MIPS-X: The External Interface

Arturo Salz, Anant Agarwal, and Paul Chow

Technical Report No. CSL-TR-87-339

Computer Systems Laboratory
Department of Electrical Engineering
Stanford University
Stanford, CA 94305

Abstract

MIPS-X is a 20-MIPS-peak VLSI processor designed at Stanford University. This document describes the external interface of MIPS-X and the organization of the MIPS-X processor system, including the external cache and coprocessors. The external interface has been designed to optimize the paths between the processor, the external cache and the coprocessors. The signals used by the processor and their timing are documented here. Signal use and timings during exceptions and cache misses are also shown.

Key Words and Phrases: MIPS-X processor, RISC, external interface, cache, coprocessor.

Table of Contents

List of Figures

List of Tables

1. Introduction

MIPS-X is a single-chip RISC microprocessor designed at Stanford University. The processor is pipelined and executes an instruction every cycle. MIPS-X has a peak bandwidth to the external memory system of 160 MBytes/sec for instructions and data. The instruction bandwidth is reduced by about a factor of six because of an on-chip instruction cache! The on-chip cache can be tested or disabled using some pins in the external interface. In addition, MIPS-X needs to communicate with multiple coprocessors. The external interface allows the transfer of instructions and data between the processor and the coprocessors. MIPS-X can also operate in a demand-paged memory system environment and support maskable and unmaskable interrupts.

To relax the constraints on some of the critical paths in the memory system a late miss detect and a delayed write-back are used. To speed up floating-point load and store operations, a special floating-point coprocessor interface allows direct transfer of data between the floating-point coprocessor and the external cache. All coprocessor instructions are transmitted over the address bus instead of using an extra set of pins.

A basic system configuration of MIPS-X is described first. Section 3 then provides a specification of the external pins. Section 4 deals with external cache accesses for load and store instructions, and Section 5 details external cache accesses during instruction cache misses. Section 6 deals with coprocessors in general and Section 7 describes the special floating-point coprocessor interface. Exception handling is discussed in Section 8 and the last few sections discuss processor reset and the testing features.

This document covers two versions of the chip. The main description is for Revision 2 of the chip. Appendix II states the differences between Revision 1 and Revision 2.

2. The MIPS-X System

The MIPS-X system is a high performance, RISC multiprocessor. The major components of the MIPS-X system are a MIPS-X processor, an external cache containing both instructions and data (Ecache), and possibly several coprocessors.

The MIPS-X processor [2, 3] provides the following resources: an on-chip instruction cache with 512 words (2K bytes) of storage, virtual memory support, 32 general purpose registers, a 4 gigabyte addressing space, a coprocessor interface, and floating point support via a floating-point coprocessor. There are five pipeline stages. The stages are instruction fetch (IF), register fetch (RF), compute (ALU), memory access (MEM), and writeback (WB). The instruction is fetched

[1] The miss rate of the on-chip cache is about 10%. During a miss two instructions are fetched during the two cycles when the processor is stalled. This leads to an average instruction fetch rate of one instruction every six cycles, or one-sixth of the original requirements.

in IF, the instruction is decoded and registers fetched in RF, and result or address computation performed in ALU. During the MEM cycle, a possible memory access takes place, and during the WB cycle, the result (either from an ALU computation or data from a memory operation) is written into the registers or into memory.

The on-chip instruction cache (Icache) significantly reduces the bandwidth requirements of the processor and acts as a second port to memory. The presence of the Icache is crucial to achieving single cycle execution, without severely impacting the cycle time of the machine or using a large number of pins. The Icache has four sets, associativity equal to eight, and block size of 16 words. The replacement scheme is pseudo-random and is implemented using a ring counter. Simulations against several MIPS-X instruction traces show that the instruction cache will have a miss rate in the neighborhood of 10%. Details of the MIPS-X instruction cache design are described elsewhere [1].

Each MIPS-X processor may have several coprocessors. The number of coprocessors is not limited by the hardware, but by the number of bits in the coprocessor instruction that are used to specify the coprocessor number. The Programmer's Manual [2] shows three bits for up to eight coprocessors. Figure 1 shows the MIPS-X and coprocessor interconnection. It includes a MIPS-X main processor and three coprocessors. One coprocessor is special because load and store operations to its registers can take place without going through the MIPS-X registers. This coprocessor is intended to be the floating-point unit.

