map it into the first 512 Mbytes of physical memory). Typically some Kernel data and some of its executable code are kept here.

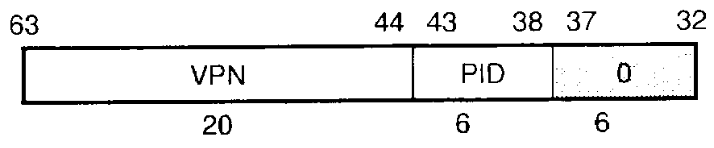
Kernel Uncached, Unmapped – kseg1:

Kernel segment kseg1 begins at 0xa0000000 and is also 512 Mbytes long. Like kseg0 it is direct-mapped onto the first 512 Mbytes of physical address space using no TLB entries. That is to say that the physical address selected is defined by subtracting 0xa0000000 from the virtual address. Unlike kseg0, it uses uncached references. An operating system typically uses kseg1 for I/O registers, ROM code and disk buffers.

Kernel Mapped – kseg2:
This segment is 1 Gbyte long, beginning at 0xc0000000. Like kuseg, it uses TLB entries to map virtual addresses to arbitrary physical ones, with or without caching (N-bit in a TLB entry). Operating systems typically use kseg2 for Kernel stacks and pre-process data that it must remap on context switches, for User page tables (memory maps), and some dynamically allocated data areas.

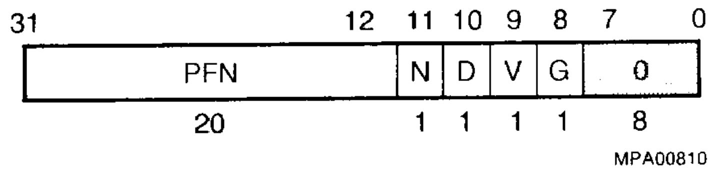
Figure 13
TLB EntryLo & EntryHi Registers

TLB EntryHi Register



VPN	:	Virtual Page Number. Bits 31. .12 of virtual address.
PID	:	Process ID field. A 6-bit field which lets multiple processes share the TLB while each process has a distinct mapping of otherwise identical virtual page numbers.
0	:	Reserved. Currently ignores writes, returns zero when read.

TLB EntryLo Register



PFN	:	Page Frame Number. Bits 31. .12 of the physical address. The SAB-R2000A maps a virtual page to the PFN.
N	:	Non-cacheable. If this bit is set, the page is marked as non-cacheable and the SAB-R2000A directly accesses main memory instead of first accessing the cache.
D	:	Dirty. If this is set, the page is marked as "dirty" and therefore writable. This bit is actually a "write-protect" bit that software can use to prevent alteration of data. If an entry is accessed for a write operation when the D bit is cleared, the SAB-R2000A causes a TLB Mod trap. The TLB entry is not modified on such a trap.
V	:	Valid. If this bit is set, it indicates that the TLB entry is valid; otherwise, a TLBL or TLBS Miss occurs.
G	:	Global. If this bit it set, the SAB-R2000A ignores the PID match requirement for valid translation. In kseg2, the Global bit lets the kernel access all mapped data without requiring it to save or restore PID (Process ID) values.
0	:	Reserved. Currently ignores writes, returns zero when read.

Virtual Memory Control

A high level description of the address space has been given – now the low-level details of the virtual memory system shall be discussed. The following registers which will be described are used for virtual memory control.

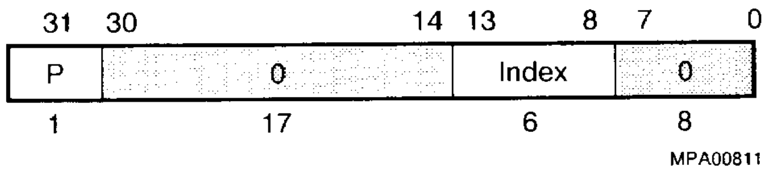
EntryHi and EntryLo:

This register pair and a single TLB entry have the same format, and so are described together. These two registers provide the data pathway (i.e are a buffer) through which the TLB is read, written or probed. When address translation exceptions occur, these registers are loaded with the relevant information about the address that caused the exception. EntryLo is the natural form of a Page Table Entry (PTE), however, since PTE's are always loaded by system software, not by the SAB-R2000A hardware, an operating system can use another format for memory resident PTE's. The register pair is illustrated in figure 12, a TLB entry is equivalent to the concatenation of these two registers.

Index Register:

The Index register is a 32-bit, read/write register, which has a 6-bit Index field. This field runs from 0 to 63, indexes an entry in the TLB and is used as a subscript to read or write any TLB entry. The high-order bit shows the success or failure of a TLB Probe (tlbp) Instruction. Figure 14 shows the format of the Index register.

Figure 14
Index Register



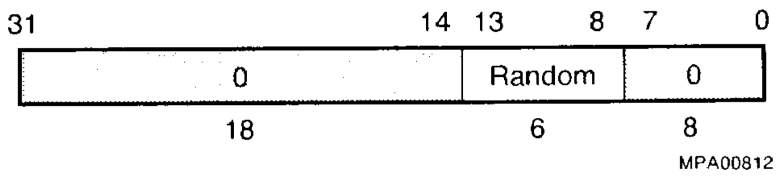
P	:	Probe failure. Set to 1 when the last TLBProbe (tlbp) instruction was unsuccessful.
Index	:	Index to the TLB entry that will be affected by the TLBRead and TLBWrite instructions.
▨ 0	:	Reserved. Currently ignores writes, returns zero when read.

Random Register:

The Random register is a 32-bit register, which has a 6-bit Random field which indexes a random entry in the TLB. The Random field is decremented every machine cycle, but is constrained to the range from 8 to 63. It is used to speed up pseudo-random TLB refill under Operating System control. The TLB Write Random (tlbwr) instruction is used to write the TLB entry that this register indexes. The first 8 entries (0 to 7) in the TLB are "safe" entries (see figure 11) because a "tlbwr" instruction can never replace the contents of these entries. Typically these 8 entries are reserved by the Operating System.

The contents of this register can be read to verify proper operation of the process (not normally required). To further simplify testing, the Random field is set to a value of 63 when the SAB-R2000A is reset. Figure 15 illustrates the Random register.

Figure 15
The Random Register



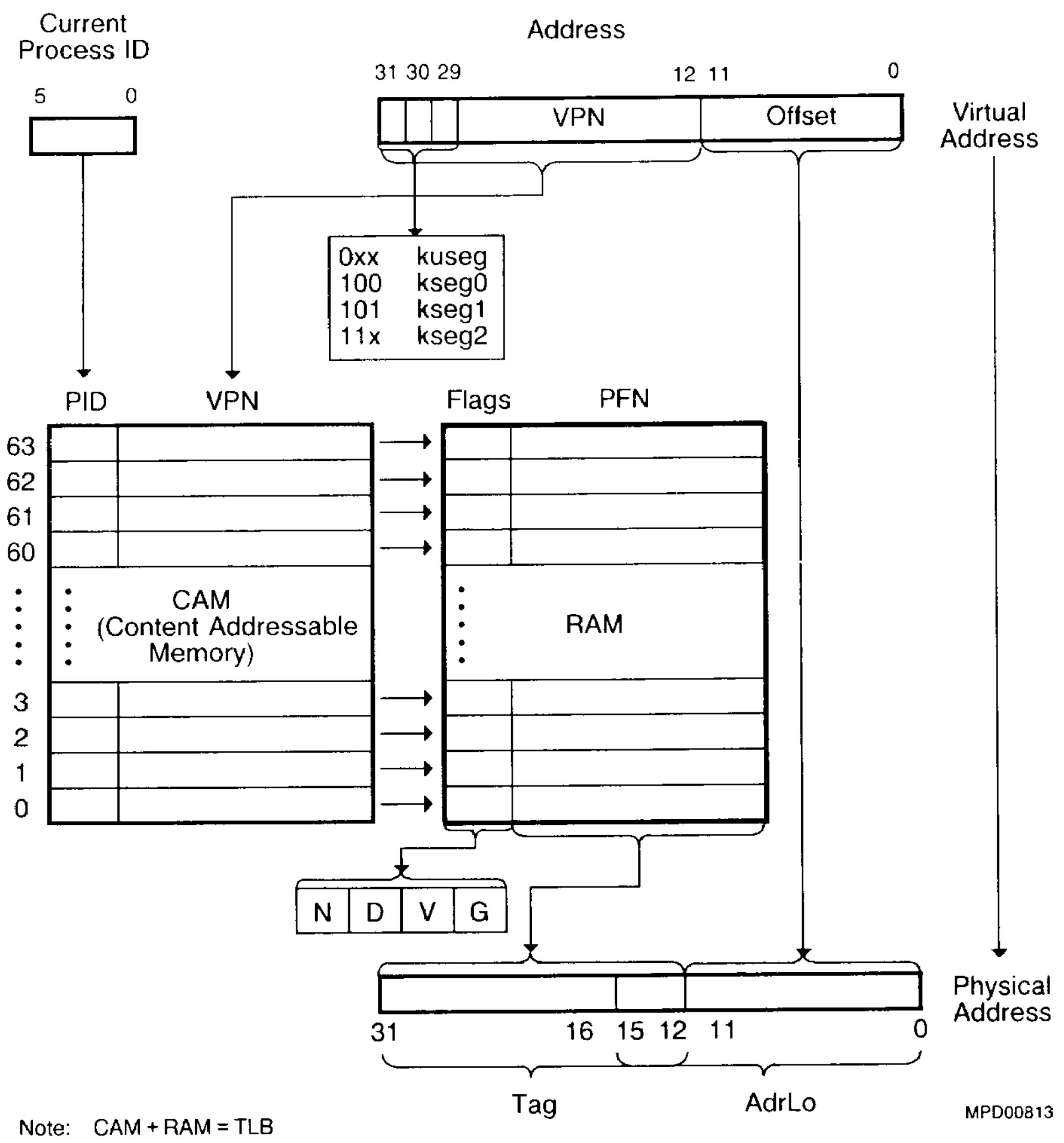
Random: A random index (with a value ranging from 8 to 63) to a TLB entry.

0 : Reserved. Currently ignores writes, returns zero when read.

Virtual Address Translation

During virtual-to-physical address translation, the processor compares the PID (Process Identification) number and the highest 20 bits (the VPN) of the virtual address to the contents of the TLB. Figure 16 illustrates the TLB address translation process.

Figure 16
TLB Operation



Note: CAM + RAM = TLB

A virtual address matches a TLB entry when:

- the Virtual Page Number (VPN) field of the virtual page address matches the VPN field of the TLB entry and,
- either the Global (G) bit of the TLB entry is set, or the PID field of the virtual address (as held in the EntryHi register) matches the PID field of the TLB entry.

While the Valid (V) bit of the TLB entry must be set for a valid translation to take place, it is not involved in the determination of a matching TLB entry.

If a TLB entry matches, the physical address and access control bits (N, D and V) are retrieved from the matching TLB entry. Otherwise a TLB miss (reference to kseg2), or a UTLB miss (reference to kuseg) exception occurs. If the V bit is not set, a TLB miss exception is taken. Finally, if the access is a Write and the Dirty (D) bit is not set, a TLB modification exception occurs. If the Non-cacheable (N) bit is set, the physical address that is retrieved is used to access main memory, bypassing the cache.

Exception Handling System

The term exception is used for any infrequent or exceptional event that causes the processor to make a temporary transfer of control from its current process to another process that services the event. There are two main classes of exceptions in the SAB-R2000A:

- machine exceptions, such as program traps, overflow and address translation exceptions.
- external asynchronous exceptions, which include six maskable external hardware interrupts, bus error and reset.

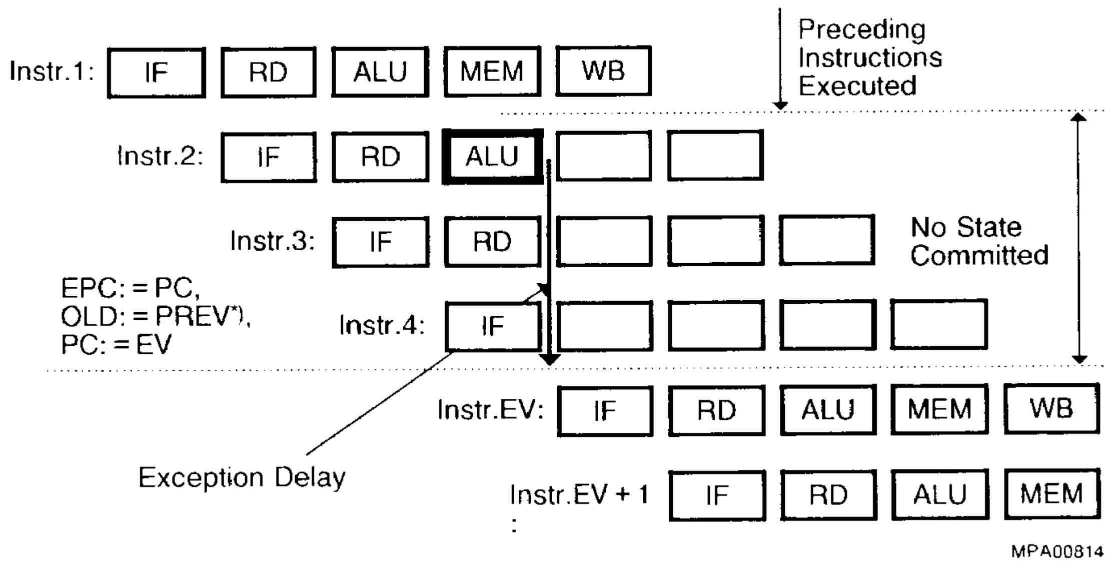
There are eight interrupts available, the hardware generates six and software two.

The exception handling system is responsible for efficiently handling relatively infrequent events, such as TLB misses, arithmetic overflow, I/O interrupts and system calls. On detection of an Exception the SAB-R2000A suspends the normal sequence of instruction execution; the processor exits User mode and is forced into Kernel mode where it can respond to the exceptional event. These events interrupt the normal execution flow by aborting the instruction which causes the exception and all those following in the pipeline which have already begun execution. The Exception Program Counter (EPC) is loaded with the appropriate restart location where execution should resume after the exception has been serviced. The restart location in the EPC is the address of the instruction which caused the exception or, if the instruction was executing in a branch delay slot, the address of the branch or jump instruction immediately preceding the delay slot. The SAB-R2000A then performs a direct jump into a designated handler routine.

A minimal amount of state is saved in the CP0 registers in order to facilitate the analysis of the cause of the exception, the servicing of the event which caused it, and the resumption of the original flow of execution.

The pipelined nature of the SAB-R2000A complicates the exception handling system. Exceptions which occur late in the pipeline effectively necessitate aborting instructions which have already begun execution but which logically should execute after the exception handling routine. Even taking this into account all SAB-R2000A exceptions are precise. That is, each exception is handled in a way that reflects serial completion of all instructions prior to the exception; the instruction which causes it and all those that follow are aborted and can be re-executed after servicing the exception. What this means (referring to figure 17) is that for example when an exception occurs in the ALU stage of Instr. 2, that Instr. 3 and 4, which have already started execution in the pipeline, do not alter the state of the machine so that execution may always properly resume after servicing the exception. Precise exception handling is shown in figure 17.

Figure 17
Precise Exception Handling

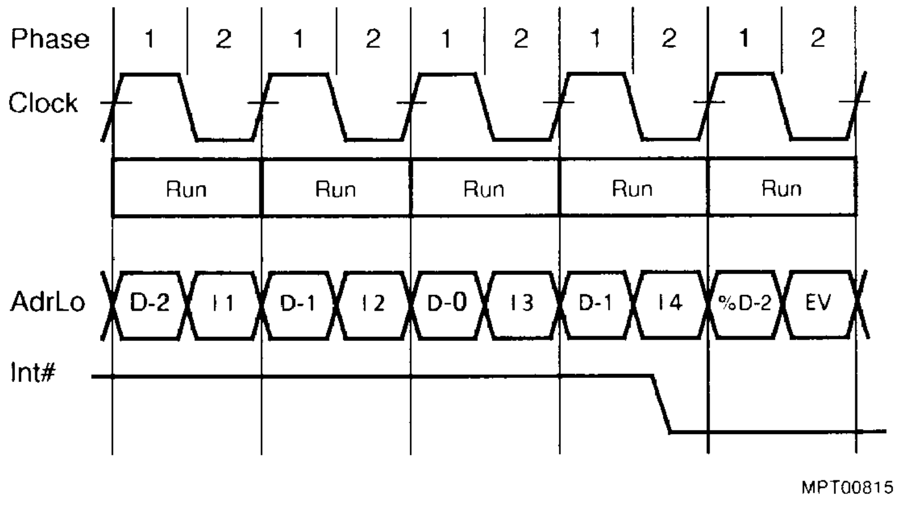


*) Refer to KU and IE bits of Status Register
EV: Exception Vector
EPC: Exception PC

Even though the machine is relatively deeply pipelined, exceptions are reported synchronously, so that all exceptions for a particular instruction are reported prior to exceptions for its succeeding instructions. Said another way, all exceptions are reported as if the processor was not pipelined.

There is only a one-cycle delay from when an exception occurs to when the exception handling routine is started (i.e. reaction time). This can be seen in figure 17, i.e. an exception occurs in the ALU cycle of Instr.2 (e.g. Overflow) and one cycle later the first instruction of the servicing routine is started (i.e. Instr.EV). This can be seen on the AdrLo bus for an Interrupt exception as is shown in figure 18. In the cycle after Int# (indicating an external interrupt exception) has been asserted the Exception Vector (EV) address is on the address bus.

Figure 18
Interrupt Exception Latency

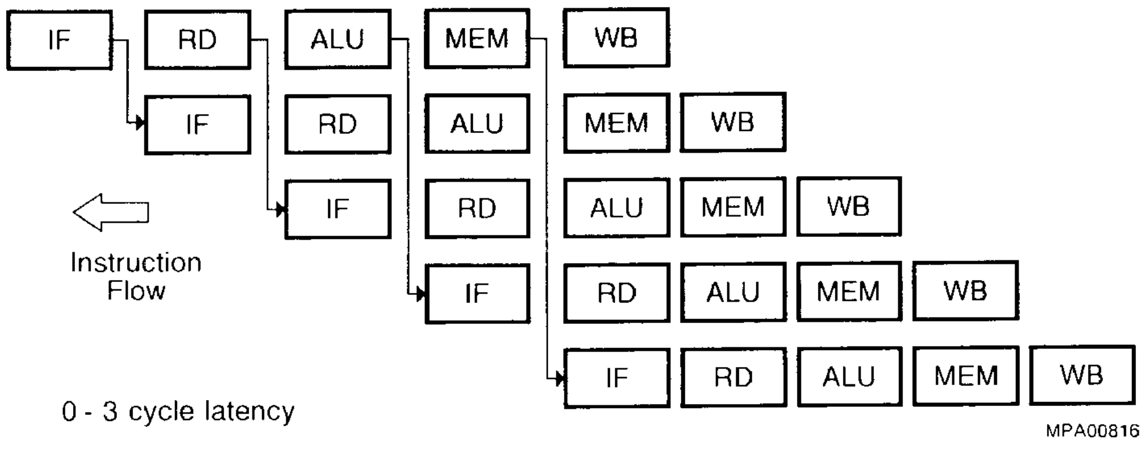


Where:

D-n	is the Data Transaction for Instr. n
In	is the Instruction Fetch for Instr. n
%	means an incorrect datum.
EV	Exception Vector

There is also an associated exception latency. This depends on how late the exception occurs in the pipeline. The exception latency can be 0-3 cycles in duration. It is the number of instructions which were aborted, that had already begun execution in the pipeline but which logically should execute after the exception handling routine – as discussed earlier. The concept of Exception latency is illustrated in figure 19.

Figure 19
Exception Latency



Exception Types

Table 2 lists each of the exception types which are handled by the processor, giving a short description of each.

There are only 3 exception vectors provided, one for Reset and one for UTLB miss exceptions (0xbfc00000 and 0x80000000 respectively), each of the remaining exceptions causes execution to resume at the General exception vector (0x80000080). When the BEV bit of the Status Register is set, the UTLB miss exception vector address is changed to 0xbfc00100 and the General exception vector address is changed to 0xbfc00180.

Table 2
SAB-R2000A Exceptions

Maskable	Exception	Mnemonic	Cause	Exception Vector
NO	Reset	Reset	Assertion of SAB-2000A's Reset signal causes an exception that transfers control to the special vector at virtual address 0xbfc00000.	0xbfc00000
	UTLB miss	UTLB	User TLB miss. A reference is made (in either User mode or Kernel mode) to a page in kuseg that has no matching TLB entry.	0x80000000 or 0xbfc00100
	TLB miss	TLBL (load) TLBS (store)	A referenced TLB entry's Valid bit isn't set or there is a reference to a *kseg2* page that has no matching TLB entry.	0x80000080 or 0xbfc00180
	TLB modified	Mod	During a store instruction, the Valid bit is set but the Dirty bit is not set.	
	Bus Error	IBE (Instruction) DBE (data)	Assertion of the SAB-R2000's BERR# signal due to such external events as bus timeout, backplane bus parity errors, invalid physical addresses or invalid access types.	
	Address Error	AdEL (load) AdES (store)	Attempt to load, fetch, or store an unaligned word; that is, a word or halfword at an address not evenly divisible by 4 or 2, respectively. Also caused by reference to a virtual address with most significant bit set while in User mode	
	Overflow	Ovf	Two's complement overflow during add or subtract.	
	System call	Sys	Execution of the syscall instruction.	
	Breakpoint	Bp	Execution of the break instruction.	
	Reserved Instruction	RI	Execution of an instruction with an undefined or reserved major operation code (bits 31..26), or a special instruction whose minor opcode (bits 5..0) is undefined.	
	Coprocessor Unusable	CpU	Execution of a coprocessor instruction when the CU (Coprocessor Usable) bit is not set for the target coprocessor.	
YES	Interrupt	Int	Assertion of one of the SAB-R2000A's six hardware interrupt inputs or setting of one of the two software interrupt bits in the Cause register.	

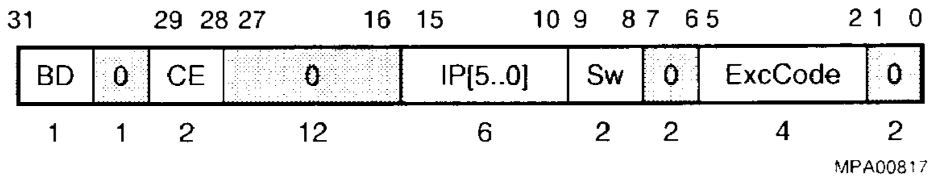
The Exception Handling Registers

The CP0 registers shown on the right of figure 11 contain information that is related to exception processing. Software can examine these registers during exception processing to determine such things as the cause of an exception, and the state of the CPU at the time of an exception. Two other registers, the Index register and the Random register, described in the Memory Management System section, may also contain useful information when handling exceptions related to virtual memory errors.

Cause Register:

The contents of this 32-bit register describe the last exception as shown in figure 20. The ExcCode field indicates the reason for the exception as listed in table 3. The remaining fields indicate pending external interrupts (IP), pending software interrupts (Sw), which, if any, coprocessor was found unusable (CE), and the occurrence of an exception in a branch delay slot (BD). All bits in the register, excluding the Sw bits, are read only. The Sw bits can be written into, to set or reset software interrupts.

Figure 20
Cause Register



BD	:	Branch Delay. Set to 1 if last exception was taken while executing in a branch delay slot.
CE	:	Coprocessor Error. Indicates the unit number referenced when a Coprocessor Unusable Exception is taken.
IP	:	Interrupts Pending. Indicates the external interrupts that are pending. IP[5..0] = Interrupt [5..0]
Sw	:	Software Interrupts. Indicates which of the two software interupts are pending. This field may be written into to set or reset software interrupts.
ExcCode	:	Exception Code field. Described in table 3.
▨ 0	:	Reserved. Currently ignores writes, returns zero when read.

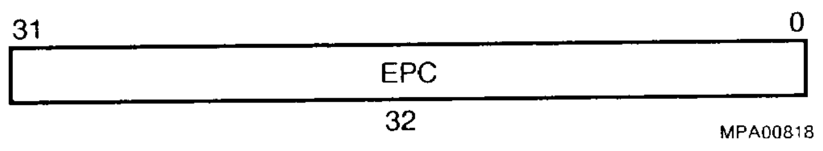
Table 3
Cause Register ExcCode Field

Number	Mnemonic	Description
0	Int	External interrupt
1	MOD	TLB modification exception
2	TLBL	TLB miss exception (Load or instruction fetch)
3	TLBS	TLB miss exception (Store)
4	AdEL	Address error exception (Load or instruction fetch)
5	AdES	Address error exception (Store)
6	IBE	Bus error exception (for an instruction fetch)
7	DBE	Bus error exception (for a data load or store)
8	Sys	Syscall exception
9	Bp	Breakpoint exception
10	RI	Reserved Instruction exception
11	CpU	Coprocessor Unusable exception
12	Ovf	Arithmetic overflow exception
13 – 15	–	reserved

Exception Program Counter Register (EPC):

On exception, this register records the address to where processing should be resumed after an exception has been serviced. The EPC register contains the virtual address of the instruction which was the cause of the exception; when that instruction is in a branch delay slot, the EPC contains the virtual address of the immediately preceding branch or jump instruction. The EPC register format is shown in figure 21.

Figure 21
EPC Register



Status Register:

The Status register is a read/write register that contains the Kernel/User mode, interrupt enable and diagnostic state of the processor, i.e. it contains all major machine status bits. All bits in the Status register, excluding TS which is read only, are readable and writable. Figure 22 shows the format of the Status Register.

Figure 22
Status Register

31 28	27 23	22	21	20	19	18	17	16	15 8	7 6	5	4	3	2	1	0
CU (Cu3..Cu0)	0	BEV *)	TS *)	PE *)	CM *)	PZ *)	SwC *)	IsC *)	IntMask Intr5..0 Sw1.0	0	KUo	IEo	KUp	IEp	KUc	IEc
4	5	1	1	1	1	1	1	1	8	2	1	1	1	1	1	1

MPA00819

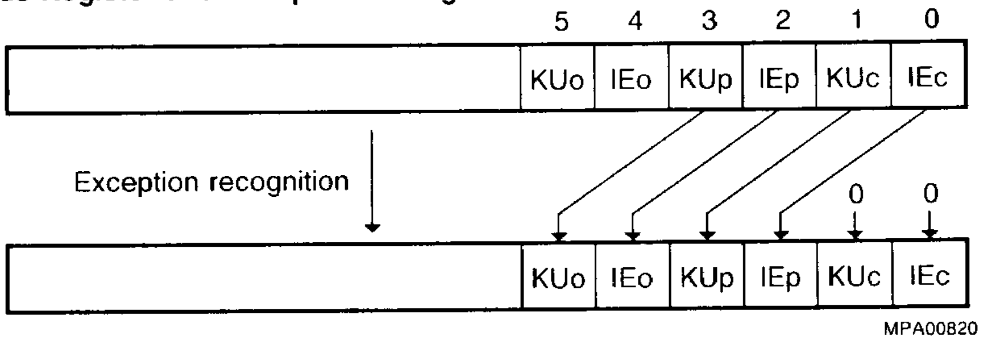
*) Indicates primary use is for diagnostics and testing.

CU	:	Coprocessor Usability. These bits control usability of the four possible coprocessors: Cu3, Cu2, Cu1, and Cu0. If a CU bit is set (= 1), that coprocessor is usable.
*) BEV	:	Bootstrap Exception Vector. If set to 1, causes the SAB-R2000A to use the alternate, bootstrap vectors for UTLB Miss and general exceptions.
*) TS	:	TLB Shutdown. Set to 1 if SAB-R2000A has disabled TLB due to catastrophic error. Cleared only by Reset.
*) PE	:	Parity Error. Set to 1 if cache parity error occurs. Reset by writing a 1 to this bit.
*) CM	:	Cache Miss. Set to 1 if most recent D-Cache load resulted in a miss (only when the D-Cache is isolated).
*) PZ	:	Parity Zero. When set to 1, causes zero to replace normal outgoing parity bits.
*) SwC	:	Swap Caches. Controls switching of control signals for I-Cache and D-Cache.
*) IsC	:	Isolate Cache. When set to 1, isolates D-Cache from main memory system.
IntMask	:	Interrupt Mask. When a bit is set to 1, the corresponding hardware interrupt [Intr# 5..0] or software interrupt [Sw1..0] is enabled.
KUo	:	Kernel/User mode, old. Set to 0 if Kernel, 1 if User.
IEo	:	Interrupt Enable, old. Set to 1 to enable, 0 to disable.
KUp	:	Kernel/User mode, previous. Set to 0 if Kernel, 1 if User.
IEp	:	Interrupt Enable, previous. Set to 1 to enable, 0 to disable.
KUc	:	Kernel/User mode, current. Set to 0 if Kernel, 1 if User.
IEc	:	Interrupt Enable, current. Set to 1 to enable, 0 to disable.
0	:	Reserved. Currently ignores writes, returns zero when read.

The Status register contains a three level stack (current, previous, and old) of the Kernel/ User mode bit (KU) and the Interrupt Enable (IE) bit.

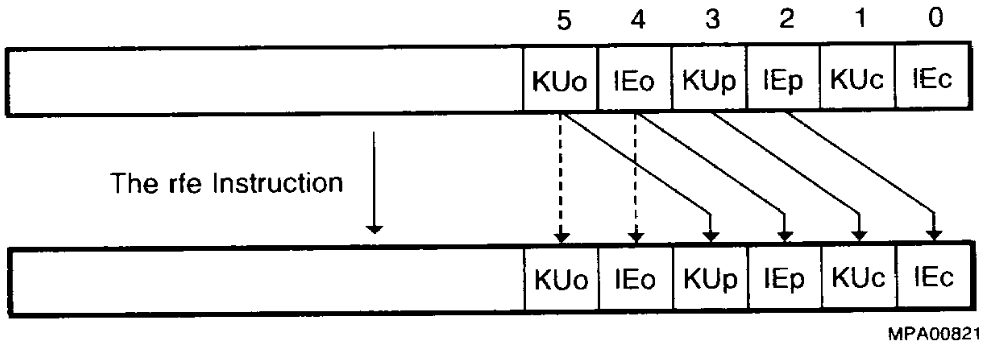
When an exception is taken the stack is pushed, i.e. the current Kernel/User mode (KUc) and current interrupt enable mode (IEc) bits are saved into the previous mode bits. The previous mode bits (KUp and IEp) are saved into the old mode bits (KUo and IEo). The current mode bits are cleared to cause the processor to enter Kernel mode and turn off interrupts. This three level set of mode bits allows the SAB-R2000A to respond to two levels of exceptions before software must save the contents of the Status register. figure 23 shows how the mode bits are pushed when an exception is taken.

Figure 23
Status Register and Exception Recognition



When an exception handler has completed execution the processor must return to the system context that existed prior to the exception. This is achieved by the Restore From Exception (rfe) instruction. The rfe instruction, when executed, pops the three level stack, i.e. the previous mode bits (KUp and IEp) are restored into the current mode bits (KUc and IEc). Likewise, the old mode bits (KUo and IEo) are restored into the previous mode bits. The old mode bits themselves remain unchanged. The actions of the rfe instruction are illustrated in figure 24.

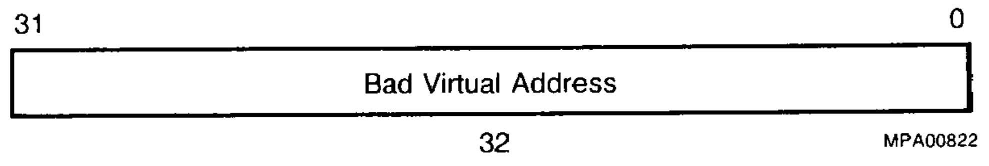
Figure 24
Restoring from Exceptions



Bad Virtual Address Register:

The BadVAddr register saves the entire bad virtual address for any addressing exception; AdeL or AdeS. Figure 25 shows this register format.

Figure 25
BadVAddr Register



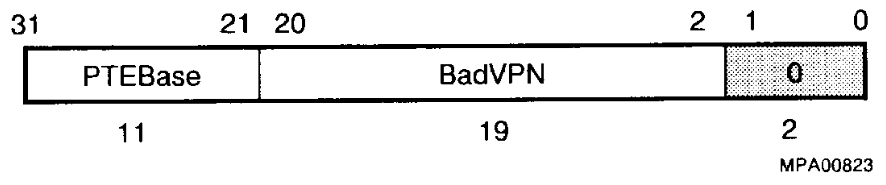
Context Register:

The Context register contains a pointer to the current user process's page map, located in kseg2 (kernel-mapped). It is designed for use in the UTLB miss handler, which loads TLB entries for normal user mode references.

The BadVPN field is not writable, it holds the VPN from the most recent virtual address for which the translation was invalid (i.e. an address exception). The 19-bit BadVPN field contains bits 30...12 (user segment Virtual Page Number) of the BadVAddr register. Bit 31 is excluded, because the UTLB miss handler is only invoked on user segment references whose highest virtual address is 0x7FFFFFFF.

The PTEBase field is writable as well as readable and indicates the base address of the page map of the current user address space. This register is implemented for the convenience of the Operating System. Figure 26 shows the format of the Context register.

Figure 26
Context Register

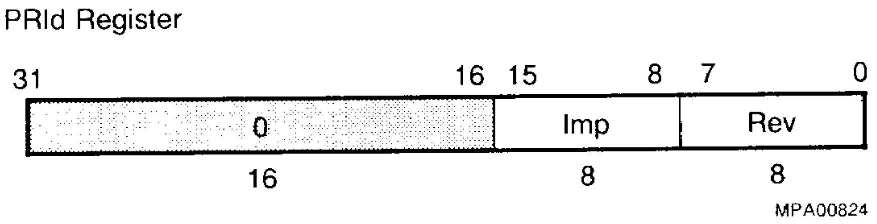


PTEBase	:	Holds the base for the Page Table Entry (set by software).
BadVPN	:	Holds the failing Virtual Page Number (set by hardware). This field is read-only and contains bits 30..12 (user-segment VPN) of the BadVAddr register.
▨ 0	:	Unused; ignored on writes, zero when read.