Accessing the Ecache is one of the critical paths of the system. To reduce the Ecache access time a direct-mapped virtual cache is proposed. A direct-mapped cache does not have the delay of an associative compare and using a virtual cache means that a large cache that does not require time for address translation can be built. Given current static RAM technology, a 16K-word to 64K-word direct-mapped cache is feasible. The proposed cache has a 4-word block size, and the entire block is fetched from main memory on a cache miss. The cache uses a write-back scheme to reduce traffic on the main memory bus. Table 1 presents some performance figures for the Ecache using simulations against realistic operating system and multiprogramming workloads.

Figure 2 shows three MIPS-X processors, their associated caches and how they could be connected into a shared-memory multiprocessor system with a single bus. The memory requests that are not satisfied by the external cache go to main memory. Important issues such as keeping the caches consistent in the shared-memory environment, I/O, and virtual to physical address translation are not discussed here.

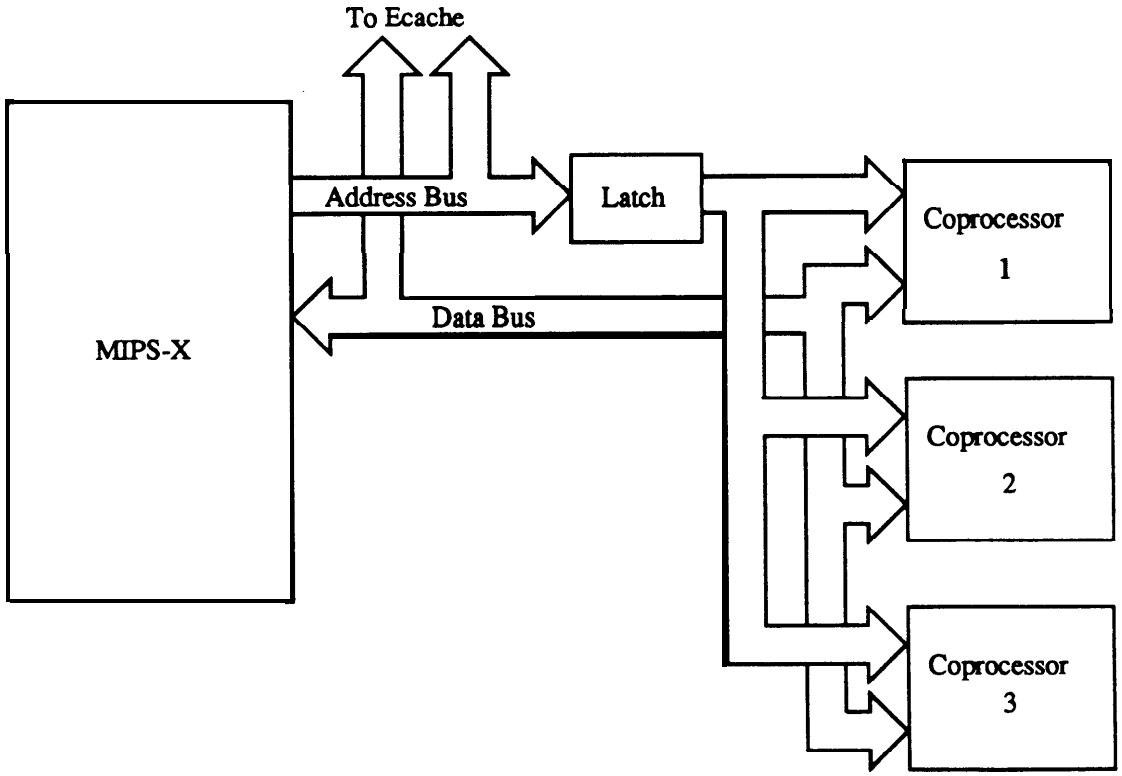


Figure 1: MIPS-X and coprocessors.