Processor Revision Identifier Register:

This 32-bit read only register contains information that identifies the implementation and revision level of the Processor and System Control Coprocessor. The format of the register is shown in figure 27.

Figure 27
Processor Revision Identifier Register



Imp	:	Implementation identifier.
Rev	:	Revision identifier.
0	:	Reserved. Currently ignores writes, returns zero when read.

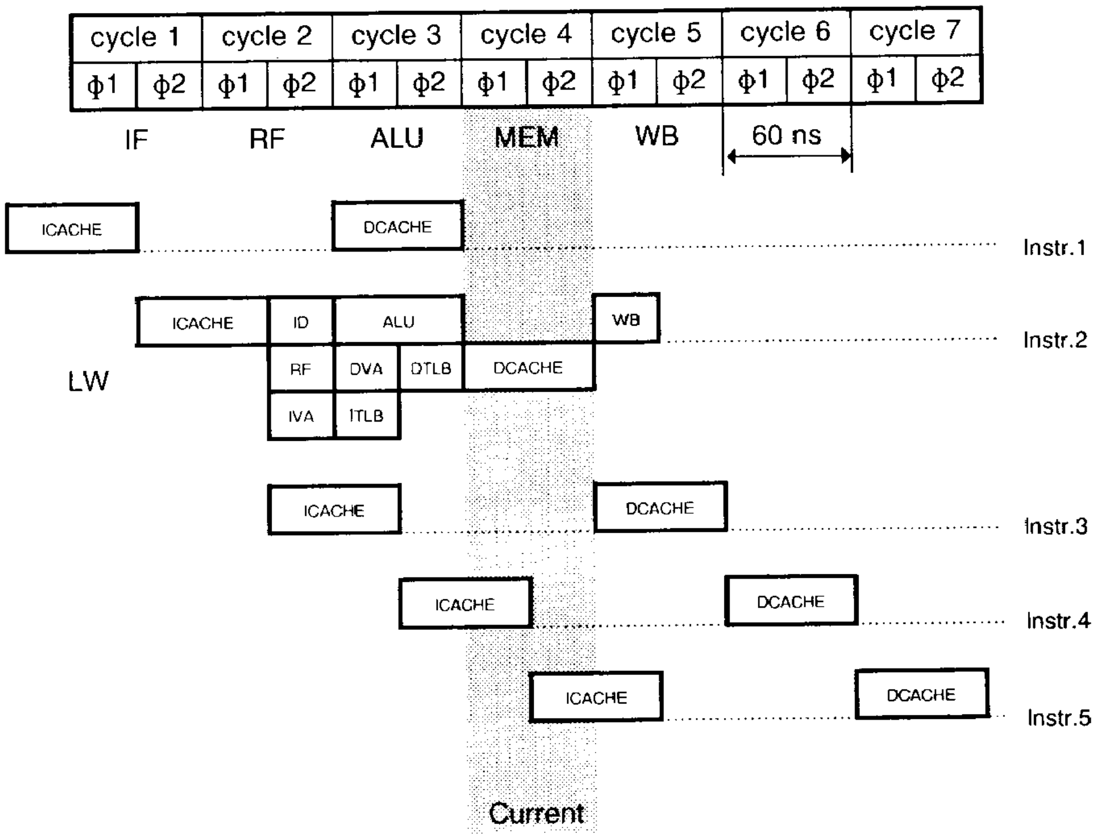
Memory System Hierarchy

The high performance capabilities of the SAB-R2000A processor places stringent demands on the memory system configuration. In order to achieve the goal of an instruction execution rate of one instruction per CPU cycle, the SAB-R2000A demands a memory bandwidth of 128 MBytes/second at 16 MHz from the memory system configuration. The memory system requirements can be seen in figure 28.

This high memory bandwidth is realized by a high performance memory hierarchy which centers on the use of external caches. Separate data and instruction caches are implemented, and the processor alternates accesses of the two caches during each CPU cycle - thus 2 words/CPU cycle, as shown in figure 28, are accessed. Both caches are physical as opposed to virtual, and ,may vary in size from 4 to 64 Kbytes each depending on the performance required, and are implemented using standard SRAM devices.

The update policy employed is a write-through policy, which simplifies the data consistency problem between cache and main memory. All data that is written to the data cache is also written out to main memory. Write buffers capture this data from the SAB-R2000A at CPU clock rates and then update main memory at its slower clock rate – therefore not stalling the processor. A simplified diagram of the high performance memory system is shown in figure 29.

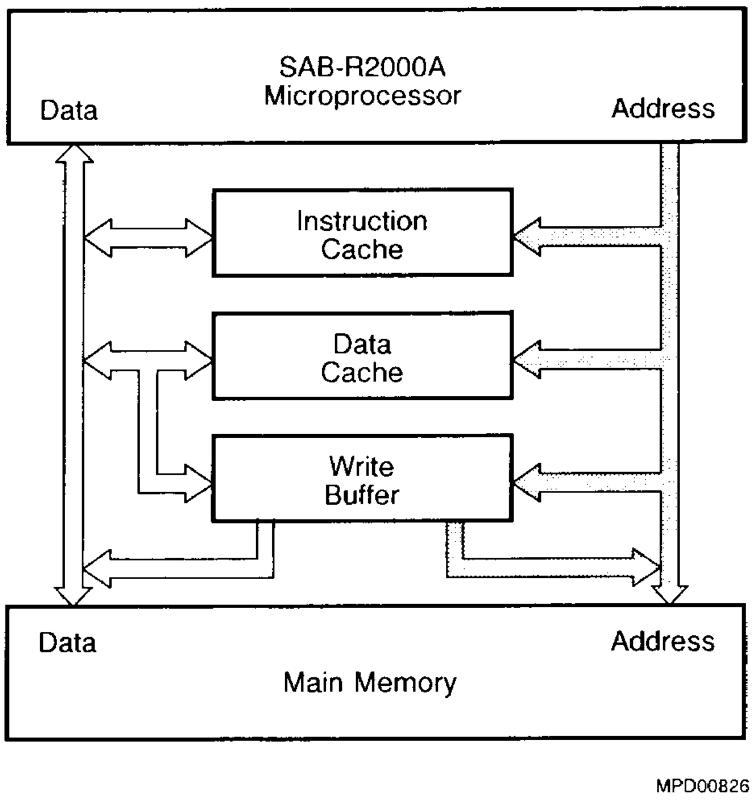
Figure 28
Memory Bandwidth



2 Words/Cycle
128 Mbytes/s at 16 MHz

MPA00825

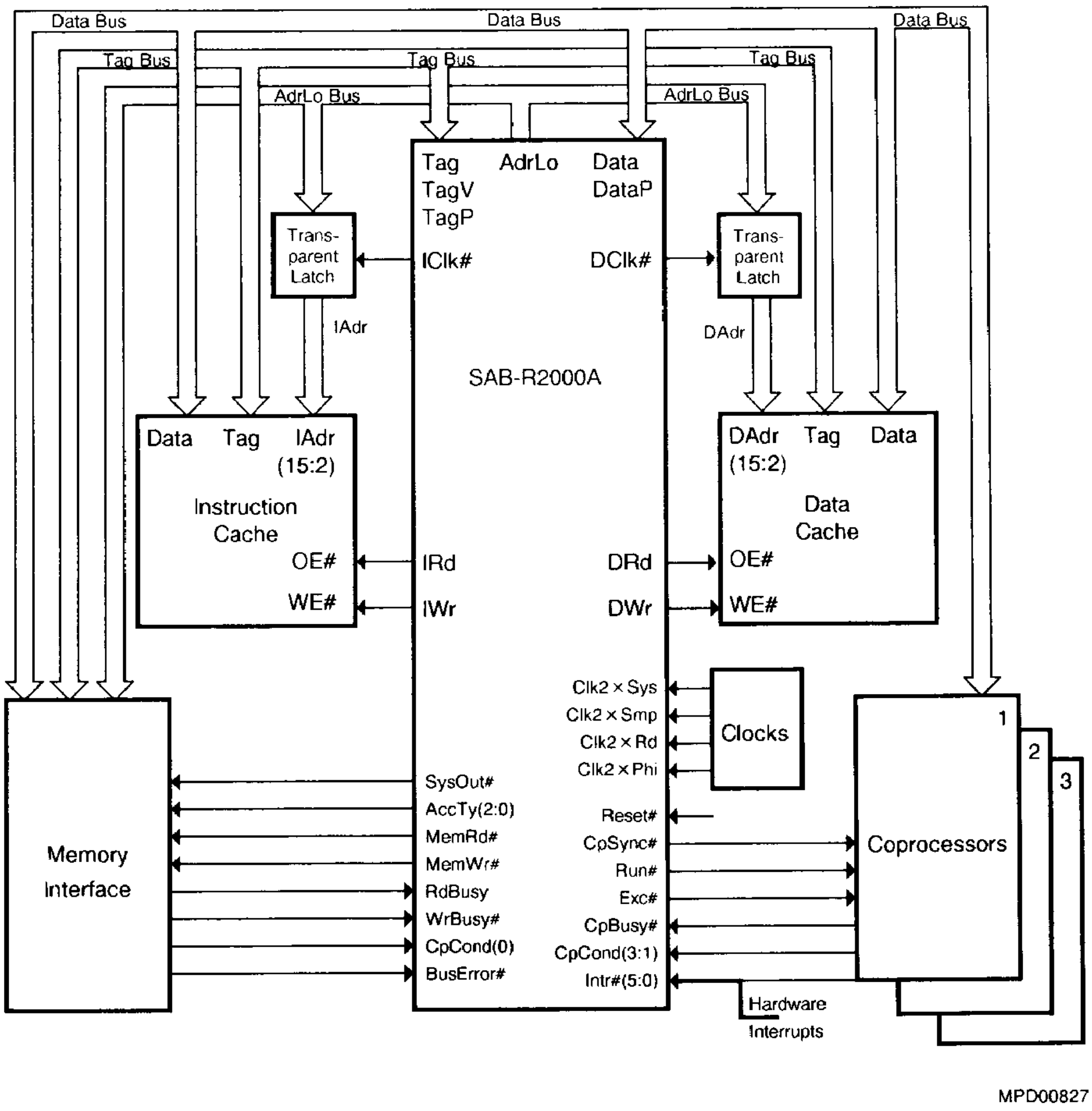
Figure 29
An SAB-R2000A System with a High-Performance Memory System



Processor Interface

The SAB-R2000A supports interfaces to external caches, main memory and coprocessors. Figure 30 illustrates the external interfaces of the SAB-R2000A processor.

Figure 30
Processor Interface

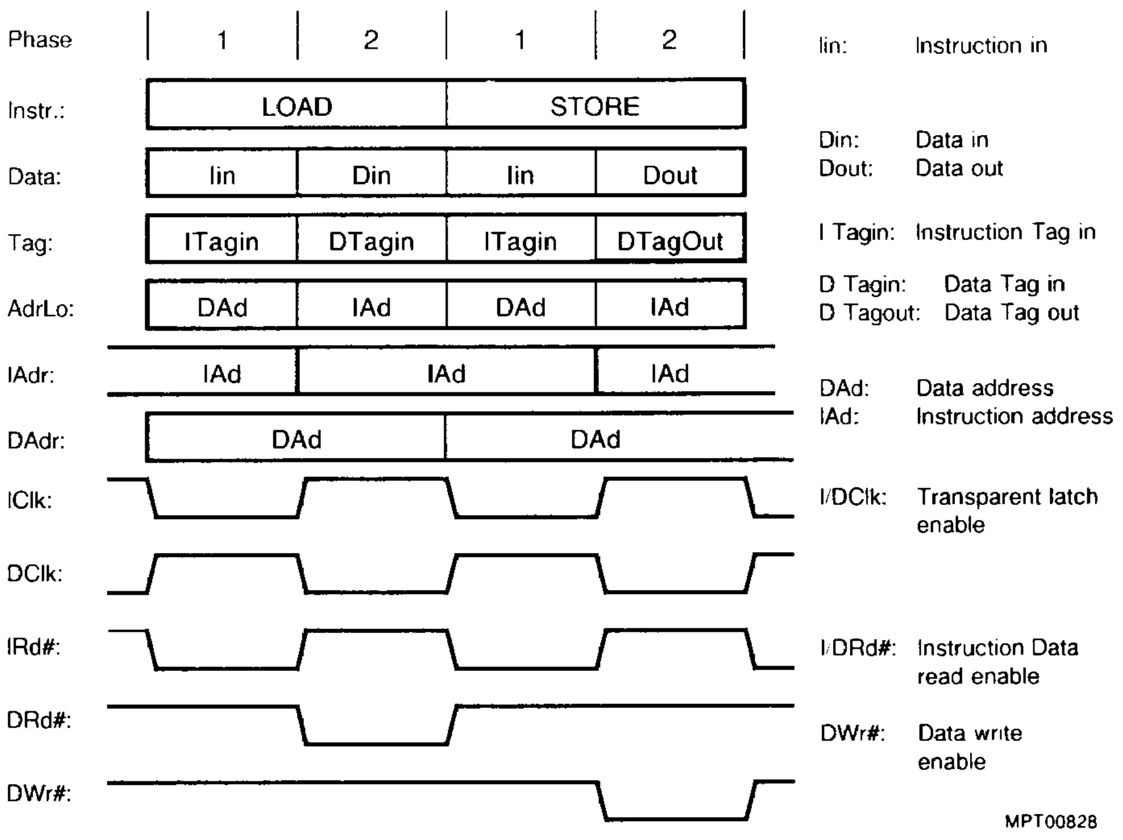


External Cache Interface

As described earlier the SAB-R2000A supports separate caches for instructions and data. As was seen in figure 3 the physical address coming out of the TLB is split across the external buses; AdrLo (16 bits) and TAG (20 bits plus valid and 3 parity bits). The caches are addressed by the 16 bit address bus AdrLo. Since AdrLo presents byte addresses and the caches are organised as words, its least significant 2 bits are not used by the caches. The most significant four bits of AdrLo bus are identical to the least significant four bits of the TAG bus but are output with AdrLo timing. This overlap allows cache size to vary with implementation (i.e. from 4 to 64 Kbytes).

The processor interleaves accesses to the two caches on the AdrLo, Tag and Data buses. Instruction fetch begins with AdrLo (IAd) clocked through a transparent latch by IClk# during phase 2 of a machine cycle, and continues until Data (Iin) and Tag (ITin) are latched on the chip at the end of the next phase 1. This is shown in figure 31. In the diagram the IAdr and DAdr buses are latched versions of AdrLo – refer also to figure30.

Figure 31
Cache Interface Operation



Similarly, data fetch begins with AdrLo (DAd) clocked by DClk# during phase 1, and completes with Data (Din) and Tag (DTin) latched on the chip at the end of phase 2. During data stores, all three buses are outputs from the chip to the cache and memory interface. The AdrLo (DAd) is transmitted during phase 1 and the Data (Dout) and Tag (DTagout) during phase 2. The memory interface combines AdrLo and Tag to generate the full 32-bit real address for main memory access. Refer to figure 31 for details.

The cache interface integrates all circuitry that would normally be required between a processor and raw cache RAMS, such as the control lines for cache write and tristate output enables which are all generated on chip.

Note: *Partial word stores such as Store byte and Store halfword are written directly to memory and the associated word in the cache is invalidated.*

Memory Interface

The memory interface which is shown on the bottom left of figure 30 contains several signals which synchronize memory access events. MemRd# and MemWr# are asserted on cache miss (i.e. main memory read access initiated) and store respectively. The access type, i.e. byte, half word, tribyte, and word transfers are determined by the AccTy2:0 bits.

The principal supporting mechanism for main memory operations is the processor stall cycle. Main memory stalls occur when loads miss in the cache or when stores are blocked by the write buffer. RdBusy and WrBusy# control the termination and initiation of the stalls when the cache misses or the write buffer is full. BusError# warns of memory access errors such as parity error or bus timeout. The memory interface can also support system configurations where one or both caches are missing.

External Coprocessor Interface

The external coprocessor interface is illustrated on the bottom right of figure 30. It is designed to support the SAB-R2010A floating point accelerator, in what is called a tightly coupled interface, and up to two additional coprocessors. External coprocessors are connected to the Data bus only. During each cycle in which a valid Instruction-Data pair is on the bus, the coprocessors accept an Instruction. The coprocessors decode the Instruction in parallel with the main processor and, if it is a coprocessor Instruction, one of the coprocessors will proceed to execute the Instruction. A coprocessor condition (CpCond) signal, one for each coprocessor type, allows the main processor to branch on a coprocessor condition set up by a previous operation. Any coprocessor can assert CpBusy to stall the main CPU when a coprocessor instruction is issued while the coprocessor in question still has the required functional unit busy with an earlier operation. The SAB-R2000A asserts Run# to advance operations in the coprocessors. When Run# is deasserted in the n th cycle, coprocessors disregard the Instruction-Data pair presented in the n-1th cycle. The assertion of Exc# (Exception) indicates that the SAB-R2000A is taking an exception. CpSync# is used for timing synchronization between the SAB-R2000A and the coprocessor.