Cache Size	Miss Rate
16 KWords	2.94%
32 KWords	2.25%
64 KWords	1.66%

Table 1: Miss rates in the external cache

3. Signal Description

This section describes the functions of the pins on MIPS-X. A functional diagram of the pins is shown in Figure 3. A high signal represents a logic "1" level or active signal Logic "0" is represented by a low signal. An active low signal name has "_b" appended to it. Appendix I gives the pin numbers for the 144 pin PGA for the Revision 1 and Revision 2 parts. Appendix II describes the differences between the Revision 1 part and the Revision 2 part.

Power & Clock Signals

Vdd	5 volt power supply (12 pins).
Gnd	Ground (12 pins).
ϕ_1	Input. Phase 1 of a non-overlapping clock.
ϕ_2	Input. Phase 2 of a non-overlapping clock.

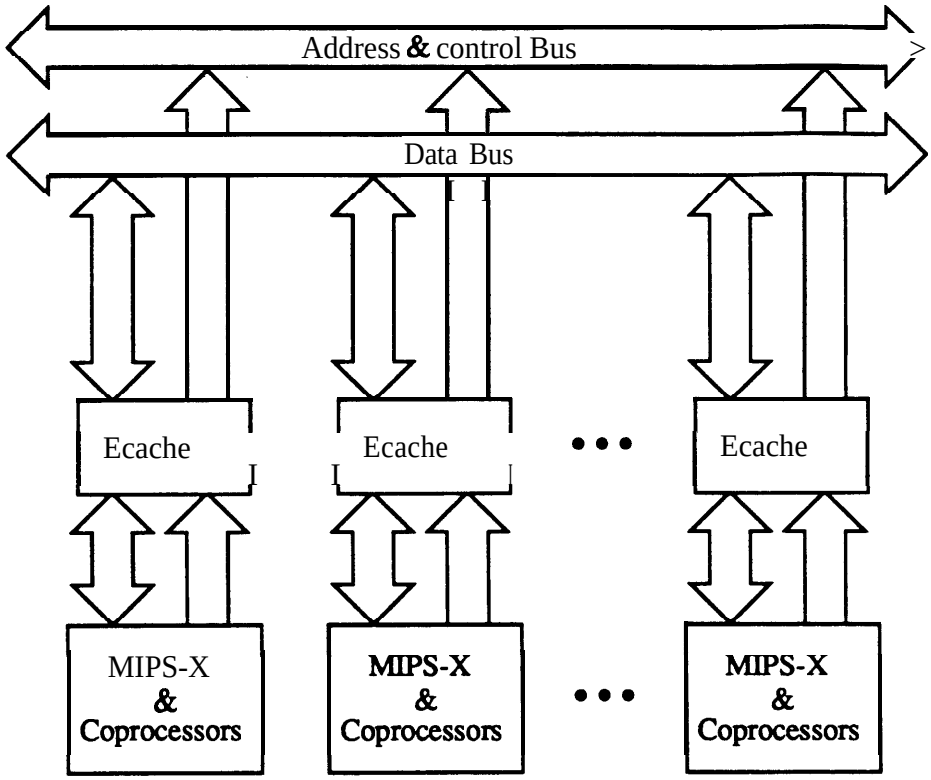


Figure 2: MIPS-X systems on a bus

System Buses

$Address_0$-$Address_{31}$	Three-state outputs. 32-bit Address bus with byte addressing. The two LSBs of the address ($Address_{30}$ and $Address_{31}$) are ignored by the memory system since the memory system access is word aligned. However, these two bits are needed for coprocessor instructions.
$Data_0$-$Data_{31}$	Input/Output. 32-bit Data bus.

Memory Control

MemCycle	Output. Indicates that the current cycle is a valid memory cycle. If this signal is not active, the memory subsystem should ignore the address and data lines during the cycle.
Read/ Write-b	Output. Defines the direction of a data bus transfer. A high signal means that a data bus read is occurring.
BypassCache	Output. The memory transaction should not use the external cache. This pin is active during the execution of the *ldt* (load-through) and *stt* (store-through) instructions.
AddressTristate	**Input.** Tristate the address bus. This signal connects directly to the pads and is independent of the clocks.