System Configuration

Due to the flexible interfaces of the SAB-R2000A it can be used in a variety of system configurations, ranging from high-end Workstations and parallel processors to low-end embedded control applications.

A high performance system configuration, which is suitable for high-end computing applications, is shown in figure 32.

The main components of this system, along with the SAB-R2000A, are separate Instruction and Data caches (64Kbyte each) and the SAB-R2010A Floating Point Accelerator. Main memory consists of DRAM. When a 16 MHz SAB-R2000A and SAB-R2010A are used in conjunction with 30ns SRAMs this system can deliver 12 VAX mips. Such system configurations are employed in high-end UNIX Workstations.

The SAB-R2000A is not only suited to high-end computing applications, it is also well suited to provide cost effective solutions for embedded control applications. The SAB-R2000A may be used to design systems with different degrees of performance. This can be achieved by varying the cache size (4 to 64 Kbyte each), the number of primary caches (0, 1 or 2), the I/O system configuration and the frequency the system will run at. There is a 512 Mbyte unmapped uncached region in the Address Space organisation (see the Memory Management System section) which can be used for a slow main memory interface when a cache is not implemented. There are also two other possibilities to implement a system without a cache

(a) cause a cache miss all the time; this can be achieved by having an external register, which upon cache access requests by the processor returns a deasserted Valid bit (i.e. invalid cache entry).

(b) Have the Operating System mark every page as uncacheable in the page table/TLB by setting the "N" (Non-cacheable) bit in the TLB entry. This should be done for both Kernel and User.

An example of a cost effective system configuration for a deterministic real time embedded application is illustrated in figure 33.

Figure 32
Solution for High-End-Computing Application

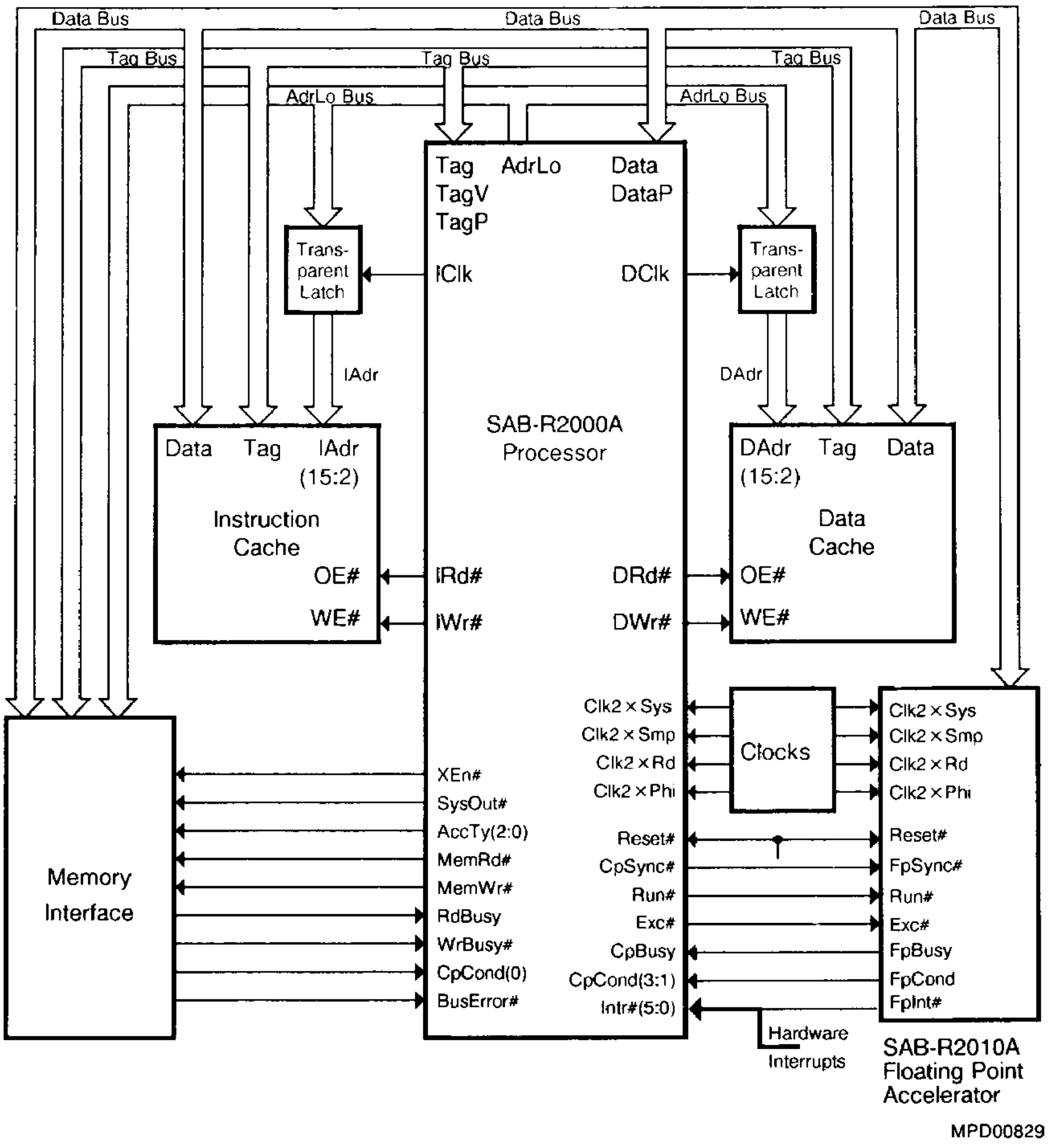
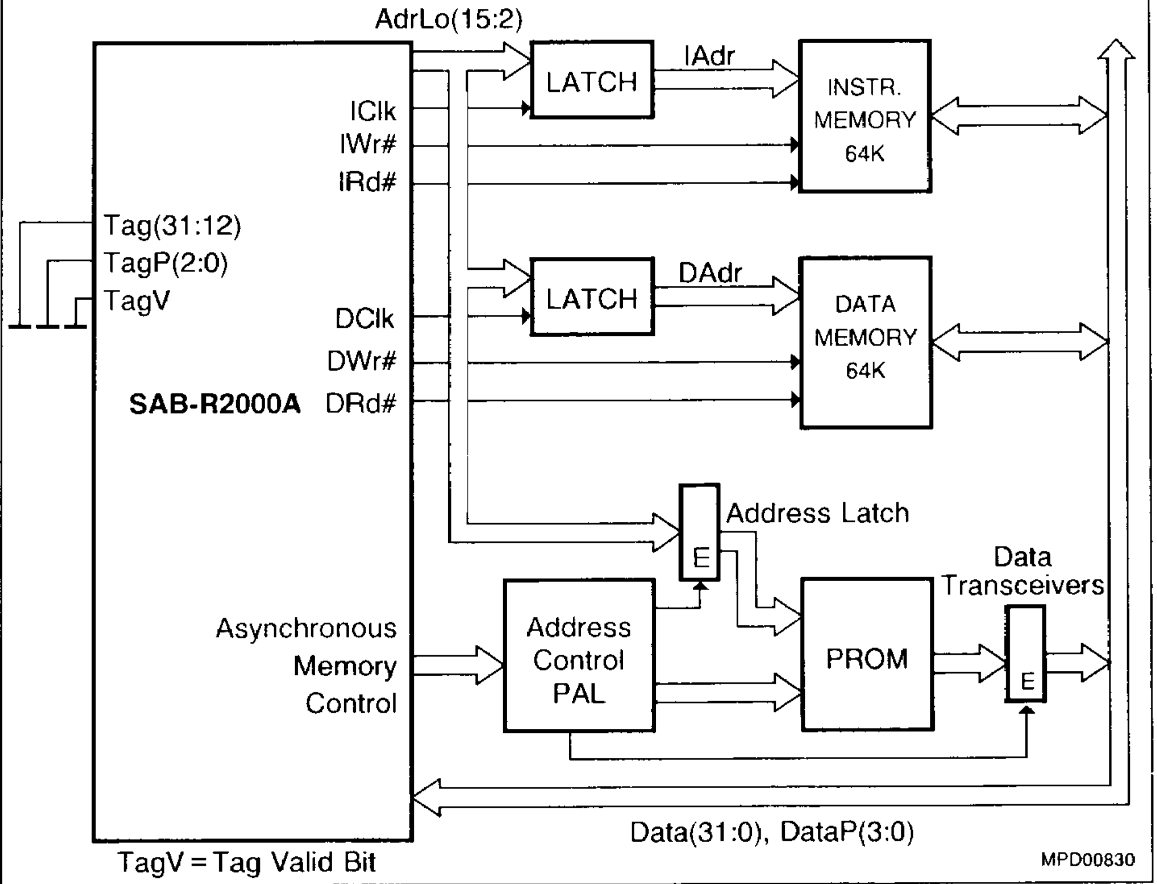


Figure 33
Solution for a Predictable Real Time Controller System



It is extremely difficult to apply the normal cache solution here due to the deterministic requirement. The SAB-R2000A can be configured so that a deterministic behaviour can be guaranteed. The technique employed is to use the synchronous bus (cache interface) to drive SRAMs where they preform the function of local main memory. The system configuration illustrated here is an example of a real time system with predictable responses of external and internal events, minimum context switch overhead and with a deterministic behaviour.

The system consists of a memory system with 64 Kbytes for Instructions and 64 Kbytes for Data (SRAMs). The asynchronous memory control (main memory interface) is used to address the PROM which contains the program to be loaded at initialization. With a 16 MHz SAB-R2000A and 20 ns SRAMs 10-12 VAX mips can be achieved. The worst case reaction time to an interrupt is 7 to 9 instructions, which consists of the system overhead until the user interrupt routine takes control. In this case, due to the fact that everything is in the SRAM's it takes only one cycle per instruction so the response time to an interrupt is 7 to 9 machine cycles (420-540 ns).

Instruction Set Summary

The following section is a table of the instructions available in the SAB-R2000A. The instructions are listed in alphabetical order. For a more detailed description of the operation of each instruction refer to the "SAB-R2000A Users Manual". A chart at the end of this section lists the bit encoding for the constant fields of each instruction.

Instruction Notation Convention

The table that follows is split up into three columns: Instruction, Format and Operation. The Instruction column contains the mnemonic name of the instruction and its meaning. The instruction format (refer to figure 10) and Assembly language notation, for each instruction, is listed in the Format column. The Operation column describes the operation performed by each instruction using a high level language notation. Special symbols used in the notation are described in table 4.

Table 4
SAB-R2000A Instruction Operation Notations

Symbol	Meaning
$\leftarrow$	Assignment
$\|\|$	Bit string concatenation
x^y	Replication of bit value x into a y-bit string. Note that x is always a single-bit value.
$x_{y..z}$	Selection of bits y through z of bit string x. Little-endian bit notation is always used. If y is less than z, this expression is an empty (zero length) bit string.
+	Two's complement addition
–	Two's complement subtraction
*	Two's complement multiplication
div	Two's complement integer division
mod	Two's complement modulo
<	Two's complement less than comparison
and	Bitwise logic AND
or	Bitwise logic OR
xor	Bitwise logic XOR
nor	Bitwise logic NOR
GPR[x]	General Register x. The content of GPR[0] is always zero. Attempts to alter the content of GPR[0] have no effect
CPR[z,x]	Coprocessor unit z, general register x
CCR[z,x]	Coprocessor unit z, control register x
T + i	Indicates the time steps (CPU cycles) between operations. Thus, operations identified as occurring at T + 1 are performed during the cycle following the one where the instruction was initiated. This type of operation occurs with loads, stores, jumps, branches and coprocessor instructions.
vAddress	Virtual address
pAddress	Physical address

In the Load/Store operation descriptions, the functions listed in table 5 are used to summarize the handling of virtual addresses and physical memory.

Table 5
Load/Store Common Functions

Function	Desription
Addr Translation	Uses the TLB to find the physical address given the virtual address. The function fails and an exception is taken if the entry for the page containing the virtual address is not present in the TLB (Translation Lookaside Buffer).
Load Memory	Uses the cache and main memory to find the contents of the word containing the specified physical address. The low-order two bits of the address and the access type field indicate which of each of the four bytes within the data word need to be returned. If the cache is enabled for this access. The entire word is returned and loaded into the cache.
Store Memory	Uses the cache, write buffer, and main memory to store the word or part of word specified as data into the word containing the specified physical address. The low-order two bits of the address and the access type field indicate which of the four bytes within the data word should be stored.

Instruction Set Summary

Instruction	Format	Operation
ADD: Add	R-type; ADD rd, rs, rt	T: $GPR[rd] \leftarrow GPR[rs] + GPR[rt]$
ADDI: Add Immediate	I-type; ADDI rt, rs, immediate	T: $GPR[rt] \leftarrow GPR[rs] + (immediate_{15})^{16} \Vert immediate_{15..0}$
ADDIU: Add Immediate Unsigned	I-type; ADDIU rt, rs, immediate	T: $GPR[rt] \leftarrow GPR[rs] + (immediate_{15})^{16} \Vert immediate_{15..0}$
ADDU: Add Unsigned	R-type; ADDU rd, rs, rt	T: $GPR[rd] \leftarrow GPR[rs] + GPR[rt]$
AND: And	R-type; AND rd,rs,rt	T: $GPR[rd] \leftarrow GPR[rs]$ and $GPR[rt]$
ANDI: And Immediate	I-type; ANDI rt, rs, immediate	T: $GPR[rt] \leftarrow 0^{16} \Vert$(immediate and $GPR[rs]_{15..0}$)
BCzF: Branch On Coprocessor z False	I-type; BCzF offset	T: target $\leftarrow (offset_{15})^{14} \Vert offset \Vert 0^2$ condition $\leftarrow$ not CpCond[z] T + 1: if condition then PC $\leftarrow$ PC + target endif
BCzT: Branch On Coprocessor z True	I-type; BCzT offset	T: target $\leftarrow (offset_{15})^{14} \Vert offset \Vert 0^2$ condition $\leftarrow$ CpCond[z] T + 1: if condition then PC $\leftarrow$ PC + target endif
BEQ: Branch on Equal	I-type; BEQ rs, rt, offset	T: target $\leftarrow (offset_{15})^{14} \Vert offset \Vert 0^2$ condition $\leftarrow$ (GPR[rs] = GPR[rt]) T + 1: if condition then PC $\leftarrow$ PC + target endif
BGEZ: Branch on Greater than or Equal to Zero	I-type; BGEZ rs, offset	T: target $\leftarrow (offset_{15})^{14} \Vert offset \Vert 0^2$ condition $\leftarrow$ ($GPR[rs]_{31} = 0$) T + 1: if condition then PC $\leftarrow$ PC + target endif

Instruction	Format	Operation

BGEZAL:
Branch on
Greater than or
Equal to Zero
And Link
	I-type;
	BGEZAL rs, offset
		T: target ← $(\text{offset}_{15})^{14}$ ∥ offset ∥ 0^2
		condition ← (GPR[rs] $_{31}$ = 0)
		GPR[31] ← PC + 8
		T + 1: if condition then
		PC ← PC + target
		endif

BGTZ:
Branch on
Greater Than
Zero
	I-type;
	BGTZ rs, offset
		T: target ← $(\text{offset}_{15})^{14}$ ∥ offset ∥ 0^2
		condition ← (GPR[rs] $_{31}$ = 0) and (GPR[rs] ≠ GPR[r0])
		T + 1: if condition then
		PC ← PC + target
		endif

BLEZ:
Branch on Less
than or Equal to
Zero
	I-type;
	BLEZ rs, offset
		T: target ← $(\text{offset}_{15})^{14}$ ∥ offset ∥ 0^2
		condition ← (GPR[rs] $_{31}$ = 1) or (GPR[rs] = GPR[r0])
		T + 1: if condition then
		PC ← PC + target
		endif

BLTZ:
Branch on Less
Than Zero
	I-type;
	BLTZ rs, offset
		T: target ← $(\text{offset}_{15})^{14}$ ∥ offset ∥ 0^2
		condition ← (GPR[rs] $_{31}$ = 1)
		T + 1: if condition then
		PC ← PC + target
		endif

BLTZAL:
Branch on Less
Than Zero And
Link
	I-type;
	BLTZAL rs, offset
		T: target ← $(\text{offset}_{15})^{14}$ ∥ offset ∥ 0^2
		condition ← (GPR[rs] $_{31}$ = 1)
		GPR[31] ← PC + 8
		T + 1: if condition then
		PC ← PC + target
		endif

BNE:
Branch on Not
Equal
	I-type;
	BNE rs, rt, offset
		T: target ← $(\text{offset}_{15})^{14}$ ∥ offset ∥ 0^2
		condition ← (GPR[rs] ≠ GPR[rt])
		T + 1: if condition then
		PC ← PC + target
		endif

BREAK: Break	R-type; BREAK	PC ← Exception Handler

Instruction	Format	Operation
CFCz: Move Control From Co-processor z	R-type; CFCz rt, rd	T: data ← CCR[z,rd] T + 1: GPR[rt] ← data
COPz: Coprocessor Operation z	Coprocessor type: COPz cofun	T: CoprocessorOperation (z. cofun)
CTCz: Move Control To Coprocessor z	R-type; CTCz	T: data ← GPR[rt] T + 1: CCR[z,rd] ← data

DIV:
Divide
	R-type;
	DIV rs, rt
		T-2: LO ← undefined
		HI ← undefined
		T–1: LO ← undefined
		HI ← undefined
		T: LO ← GPR[rs] *div* GPR[rt]
		HI ← GPR[rs] *mod* GPR[rt]

DIVU:
Divide Unsigned
	R-type;
	DIVU rs, rt
		T-2: LO ← undefined
		HI ← undefined
		T–1: LO ← undefined
		HI ← undefined
		T: LO ← (0‖GPR[rs]) *div* (0‖GPR[rt])
		HI ← (0‖GPR[rs]) *mod* (0‖GPR[rt])

J: Jump	J-type; J target	T: temp ← $PC_{31..28}$ ‖ target ‖ 0^2 T + 1: PC ← temp
JAL: Jump And Link	J-type; JAL target	T: temp ← $PC_{31..28}$ ‖ target ‖ 0^2 GPR[31] ← PC + 8 T + 1: PC ← temp
JALR: Jump And Link Register	R-type; JALR rs JALR rd, rs	T: temp ← GPR[rs] GPR[rd] ← PC + 8 T + 1: PC ← temp
JR: Jump Register	R-type; JR rs	T: temp ← GPR[rs] T + 1: PC ← temp

Instruction	Format	Operation
LB: Load Byte	I-type; LB rt, offset (base)	T: $vAddress \leftarrow (offset_{15})^{16} \parallel offset_{15..0} + GPR[base]$ $(pAddress, nonCacheable) \leftarrow AddrTranslation\ (vAddress)$ $mem \leftarrow LoadMemory\ (nonCacheable, BYTE, pAddress)$ $byte \leftarrow vAddress_{1..0}$ T + 1: if BigEndian then $GPR[rt] \leftarrow (mem_{31-8*byte})^{24} \parallel mem_{31-8*byte..24-8*byte}$ else $GPR[rt] \leftarrow (mem_{7+8*byte})^{24} \parallel mem_{7+8*byte..8*byte}$ endif
LBU: Load Byte Unsigned	I-type; LBU rt, offset (base)	T: $vAddress \leftarrow (offset_{15})^{16} \parallel offset_{15..0} + GPR[base]$ $(pAddress, nonCacheable) \leftarrow AddrTranslation\ (vAddress)$ $mem \leftarrow LoadMemory\ (nonCacheable, BYTE, pAddress)$ $byte \leftarrow vAddress_{1..0}$ T + 1: if BigEndian then $GPR[rt] \leftarrow 0^{24} \parallel mem_{31-8*byte..24-8*byte}$ else $GPR[rt] \leftarrow 0^{24} \parallel mem_{7+8*byte..8*byte}$ endif
LH: Load Halfword	I-type; LH rt, offset (base)	T: $vAddress \leftarrow (offset_{15})^{16} \parallel offset_{15..0} + GPR[base]$ $(pAddress, nonCacheable) \leftarrow AddrTranslation\ (vAddress)$ $mem \leftarrow LoadMemory(nonCacheable, HALFWORD, pAddress)$ $byte \leftarrow vAddress_{1..0}$ T + 1: if BigEndian then $GPR[rt] \leftarrow (mem_{31-8*byte})^{16} \parallel mem_{31-8*byte..16-8*byte}$ else $GPR[rt] \leftarrow (mem_{15+8*byte})^{16} \parallel mem_{15+8*byte..8*byte}$ endif

Instruction	Format	Operation
LHU: Load Halfword Unsigned	I-type; LHU rt, offset (base)	T: $vAddress \leftarrow (offset_{15})^{16} \| offset_{15..0} + GPR[base]$ $(pAddress, nonCacheable) \leftarrow AddrTranslation (vAddress)$ $mem \leftarrow LoadMemory(nonCacheable, HALFWORD, pAddress)$ $byte \leftarrow vAddress_{1..0}$ T + 1: if BigEndian then $GPR[rt] \leftarrow 0^{16} \| mem_{31-8*byte..16-8*byte}$ else $GPR[rt] \leftarrow 0^{16} \| mem_{15+8*byte..8*byte}$ endif
LUI: Load Upper Immediate	I-type; LUI rt, immediate	T: $GPR[rt] \leftarrow immediate \| 0^{16}$
LW: Load Word	I-type; LW rt, offset (base)	T: $vAddress \leftarrow (offset_{15})^{16} \| offset_{15..0} + GPR[base]$ $(pAddress, nonCacheable) \leftarrow AddrTranslation (vAddress)$ $mem \leftarrow LoadMemory (nonCacheable, WORD, pAddress)$ $byte \leftarrow vAddress_{1..0}$ T + 1: $GPR[rt] \leftarrow mem;$
LWCz: Load Word to Coprocessor z	I-type; LWCz rt, offset (base)	T: $vAddress \leftarrow (offset_{15})^{16} \| offset_{15..0} + GPR[base]$ $(pAddress, nonCacheable) \leftarrow AddrTranslation (vAddress)$ $mem \leftarrow LoadMemory(nonCacheable, WORD, pAddress_{31..2} \| 0^2)$ $byte \leftarrow vAddress_{1..0}$ T + 1: $CPR[z,rt] \leftarrow mem;$

Instruction	Format	Operation

LWL:
Load Word Left
	I-type;
	LWL rt, offset (base)
		T: vAddress $\leftarrow$ (offset$_{15}$)^{16} $\|$ offset$_{15..0}$ + GPR[base]
		(pAddress, nonCacheable) $\leftarrow$ AddrTranslation (vAddress)
		byte $\leftarrow$ vAddress$_{1..0}$
		if BigEndian then
		  mem $\leftarrow$ LoadMemory(nonCacheable,WORD-byte,pAddress)
		else
		  mem $\leftarrow$ LoadMemory(nonCacheable,byte,pAddress$_{31..2}$ $\|$ 0^2)
		endif
		T + 1: if BigEndian then
		  GPR[rt] $\leftarrow$ mem$_{31-8*byte..0}$ $\|$ GPR[rt]$_{8*byte-1..0}$
		else
		  GPR[rt] $\leftarrow$ mem$_{7+8*byte..0}$ $\|$ GPR[rt]$_{23-8*byte..0}$
		endif

LWR:
Load Word Right
	I-type;
	LWR rt, offset (base)
		T: vAddress $\leftarrow$ (offset$_{15}$)^{16} $\|$ offset$_{15..0}$ + GPR[base]
		(pAddress, nonCacheable) $\leftarrow$ AddrTranslation (vAddress)
		byte $\leftarrow$ vAddress$_{1..0}$
		if BigEndian then
		  mem $\leftarrow$ LoadMemory(nonCacheable,byte,pAddress$_{31..2}$ $\|$ 0^2)
		else
		  mem $\leftarrow$ LoadMemory(nonCacheable,byte,WORD-byte,pAddress)
		endif
		T + 1: if BigEndian then
		  GPR[rt] $\leftarrow$ GPR[rt]$_{31..8+8*byte}$ $\|$ mem$_{31..24-8*byte}$
		else
		  GPR[rt] $\leftarrow$ GPR[rt]$_{31..24-8*byte}$ $\|$ mem$_{31..8+8*byte}$
		endif

MFC0: Move From System Control Coprocessor z	R-type; MFC0 rt, rd	T: data CPR[0,rd] T + 1: GPR[rt] $\leftarrow$ data
MFCz: Move From Coprocessor z	R-type; MFCz rt, rd	T: data $\leftarrow$ CPR[z,rd] T + 1: GPR[rt] $\leftarrow$ data

Instruction	Format	Operation
MFHI: Move From HI	R-type; MFHI rd	T: GPR[rd] ← HI
MFLO: Move From LO	R-type; MFLO, rd	T: GPR[rd] ← LO
MTC0: Move To System Control Coprocessor	R-type; MTC0 rt, rd	T: data ← GPR[rt] T + 1: CPR[0,rd] ← data
MTCz: Move To Coprocessor z	R-type; MTCz rt, rd	T: data ← GPR[rt] T + 1: CPR[z,rd] ← data
MTHI: Move To HI	R-type; MTHI rs	T–2: HI ← undefined T–1: HI ← undefined T: HI ← GPR[rs]
MTLO: Move To LO	R-type; MTLO rs	T–2: LO ← undefined T–1: LO ← undefined T: LO ← GPR[rs]

MULT:
Multiply
	R-type;
	MULT rs,rt
		T–2: LO ← undefined
		HI ← undefined
		T–1: LO ← undefined
		HI ← undefined
		T: t ← GPR[rs]*GPR[rt]
		LO ← $t_{31..0}$
		HI ← $t_{63..32}$

MULTU:
Multiply Unsigned
	R-type;
	MULTU rs,rt
		T–2: LO ← undefined
		HI ← undefined
		T–1: LO ← undefined
		HI ← undefined
		T: t ← (0 ‖ GPR[rs])*(0 ‖ GPR[rt])
		LO ← $t_{31..0}$
		HI ← $t_{63..32}$

NOR: Nor	R-type; NOR rd, rs, rt	T: GPR[rd] ← GPR[rs] *nor* GPR[rt]

Instruction	Format	Operation
OR: Or	R-type; OR rd, rs, rt	T: $GPR[rd] \leftarrow GPR[rs]$ *or* $GPR[rt]$
ORI: Or Immediate	I-type; ORI rt, rs, immediate	T: $GPR[rt] \leftarrow GPR[rs]_{31..16} \Vert (immediate$ *or* $GPR[rs]_{15..0})$
RFE: Restore From Exception	R-type; RFE	T: $SR \leftarrow SR_{31..4} \Vert SR_{5...2}$
SB: Store Byte	I-type; SB rt, offset (base)	T: $vAddress \leftarrow (offset_{15})^{16} \Vert offset_{15..0} + GPR[base]$ $(pAddress, nonCacheable) \leftarrow AddrTranslation\ (vAddress)$ $byte \leftarrow vAddress_{1..0}$ if BigEndian then $data \leftarrow GPR[rt]_{7+8*byte..0} \Vert 0^{24-8*byte}$ else $data \leftarrow GPR[rt]_{31-8*byte..0} \Vert 0^{8*byte}$ endif T + 1: StoreMemory (nonCacheable, BYTE, data, pAddress)
SH: Store Halfword	I-type; SH rt, offset (base)	T: $vAddress \leftarrow (offset_{15})^{16} \Vert offset_{15..0} + GPR[base]$ $(pAddress, nonCacheable) \leftarrow AddrTranslation\ (vAddress)$ $byte \leftarrow vAddress_{1..0}$ IF BigEndian then $data \leftarrow GPR[rt]_{15+8*byte..0} \Vert 0^{15-8*byte}$ else $data \leftarrow GPR[rt]_{31-8*byte..0} \Vert 0^{8*byte}$ endif T + 1: StoreMemory (nonCacheable, HALFWORD, data, pAddress)
SLL: Shift Left Logical	I-type; SLL rd, rt shamt	T: $GPR[rd] \leftarrow GPR[rt]_{31-shamt..0} \Vert 0^{shamt}$
SLLV: Shift Left Logical Variable	R-type; SLLV rd, rt, rs	T: $GPR[rd] \leftarrow GPR[rt]_{(31-GPR[rs]_{4..0})..0} \Vert 0^{GPR[rs]_{4..0}}$

Instruction	Format	Operation

SLT:
Set on Less
Than
	R-type;
	SLT rd, rs, rt
		T: if GPR[rs] $<$ GPR[rt] then
		  GPR[rd] $\leftarrow 0^{31} \parallel 1$
		else
		  GPR[rd] $\leftarrow 0^{32}$
		endif

SLTI:
Set on Less
Than Immediate
	I-type;
	SLTI rt, rs, immediate
		T: if GPR[rs] $<$ $((\text{immediate}_{15})^{16} \parallel \text{immediate}_{15..0})$ then
		  GPR[rt] $\leftarrow 0^{31} \parallel 1$
		else
		  GPR[rt] $\leftarrow 0^{32}$
		endif

SLTIU:
Set on Less
Than Immediate
Unsigned
	I-type;
	SLTIU rt, rs, immediate
		T: if $(0 \parallel \text{GPR[rs]}) < (0 \parallel \text{immediate}_{15})^{16} \parallel \text{immediate}_{15..0})$ then
		  GPR[rt] $\leftarrow 0^{31} \parallel 1$
		else
		  GPR[rt] $\leftarrow 0^{32}$
		endif

SLTU:
Set on Less
Than Unsigned
	R-type;
	SLTU rd, rs, rt
		T: if $(0 \parallel \text{GPR[rs]}) < (0 \parallel \text{GPR[rt]})$ then
		  GPR[rd] $\leftarrow 0^{31} \parallel 1$
		else
		  GPR[rd] $\leftarrow 0^{32}$
		endif

SRA: Shift Right Arithmetic	R-type; SRA rd, rt, shamt	T: GPR[rd] $\leftarrow (\text{GPR[rt]}_{31})^{\text{shamt}} \parallel \text{GPR[rt]}_{31..\text{shamt}}$
SRAV: Shift Right Arithmetic Variable	R-type; SRAV rd, rt, rs	T: GPR[rd] $\leftarrow (\text{GPR[rt]}_{31})^{\text{GPR[rs]}4..0} \parallel \text{GPR[rt]}_{31..(\text{GPR[rs]}_{4..0})}$
SRL: Shift Right Logical	R-type; SRL rd, rt, shamt	T: GPR[rd] $\leftarrow 0^{\text{shamt}} \parallel \text{GPR[rt]}_{31..\text{shamt}}$
SRLV: Shift Right Logical Variable	R-type; SRLV rd, rt, rs	T: GPR[rd] $\leftarrow 0^{\text{GPR[rs]}4..0} \parallel \text{GPR[rt]}_{31..(\text{GPR[rs]}_{4..0})}$

Instruction	Format	Operation
SUBU: Subtract Unsigned	R-type; SUBU rd, rs, rt	T: GPR[rd] ← GPR[rs] –GPR[rt]
SUB: Subtract	R-type; SUB rd, rs, rt	T: GPR[rd] ← GPR[rs] –GPR[rt]
SW: Store Word	I-type; SW rt, offset (base)	T: vAddress←$(\text{offset}_{15})^{16} \parallel \text{offset}_{15..0}$ + GPR[base] (pAddress, nonCacheable)←AddrTranslation (vAddress) data←GPR[rt] T + 1: StoreMemory (nonCacheable, WORD, data, pAddress)
SWCz: Store Word from Coprocessor z	I-type; SWCz rt, offset (base)	T: vAddress←$(\text{offset}_{15})^{16} \parallel \text{offset}_{15..0}$ + GPR[base] (pAddress, nonCacheable)←AddrTranslation (vAddress) data←CPR[z,t] T + 1: StoreMemory (nonCacheable, 15, data, $\text{pAddress}_{31..2} \parallel 0^2$)

SWL:
Store Word Left
	I-type;
	SWL rt, offset (base)
		T: vAddress←$(\text{offset}_{15})^{16} \parallel \text{offset}_{15..0}$ + GPR[base]
		(pAddress,nonCacheable)←AddrTranslation (vAddress)
		byte←$\text{vAddress}_{1..0}$
		if BigEndian then
		    data←$0^{8 \cdot \text{byte}} \parallel \text{GPR[rt]}_{31..8 \cdot \text{byte}}$
		else
		    data← $0^{24-8 \cdot \text{byte}} \parallel \text{GPR[rt]}_{31..24-8 \cdot \text{byte}}$
		endif
		T + 1: if BigEndian then
		    StoreMemory (nonCacheable,WORD-byte,data,pAddress)
		else
		    StoreMemory (nonCacheable,byte,data,$\text{pAddress}_{31..2} \parallel 0^2$)
		endif

Instruction	Format	Operation

SWR:
Store Word
Right
	I-type;
	SWR rt, offset (base)
		T: vAddress ← $(\text{offset}_{15})^{16}$ ‖ $\text{offset}_{15..0}$ + GPR[base]
		(pAddress, nonCacheable) ← AddrTranslation (vAddress)
		byte ← $\text{vAddress}_{1..0}$
		if BigEndian then
		    data ← $\text{GPR[rt]}_{7+8\cdot\text{byte}..0}$ ‖ $0^{24-8\cdot\text{byte}}$
		else
		    data ← $\text{GPR[rt]}_{31-8\cdot\text{byte}..0}$ ‖ $0^{8\cdot\text{byte}}$
		endif
		T + 1: if BigEndian then
		    StoreMemory (nonCacheable,byte,data, $\text{pAddress}_{31..2}$ ‖ 0^2)
		else
		    StoreMemory (nonCacheable,WORD-byte,data,pAddress)
		endif

SYSCALL: System Call	R-type; SYSCALL	PC ← ExceptionHandler

TLBP:
Probe TLB for
matching entry
	R-type;
	TLBP
		T: Index ← 1 ‖ 0^{31}
		for i in 0..TLBEntries-1
		    if (($\text{TLB}_{63..44}$[i] = $\text{EntryHi}_{31..12}$) and
		        (TLB_8 or ($\text{TLB}_{43..38}$ = $\text{EntryHi}_{11..6}$))) then
		        Index ← 0^{18} ‖ $\text{i}_{5..0}$ ‖ 0^8
		    endif
		endfor