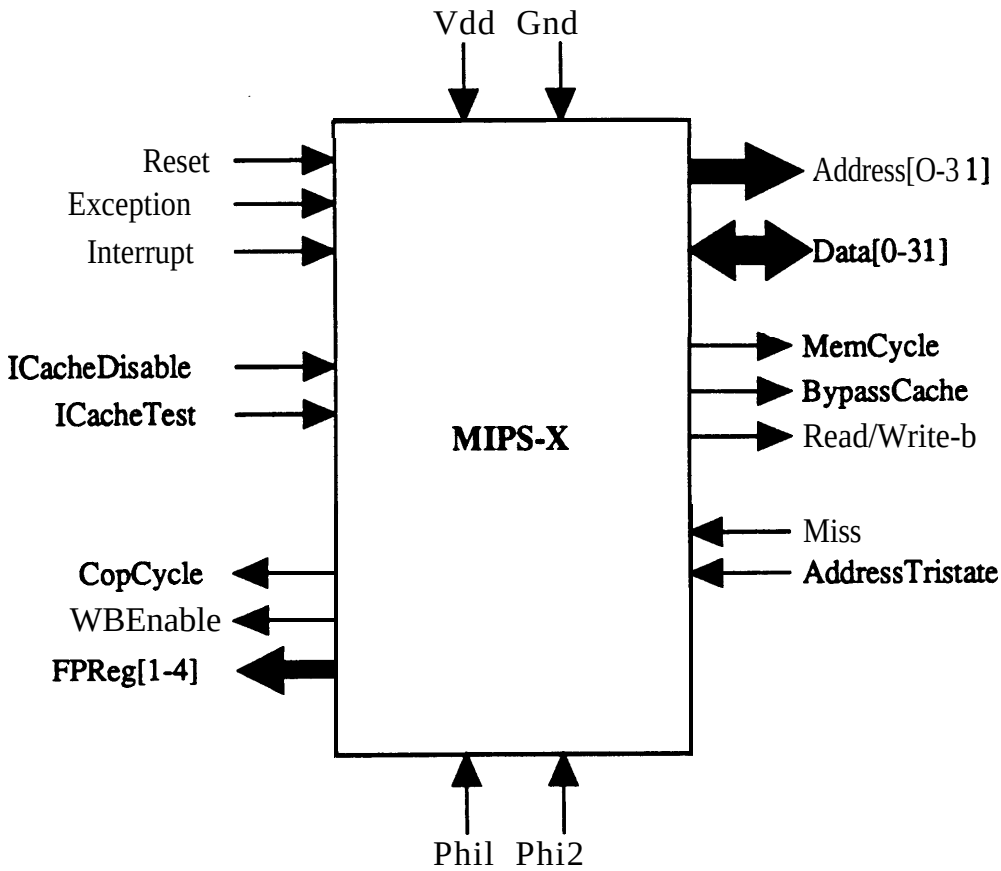


Figure 3: Functional signal diagram for MIPS-X

Miss — Input. This signal is used to stall the processor. See Section 4.1.

Exception Control

Reset — Input. Resets the processor. See Section 9 for more details.

Interrupt — Input. **Maskable** interrupt.

Exception — Input. **Non-maskable** interrupt.

Coprocessor Control

CopCycle — Output. Indicates that the **current** MEM cycle is a coprocessor cycle. If ***MemCycle*** is active then this is an FP load or store, otherwise it is one of the other coprocessor instructions.

WBEnable — Output. *Write Back **Enable*** indicates to the **coprocessors** that a **write-**back (or change in coprocessor internal state) of the last coprocessor instruction issued may be **performed.** See Section 6.1 for more details.

$FPReg_1$-$FPReg_4$ — Outputs. **4-bit** floating-point register number. Indicates which FPU register is to be used for data transfers between FPU registers and memory during ***ldf*** and stf instructions. An **FPU** load or store cycle

occurs when both ***CopCycle*** and ***MemCycle*** *are* active.

Other

ICacheDisable	Input. Disable the on-chip **Icache** causing the processor to execute cache-miss cycles.
ICacheTest	Input. Used to read and write the instruction cache directly. See Section 10 for more details.
Vbias	Input. This should normally be left open. There is an internal Vbias generator that connects to all of the latches to make them static. When this pin is grounded, the weak feedback is turned off and the latches become dynamic. This pin can also be driven to set a different bias level than the value generated internally.