TLBR: Read indexed TLB entry	R-type; TLBR	T: EntryHi ← TLB $[\text{Index}_{13..8}]_{63..32}$ EntryLo ← TLB $[\text{Index}_{13..8}]_{31..0}$
TLBWI: Write Indexed TLB entry	R-type; TLBWI	T: TLB $[\text{Index}_{13..8}]_{63..32}$ ← EntryHi TLB $[\text{Index}_{13..8}]_{31..0}$ ← EntryLo
TLBWR: Write Random TLB entry	R-type; TLBWR	T: $\text{TLB}[\text{Random}_{13..8}]_{63..32}$ ← EntryHi $\text{TLB}[\text{Random}_{13..8}]_{31..0}$ ← EntryLo
XOR: Exclusive Or	R-type; XOR rd, rs, rt	T: GPR[rd] ← GPR[rs] *xor* GPR[rt]
XORI: Exclusive Or Immediate	I-type; XORI rt, rs, immediate	T: GPR[rt] ← $\text{GPR[rs]}_{31..16}$ ‖ (immediate *xor* $\text{GPR[rs]}_{15..0}$)

Instruction Encoding

op

31..29 \ 28..26	0	1	2	3	4	5	6	7
0	SPECIAL	REGIMM	J	JAL	BEQ	BNE	BLEZ	BGTZ
1	ADDI	ADDIU	SLTI	SLTIU	ANDI	ORI	XORI	LUI
2	COP0	COP1	COP2	COP3	⊗	⊗	⊗	⊗
3	⊗	⊗	⊗	⊗	⊗	⊗	⊗	⊗
4	LB	LH	LWL	LW	LBU	LHU	LWR	⊗
5	SB	SH	SWL	SW	⊗	⊗	SWR	⊗
6	⊗	LWC1	LWC2	LWC3	⊗	⊗	⊗	⊗
7	⊗	SWC1	SWC2	SWC3	⊗	⊗	⊗	⊗

SPECIAL function

5..3 \ 2..0	0	1	2	3	4	5	6	7
0	SLL	⊗	SRL	SRA	SLLV	⊗	SRLV	SRAV
1	JR	JARL	⊗	⊗	SYSCALL	BREAK	⊗	⊗
2	MFHI	MTHI	MFLO	MTLO	⊗	⊗	⊗	⊗
3	MULT	MULTU	DIV	DIVU	⊗	⊗	⊗	⊗
4	ADD	ADDU	SUB	SUBU	AND	OR	XOR	NOR
5	⊗	⊗	SLT	SLTU	⊗	⊗	⊗	⊗
6	⊗	⊗	⊗	⊗	⊗	⊗	⊗	⊗
7	⊗	⊗	⊗	⊗	⊗	⊗	⊗	⊗

REGIMM rt

20..19 \ 18..16	0	1	2	3	4	5	6	7
0	BLTZ	BGEZ	~	~	~	~	~	~
1	~	~	~	~	~	~	~	~
2	BLTZAL	BGEZAL	~	~	~	~	~	~
3	~	~	~	~	~	~	~	~

⊗ Codes marked with a '⊗' cause unimplemented instruction exceptions and are reserved for future versions of the architecture.

~ Codes marked with a '~' are not valid and are reserved for future versions of the architecture. The results of such an encoding are undefined.

COPz rs

	23..21							
25..24	0	1	2	3	4	5	6	7
0	MF	~	CF	~	MT	~	CT	~
1	~	⊗	⊗	⊗	⊗	⊗	⊗	⊗
2	CO							
3								

COPz rt

	18..16							
20..19	0	1	2	3	4	5	6	7
0	BCF	BCT	~	~	~	~	~	~
1	~	~	~	~	~	~	~	~
2	~	~	~	~	~	~	~	~
3	~	~	~	~	~	~	~	~

COP0 function

	2..0							
4..3	0	1	2	3	4	5	6	7
0	~	TLBR	TLBWI	~	~	~	TLBWR	~
1	TLBP	~	~	~	~	~	~	~
2	RFE	~	~	~	~	~	~	~
3	~	~	~	~	~	~	~	~
4	~	~	~	~	~	~	~	~
5	~	~	~	~	~	~	~	~
6	~	~	~	~	~	~	~	~
7	~	~	~	~	~	~	~	~

⊗ Codes marked with a '⊗' cause unimplemented instruction exceptions and are reserved for future versions of the architecture.

~ Codes marked with a '~' are not valid and are reserved for future versions of the architecture. The results of such an encoding are undefined.

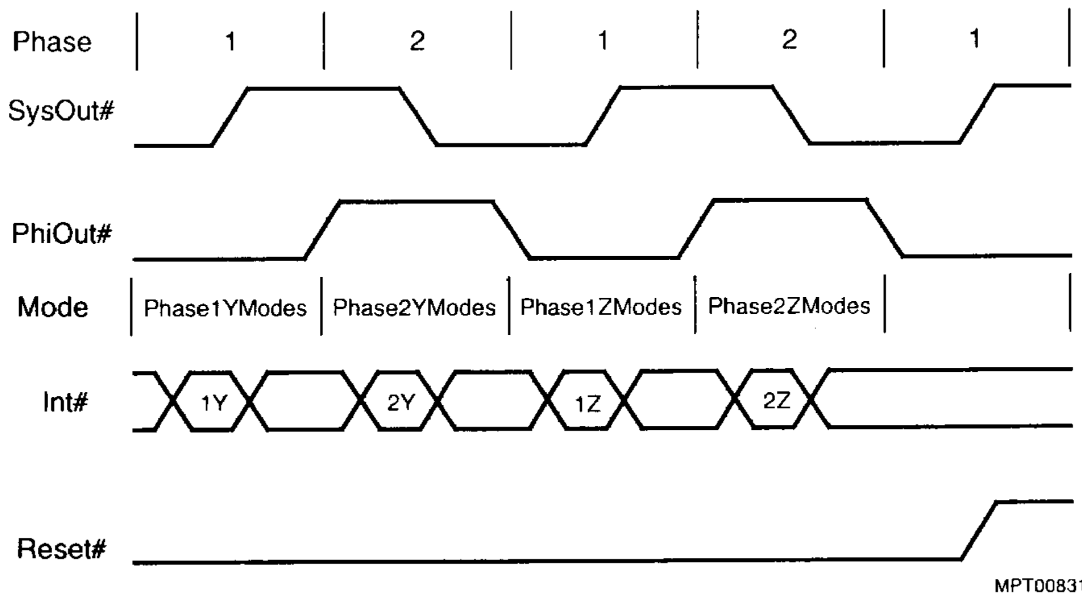
Resetting the SAB-2000A

The Reset# input signal is used to force processor execution to start at the reset vector (reset exception servicing routine) and to initialize the processor state. The Reset# signal must be asserted for a minimum of 6 cycles to guarantee processor initialization. After a reset has occured (i.e. before the exception handling routine has been executed) the following processor state is guaranteed:

- KUc, the current Kernel/User bit, is zero corresponding to Kernel mode.
- IEc, the current interrupt enable bit, is zero corresponding to interrupts disabled.
- TS, the TLB shutdown bit, is zero corresponding to TLB enabled.
- SwC, the Swap Cache bit, is zero corresponding to caches not swapped.
- BEV, the Boot Exception Vector bit, is one corresponding to selection of the bootstrap exception vector.
- The Random register is set to 63.

When the Reset# signal is deasserted in the n th cycle, the logic levels on the 6 interrupt pins during the n-2 and n-1 cycles are sampled by the processor to determine various processor operating modes such as Endianness, Cachelessness, Test etc. The last two cycles before the cycle in which the Reset# signal is deasserted are called the Y and Z cycles, respectively. Figure 34 illustrates these two cycles. The Reset# timings are described in the Timing Parameters section.

Figure 34
Y and Z Cycles on Reset# Deassertion



The Mode Select Summary Table, given in the Pin Definitions and Functions section, summarizes the processors mode selectable features and is reproduced here.

Table 6
Summary of Mode Select

Interrupt#	**Y Cycle Modes**		**Z Cycle Modes**	
	Phase 1	**Phase 2**	**Phase 1**	**Phase 2**
Int#(0)	Reserved	Reserved	Reserved	Big Endian#
Int#(1)	Reserved	Reserved	Reserved	Tristate#
Int#(2)	Reserved	Reserved	Reserved	NoCache#
Int#(3)	Reserved	Reserved	Reserved	BusDriveOn
Int#(4)	PhaseDelayOn#	PhaseDelayOn#	Asserted#	PhaseDelayOn#
Int#(5)	Reserved	Res5rved	Asserted#	R2000Md

Note that all reserved modes in this table must be driven asserted to guarantee compatibility with future processor revisions.

Asserting **PhaseDelayOn#** causes the processor to insert additional phase delay into its input clock paths. This additional phase delay allows coprocessors to minimize their skew, i.e. phase lock to the SAB-R2000A.

Byte order or Endianness is determined by the value of **BigEndian#**. Assertion will result in Big Endian ordering, while deassertion will result in Little Endian ordering.

Assertion of **Tristate#** causes the processor to tristate all of its outputs. In this condition the processor outputs can be driven by an external medium.

When **NoCache#** is asserted all memory references are forced to occur at the processor cycle rate, i.e. no cache miss stalls occur.

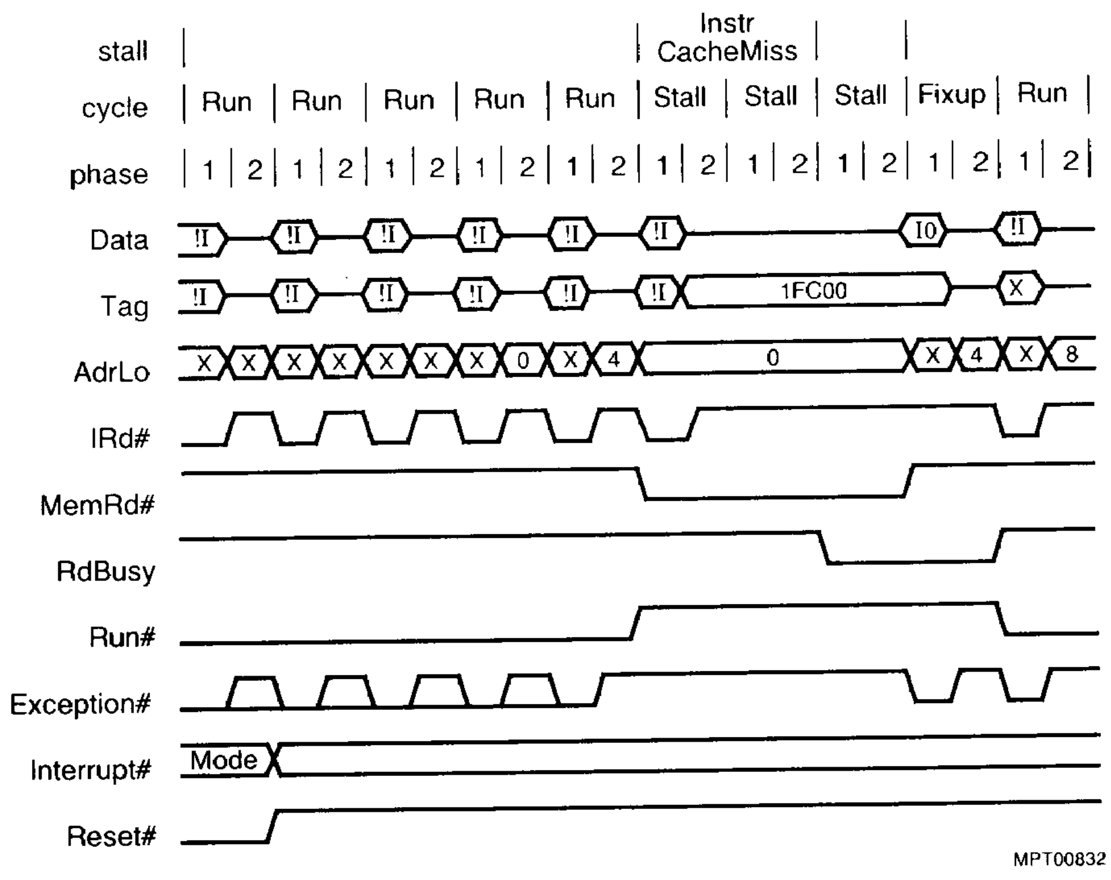
Data and Tag buses are driven during phase 2 of write busy and coprocessor busy stalls when **BusDriveOn** is asserted. If the Data and Tag buses are not being driven externally during these stalls and fast TTL inputs are connected to the bus, the BusDriveOn should be asserted to prevent bus oscillation due to the bus being pulled to the trip point of Fast TTL.

In the Mode Select Summary Table "#" means active low. This means, for example, in phase 2 of the Z cycle, if Int#0 has logic value low, Big Endian mode will be selected. On the other hand, if Int#0 has logic value high, Little Endian mode will be selected (i.e. the opposite).

Note that the Reset# signal must be asserted asynchronously and deasserted synchronously with the output Clock SysOut#. The reason for asserting Reset# asynchronously is to avoid a possible dead-lock if the Sysout# is used to clock it. More detailed information can be obtained in the "MIPS R3000 Processor Interface" specification and in a MIPS application note entitled "Resetting the R3000 and R3010".

The Reset Exception itself occurs when the Reset# signal is asserted and then deasserted. The Reset exception vector is selected to appear within the uncached, unmapped memory space (kseg1) of the machine so that instructions can be fetched and executed while the cache and memory system are still in an undefined state. The Reset exceptions special interrupt vector is 0xbfc00000. This is a virtual address and resides in kseg1 as explained in the Memory Management section. Kseg1 is direct mapped into the first 512 Mbytes of physical memory and the physical address is defined by subtracting 0xa0000000 from the virtual address of the reset exceptions interrupt vector, which yields 0x1fc00000. This can be seen in figure 35, which illustrates the sequence of events when Reset# is deasserted, i.e. when the processor comes out of reset. As can be seen the address (physical) on the Tag and AdrLo Buses is 0x1fc00000, as expected. Refer to the Timing Specification section for the notation.

Figure 35
Reset Behavior



Timing Specifications

Absolute Maximum Ratings

Ambient temperature under bias	0 to +70 °C
Storage temperature	− 65 to +150 °C
Supply Voltage (V_{CC})	− 0.5 to +7.0 V
Input voltage (V_{IN})	− 0.5 to +7.0 V
Load Capacitance on any Pin (C_{Ld})	100 pF

Note: *Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*
Not more than one output should be shorted at a time. Duration of the short should not exceed 30 seconds.

DC Characteristics

T_A = 0 to +70 °C; V_{CC} = 5 V ±5%

Parameter	Symbol	Limit values						Unit	Test condition
		12.5 MHz		16.67 MHz		20 MHz			
		min.	max.	min.	max.	min.	max.		
Operating Parameters									
Output HIGH voltage	V_{OH}	3.5	–	3.5	–	3.5	–	V	V_{CC} = min. I_{OH} = −4mA
Output LOW voltage	V_{OL}	–	0.4	–	0.4	–	0.4	V	V_{CC} = min. I_{OL} = 4mA
Input HIGH voltage	V_{IH}	2	V_{CC} + 0.25	2	V_{CC} + 0.25	2.0	V_{CC} + 0.25	V	
Input LOW voltage	V_{IL}	− 0.5[1]	0.8	− 0.5[1]	0.8	–	0.8	V	
Input HIGH voltage	V_{IHS}[2]	2.5	V_{CC} + 0.25	3.0	V_{CC} + 0.25	3.0	V_{CC} + 0.25	V	
Input LOW voltage	V_{ILS}[2]	− 0.5[1]	0.4	− 0.5[1]	0.4	–	0.4	V	
Input capacitance	C_{In}	–	10	–	10	–	10	pF	
Output capacitance	C_{Out}	–	10	–	10	–	10	pF	
Operating current	I_{CC}	–	550	–	600	–	630	mA	V_{CC} = 5.25V

1) V_{IL} min. = − 3.0 V for pulse width less than 15 ns
2) V_{IHS} and V_{ILS} apply to Clk2 × Sys, Clk2 × Smp, Clk2 × Rd, Clk2 × Phi, CpBusy, and Reset#.

AC Characteristics

$T_A = 0$ to 70 °C; $V_{CC} = 5$ V ± 5%

Note: *All output timings are given assuming 25 pf of capacitive load. Output timings should be derated where appropriate as per the table below.*

All timings referenced to 1.5 V.