3.1. Signal Timings

Table 2 gives the timing of the signals on each of the pins. In Table 2, a *stable* signal means that it is at the correct value for the entire phase. **A** *valid* signal means that the signal may change during the phase but it can be latched at the end of the phase.

OUTGOING SIGNALS:		
	Address$_0$ - Address$_{31}$	valid ϕ_2 of ALU, stable ϕ_1 of MEM
	Data, **- Data$_{31}$**	valid ϕ_1 of **WB** for *st* and MEM for *movtoc*, stable ϕ_2
	MemCycle	valid ϕ_2 of ALU, stable ϕ_1 of MEM
	Read/Write-b	valid ϕ_2 of ALU, stable ϕ_1 of MEM
	BypassCache	valid ϕ_2 of ALU, stable ϕ_1 of MEM
	CopCycle	valid ϕ_2 of ALU, stable ϕ_1 of MEM
	WBEnable	valid ϕ_2 of MEM, stable ϕ_1 of **WB**
	FPReg$_0$-FPreg$_3$	valid ϕ_2 of ALU, stable ϕ_1 of MEM
INCOMING SIGNALS		
	Data$_0$ - Data$_{31}$	valid ϕ_2 of MEM
	AddressTristate	asynchronous
	Miss	See Section 4.1.
	Reset	stable ϕ_2
	Exception	stable ϕ_2 (can monotonically rise with ϕ_2)
	Interrupt	stable ϕ_1
	ICacheDisable	stable ϕ_1
	ICacheTest	stable ϕ_1

Table 2: Signal timing

When an interrupt is acknowledged, further interrupts are masked by the processor. The *Interrupt* pin must be released before the processor unmasks interrupts in the PSW or the processor will begin another interrupt sequence. This pin is latched by the processor whenever ϕ_1 falls. When *Miss* is high, *the* ***Interrupt*** pin must be kept high until after both *Miss* and ϕ_1 fall.

The Exception pin must be high for a single ϕ_2 or it can rise monotonically with ϕ_2. In the processor, it will discharge a precharged line. The signal is not latched internally, otherwise it would take one extra cycle before the exception sequence could begin. *Miss* prevents state changes so *Exception will* only be recognized if it is high for the ϕ_2 preceding *Miss* falling.

3.2. MIPS-X Clocks

MIPS-X uses a 2-phase, non-overlapping clock. These two phases are called ϕ_1 and ϕ_2. The Revision 1 parts have been tested up to 16 MHz and the Revision 2 parts should run between 16 and 20 MHz but at the time of this printing the Revision 2 parts are not available for testing.

3.3. Pad Loading

The pads have been designed to drive a 50 pf load.

4. Memory References

This section describes the MIPS-X memory interface. It is intended that an external cache (Ecache) be used to give the processor fast access to data and instructions. For multiprocessor systems, the Ecache is also needed to reduce the traffic on the main bus. Details of the Ecache will not be discussed here but it is assumed that some of the signals are generated by the Ecache. It is also *assumed* that *the* Ecache is a *write-back* cache.

The organization of the Ecache and its interface is shown in Figure 4. The Ecache supplies both instructions and data to the processor and handles them in the same way. The processor does not indicate whether it is doing an instruction or data reference. Instructions that miss in the on-chip Icache are retrieved from the Ecache. The controller for the Ecache is a coprocessor. It supplies the process-id (PID) on an Ecache access.

MIPS-X achieves a fast external cache interface by employing two key features that will be discussed in this section:

- a *late miss* signal (called *Miss).*
- *delayed writes* into the Ecache.

These features provide the Ecache with some extra time for the time critical operations of detecting misses and writing data back into the cache.

For load and store operations, the address is valid ϕ_2 of the ALU cycle. The address is latched in an external latch by the falling edge of ϕ_2, and presented to the memory subsystem (the Ecache, and in its absence, memory) during the MEM cycle. This should provide the memory system with enough set-up time for the address lines. The memory control signals (*MemCycle*, *Read/Write_b*, *Bypass Cache* and *CopCycle*) are sent out at the same time as the address.