Parameter	Symbol	Limit values						Unit	Test condition
		12.5 MHz		16.67 MHz		20 MHz			
		min.	max.	min.	max.	min.	max.		
Clock Parameters [3]									
Input clock high	$t_{ClkHigh}$	16	–	8	–	10	–	ns	Transition ≤ 5ns
Input clock low	t_{ClkLow}	16	–	8	–	10	–	ns	Transition ≤ 5ns
Input clock period	t_{ClkP}	40	1000	30	1000	25	1000	ns	
Clk2 × Sys to Clk2 × Smp		0	$\frac{t_{Cyc}}{4}$	0	$\frac{t_{Cyc}}{4}$	0	$\frac{t_{Cyc}}{4}$	ns	
Clk2 × Smp to Clk2 × Rd		0	$\frac{t_{Cyc}}{4}$	0	$\frac{t_{Cyc}}{4}$	0	$\frac{t_{Cyc}}{4}$	ns	
Clk2 × Smp to Clk2 × Phi		11	$\frac{t_{Cyc}}{4}$	5	$\frac{t_{Cyc}}{4}$	7	$\frac{t_{Cyc}}{4}$	ns	
Run Operation Parameters									
Data enable	t_{DEn}	– 1	– 2.5	– 1	– 2	–	– 2	ns	–
Data disable	t_{DDis}	0	– 1	0	– 1	0	– 1	ns	–
Data valid	t_{DVal}	–	3.5	–	3	–	3	ns	25 pF Load
Write delay	t_{WrDly}	0	7.5	0	5	0	4	ns	25 pF Load
Data setup	t_{DS}	11.5	–	9	–	8	–	ns	–
Data hold	t_{DH}	– 2.5	–	– 2.5	–	– 2.5	–	ns	–
CpBusy setup	t_{CBS}	15	–	13	–	11	–	ns	–
CpBusy hold	t_{CBH}	– 2.5	–	– 2.5	–	– 2.5	–	ns	–
Access type(1-0)	t_{AcTy}	1	10	1	7	–	6	ns	25 pF Load
Access type(2)	t_{AT2}	1	20	1	17	–	14	ns	25 pF Load
Memory write	t_{MWr}	1	10	1	7	–	23	ns	25 pF Load
Exception	t_{Exc}	1	10	1	7	–	7	ns	25 pF Load

3) The clock parameters apply to all four 2xClocks: Clk2 × Sys, Clk2 × Smp, CLK2 × Rd, and Clk2 × Phi.

AC Characteristics (cont'd)

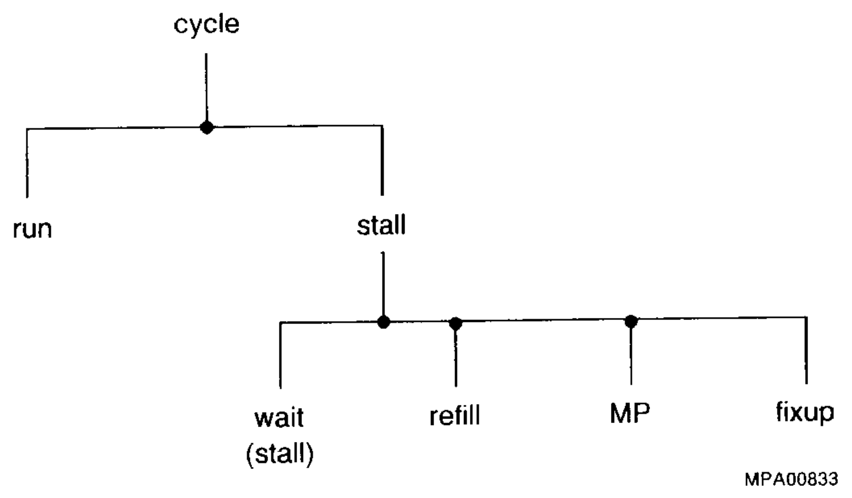
Parameter	Symbol	Limit values						Unit	Test condition
		12.5 MHz		16.67 MHz		20 MHz			
		min.	max.	min.	max.	min.	max.		
Stall Operation Parameters									
Address valid	t_{SAVal}	–	38	–	30	–	23	ns	25 pF Load
Access type	t_{SAcTy}	–	35	–	27	–	23	ns	25 pF Load
Memory read initiate	t_{MRdI}	1	35	1	27	–	23	ns	25 pF Load
Memory read terminate	t_{MRdT}	1	10	1	7	–	7	ns	25 pF Load
Run terminate	t_{Stl}	5	25	5	17	–	15	ns	25 pF Load
Run initiate	t_{Run}	5	10	5	7	–	6	ns	25 pF Load
Memory write	t_{SMWr}	5	35	5	27	–	23	ns	25 pF Load
Exception valid	t_{SExc}	5	28	5	20	–	18	ns	25 pF Load
Capacitive Load Deration									
Load derate	C_{LD}	0.5	2.5	0.5	2	0.5	1	ns/ 25pF	

As described earlier the SAB-2000A supports interfaces to external cache, main memory and coprocessors. This section describes the timing parameters and operation of the important cases for each of these interfaces along with Interrupt, Reset and Multi-processing examples.

Operation Fundamentals

A "cycle" is the basic instruction processing unit of the SAB-2000A processor. Cycles in which forward progress is made, i.e. an instruction is retired, are called "run" cycles. An instruction is retired either by its completion or in the presence of an exception its abortion. Cycles in which no forward progress is made are called "stall" cycles. Stall cycles are used for resolving urgent situations such as cache misses on loads, write system busy during stores, and coprocessor interlocks. All cycles can be classified as either run cycles or stall cycles. There are four types of stall cycles: "wait" stall cycles - simply known as stall cycles; "refill" stall cycles - which occur only during main memory reads; "multi-processor" (MP) stall cycles – allow the memory system to read or invalidate specific data cache entries; and "fixup" stall cycles – occur during the final cycle of the stall and are used in general to fix up the conditions which caused the stall. Processor transactions which occur during the first half of the cycle are called phase 1 transactions while those which occur during the second half of the cycle are called phase 2 transactions. Figure 36 summarizes the cycles in the SAB-2000A.

Figure 36
SAB-R2000A Cycles



As described earlier coprocessors maintain synchronization with the SAB-2000A by monitoring the signals Run# and Exception#. Run# is asserted by the SAB-2000A during run cycles and deasserted during stall cycles. When Run# is deasserted during the n th cycle, the coprocessor(s) disregard(s) the instruction-data pair presented during the n-1 th cycle. When Run# is reasserted during the m th cycle, the coprocessor(s) take(s), as a replacement for the instruction-data pair which was disregarded, the instruction-data pair presented during the m-1 th cycle – which was the final fixup cycle for whatever stall sequence was occurring.

Exception# is used by the coprocessor(s) to track exception related information during run cycles and stall related information during stall cycles.

- During phase 1 of run cycles Exception# indicates whether an exception has occurred for the instruction which is currently in its "writeback" pipestage. Unless the exception is occurring as a result of an interrupt request by the coprocessor, the assertion of Exception# prevents any state from being committed in the coprocessor.
- During phase 2 of run cycles Exception# indicates whether an interrupt request is being granted for the instruction which is currently in its "memory access" pipestage.
- During phase 1 of stall cycles Exception# indicates whether the current stall cycle is a fixup cycle. When a fixup cycle is occurring, it is guaranteed that the data present on the Data bus is valid. The coprocessor uses the fixup indication to qualify the use of data sampled from the bus during the stall.
- During phase 2 of stall cycles Exception# indicates whether the current stall is a Coprocessor Busy stall.

The use of the Exception# signal is summarized below.

Table 7

	phase 1	**phase 2**
Run	Exc1W#	IntGr2M#
Stall	Fixup1#	CpBusy2#

Processor Input Clocks

The SAB-R2000A has four separate double frequency (i.e. in a 16 MHz system these clocks are 32 MHz) input clocks. They can be adjusted to obtain optimum positioning of cache interface signals. The absolute timing of these clocks with respect to the processor outputs is undefined, only the differences are important. A short description of these four clocks follows.

- Clk2 × Sys:
  is the master clock and must lead all others. It determines the position of SysOut# (the processors output clock) with respect to Data, Tag and Address buses.
- Clk2xSmp:
  determines the sample point for data coming into the processor on all processor inputs except those coming directly from coprocessors.
- Clk2xRd:
  controls output enable time and provides sufficient address access to sample address hold from end of write, and data hold from end of write.
- Clk2xPhi:
  determines the position of the internal phases, phase 1 and phase 2.

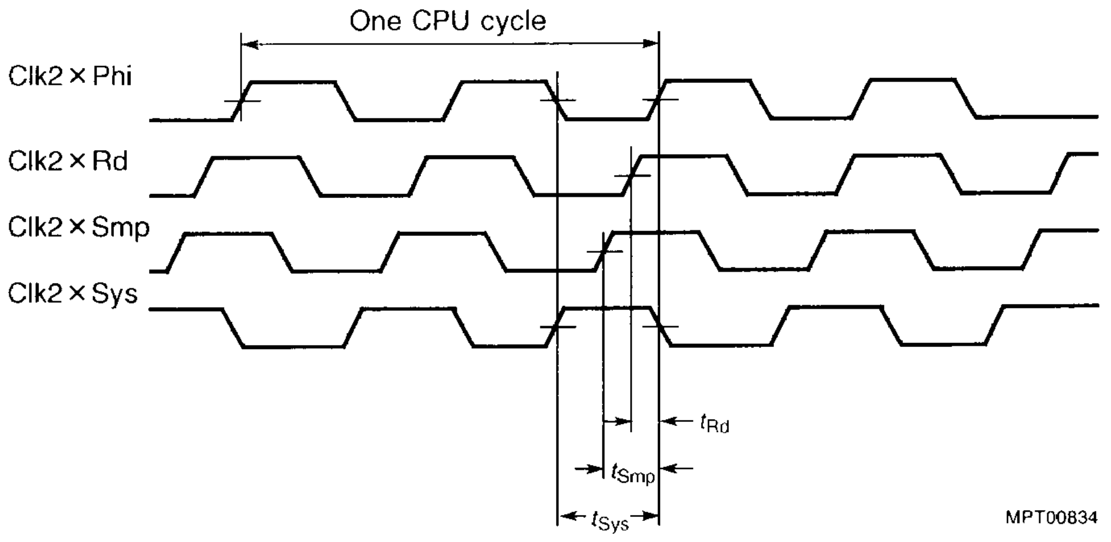
Table 8 illustrates the 2 × Clock dependency of the processors timing controlled outputs. Outputs are referenced only to "rising edges" of the 2 × Clocks. The assertion dependency is indicated by ↑ and deassertion by ↓.

Table 8

	Clk2 × Sys	Clk2 × Smp	Clk2 × Rd	Clk2 × Phi
IClk#, DClk#		↓	↑	
IRd, DRd	↓		↑	
IWr, DWr		↑ ↓		
SysOut#	↑ ↓			
Data, Tag			↓	↑
Address				↑ ↓
All Others				↑ ↓

Figure 37 shows the four 2x input clocks.

Figure 37
2x Input Clocks



In the timing diagrams which follow, timing specifications are given relative to a shifted version of the processor output clock SysOut#. The clock is called PhiOut# and is a virtual clock, i.e. the processor does not actually produce this output. It is shown in the timing diagrams for reasons of clarity, because its period is synchronous with a machine cycle. The shift amount is equal to the difference between Clk2 × Sys to Clk2 × Phi and, as is shown in figure 37, is t_{Sys}. Also, in the timing diagrams Clk2 × Sys and SysOut# are shown to clarify the relationship between these signals.

In reality SysOut# is produced rather than PhiOut# since this provides a signal with timing appropriate for synchronizing system transactions to the processor. Timings are given relative to PhiOut# since this makes determining the position of the input clocks the most straightforward. The timing of any output with respect to SysOut# can be determined from its timing with respect to PhiOut# by adding t_{Sys}.

Timing Diagram Notation

The following timing diagrams describe various transactions of the processor. Table 9 illustrates the notational conventions used in these diagrams.

Table 9

Character	Meaning
I	Instruction
D	Data
#	Active low
%	An incorrect datum
!	An unused datum
Z	The high impendance state
Ad	Address
in	into processor
out	out of processor
▨	not valid or Don't Care

Cache Timing

Cache operation was explained in the Interface section. Figure 38 illustrates cache operation and timing. During run cycles the Access Type bus, AccTy2:0, indicates whether or not a phase 2 transaction is scheduled for that cycle and the size of the datum being transferred. Table 10, below, summarizes AccTy encoding during run cycles.

Table 10

AccTy(2)	AccTy(1:0)	size
1	XX	no transaction
0	00	byte
0	01	half word
0	10	tribyte
0	11	word

Main Memory Reads

When a LOAD misses in the cache, a main memory read is initiated. Main memory reads are supported by read busy stalls and the MemRd#, RdBusy. Table 11 summarizes the meaning of the AccTy2:0 bus during main memory reads.

Table 11

AccTy(2)	AccTy(1:0)	size	type
0	00	byte	uncached/unkown
0	01	half word	uncached/unkown
0	10	tribyte	uncached/unkown
0	11	word	uncached/unkown
1	X0	word	cached/data
1	X1	word	cached/instruction

Figure 38 illustrates a single word transfer. Entry into the stall is indicated by the assertion of MemRd# which occurs in the cycle following the one in which the LOAD missed. During the stall the SAB-R2000A presents the read address on the AdrLo and Tag buses and tristates the Data bus. This state is maintained until RdBusy is deasserted. RdBusy is deasserted during phase 1 of the cycle in which the memory system will provide Data and Data parity on the Data bus. This is the termination of a read busy stall cycle.

The cycle following that in which RdBusy is deasserted is the fixup cycle. During this cycle the appropriate cache is written (in this case the Data cache) with the data returned by main memory. Simultaneously, the generated Data parity, Tag and Tag parity are also written to the cache.

Note: *The cache write does not occur if the stall was due to an uncached reference. The processor resumes run operation on the cycle following the fixup cycle if no other stall is pending.*

Figure 38
Cache Timing

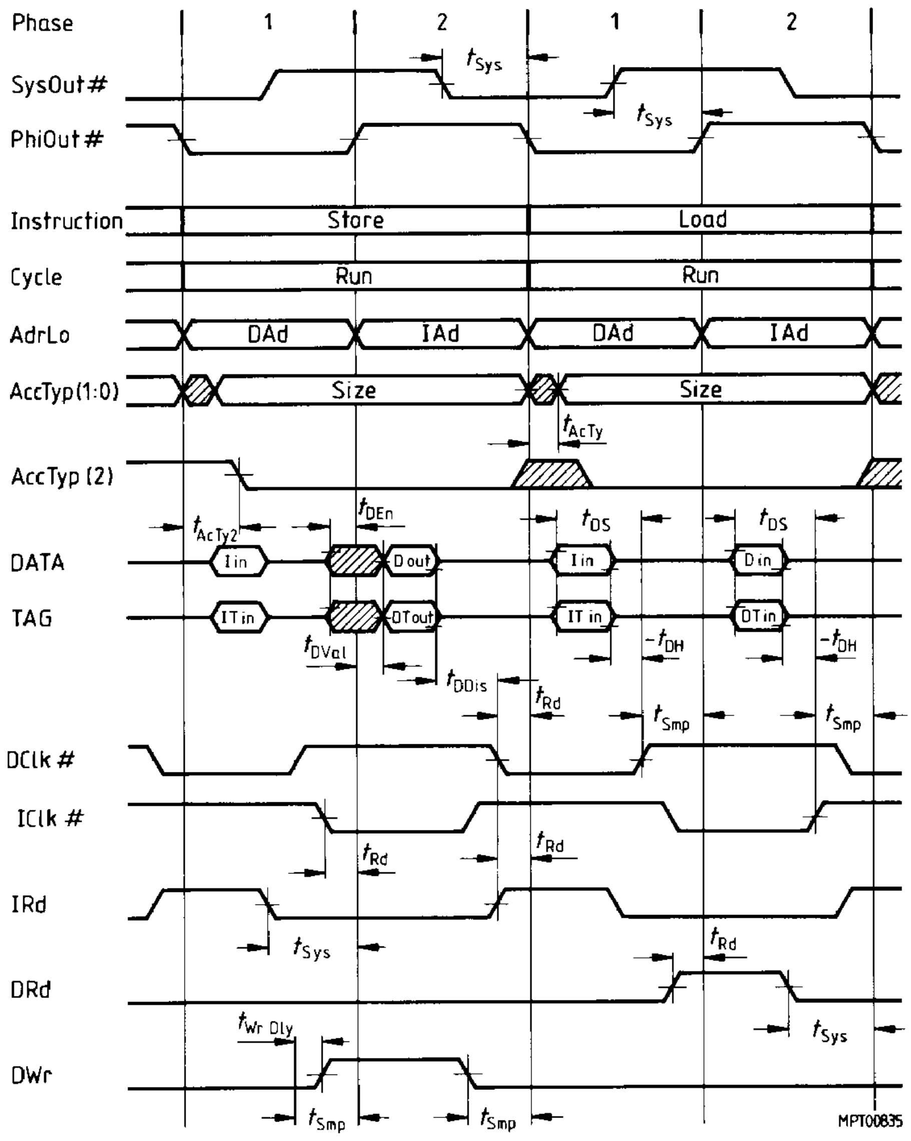
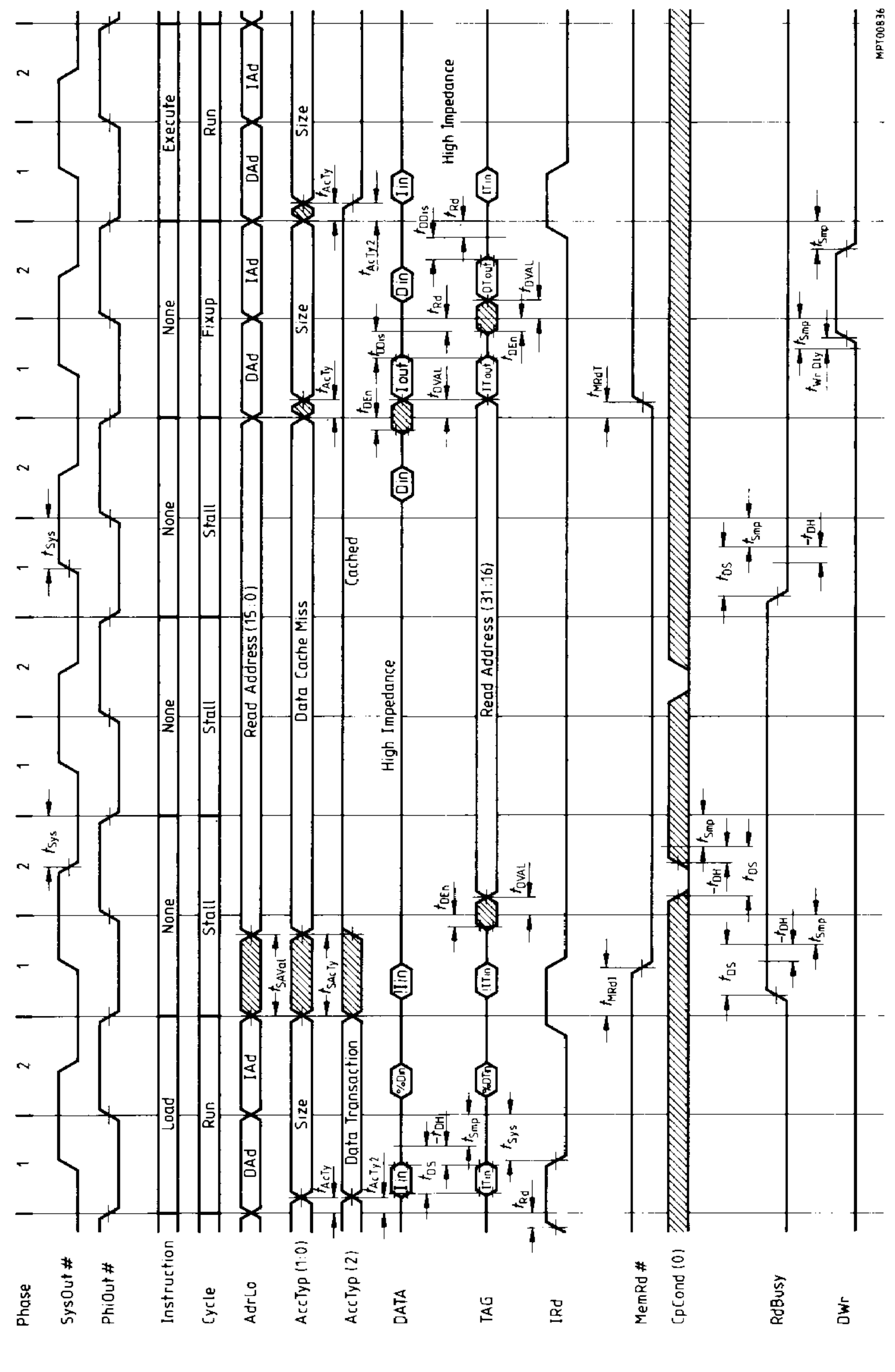


Figure 39
Single Word Main Memory Read (Data Cache Miss - Cached)



Main Memory Writes

Main memory writes are accomplished through a write buffer which accepts writes at cache speeds. A write is indicated to the Write Buffer by the assertion of the MemWr# signal. If the write buffer becomes full and a further write is attempted it causes a write busy stall, this is illustrated in figure 39. The first write (Store) shown fills the write buffer causing it to assert WrBusy#. The write which is attempted in the next cycle is not accepted by the write buffer and is redone by the processor during the fixup cycle. The write busy stall is terminated when the write buffer deasserts WrBusy# to indicate it can accept another write. The cycle following its deassertion will be the fixup cycle, where the write to the Write Buffer (Main Memory) is redone.

Interrupts

The SAB-R2000A has 6 general purpose interrupt inputs which are sampled during phase 2 of all run and fixup cycles. After causing an interrupt exception to occur, the interrupts continue to be sampled during each phase 2 to provide a level sensitive indication of the active interrupt(s). Figure 40 shows the Interrupt Timing.

Reset Timing

The Reset# input is used to force processor execution starting at the reset exception vector and to initialize processor state. Its operation is explained in the Resetting the SAB-R2000A section. Figure 41 illustrates its timing parameters.

Coprocessor Timing

During run cycles, the operation and timing of coprocessor LOADS and STORES is identical to that of the main processor. This can be seen in figure 42. To provide synchronization when required, the SAB-R2000A supports coprocessor busy stalls. The operation of such a stall is also illustrated in figure 42. The coprocessor must assert CpBusy during phase 2 of the "ALU" cycle of the coprocessor instruction to initiate such a stall. To terminate the stall CpBusy must be deasserted during phase 1. The cycle following this deassertion is the fixup cycle.

Figure 40
Main memory Write – With Write Busy Stall

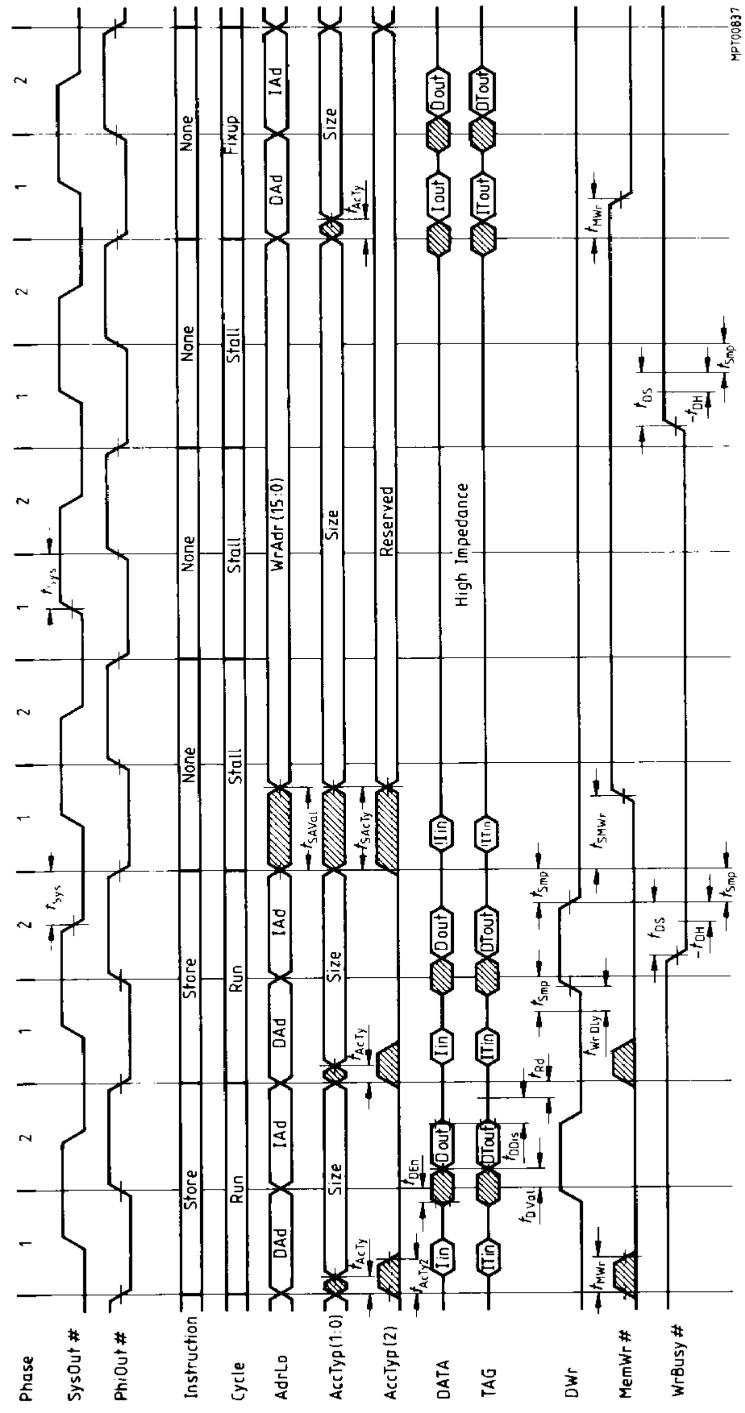


Figure 41
Interrupt Timing

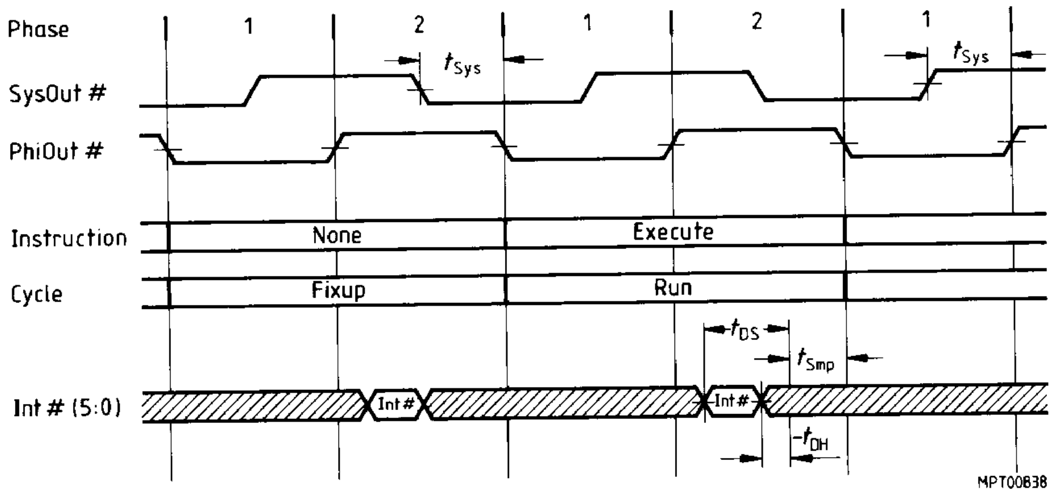


Figure 42
Mode Select Timing

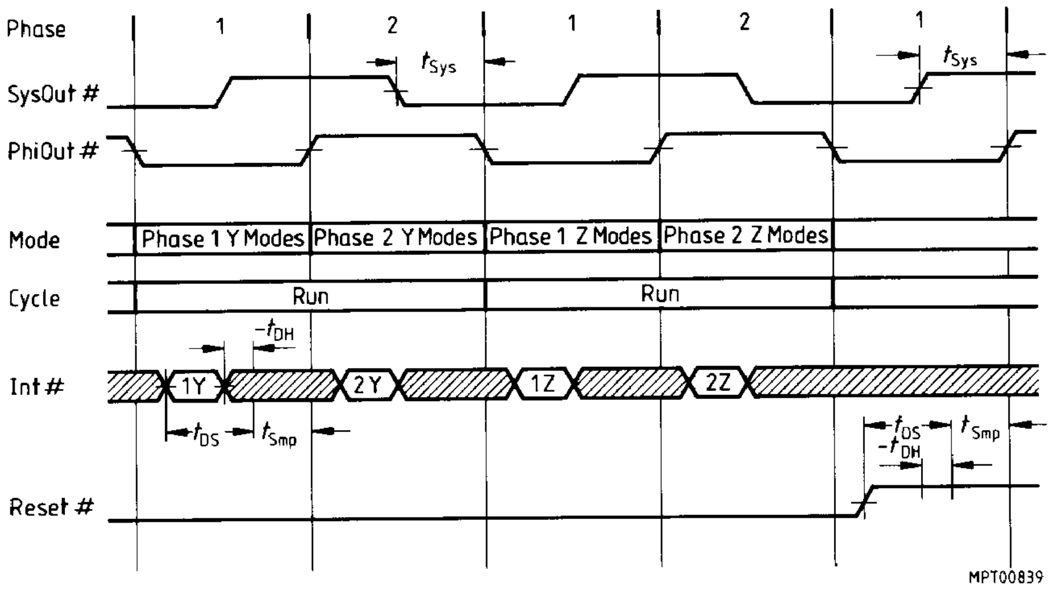
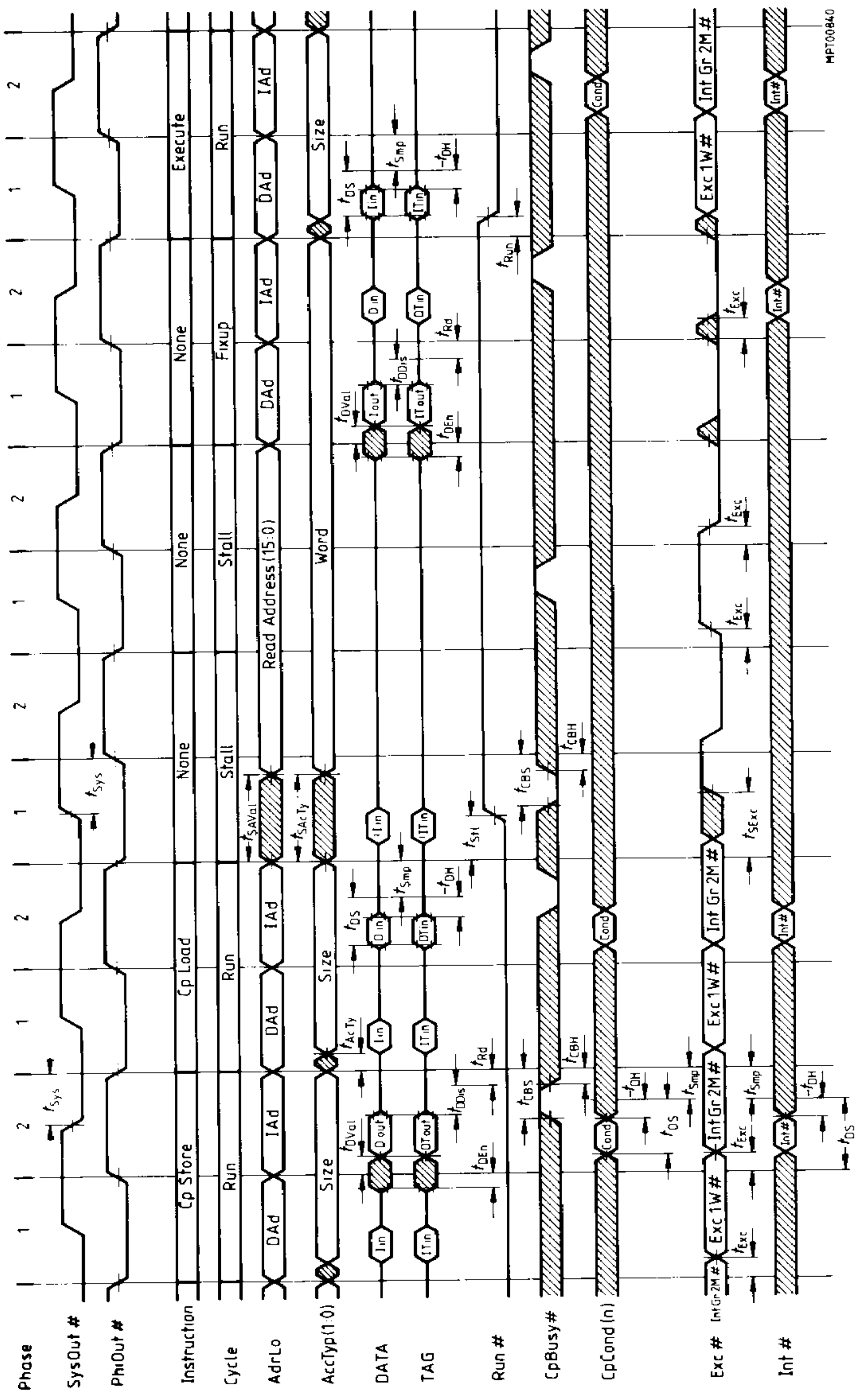


Figure 43
Coprocessor Interface Timing – With CoprocessorBusy Stall


Phase
SysOut #
PhiOut #
Instruction
Cycle
AdrLo
AccTyp(1:0)
DATA
TAG
Run #
CpBusy#
CpCond(n)
Exc #
Int #
Cp Store
Cp Load
None
Execute
Run
Stall
Fixup
DAd
IAd
Read Address (15:0)
Size
Word
Iin
Dout
Din
ITin
DTout
DTin
ITout
Iout
Cond
Exc 1W #
Int Gr 2M #
Int #
tSys
tDVal
tDDis
tDEn
tRd
tAcTy
tDS
tSmp
tDH
tSAVal
tSAcTy
tStl
tCBS
tCBH
tRun
tExc
tSExc
MPT00840