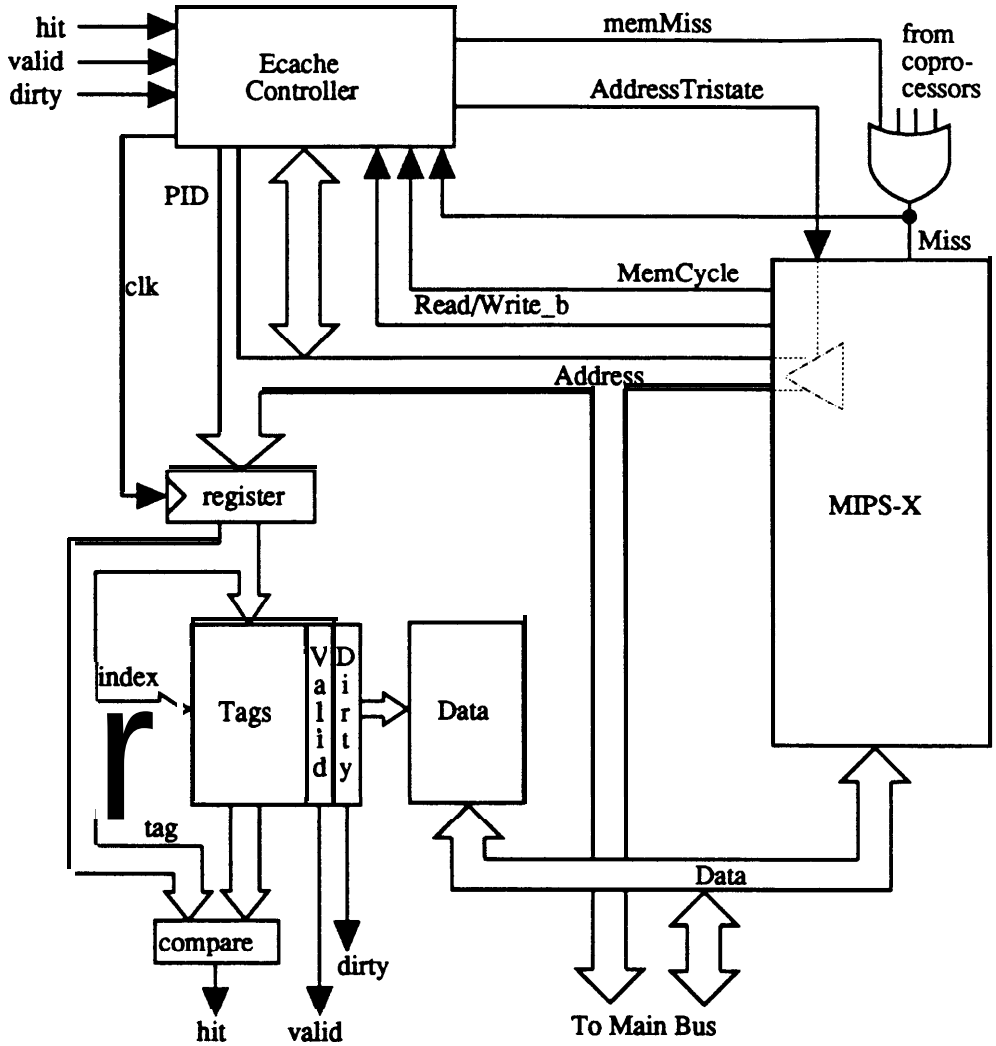


Figure 4: Ecache interface

4.1. Miss

The Miss signal is used to stall the processor, particularly during memory or cache accesses. Internally, the processor does state changes on ϕ_1 when the *Miss* signal goes low. When the *Miss* signal is high during ϕ_1 the processor is stalled. *Miss can be* lowered up to 10 ns before ϕ_1 falls and still allow the processor to successfully complete the ϕ_1 clock phase.

4.2. **Late Miss**

Typically, for correct operation, the processor should save the data from a load only if it has been determined that the data resulted from a hit in the cache. It is known that the critical path in the Ecache is not getting the data to and from the processor pins; it is determining whether the reference hit in the cache. The data from the cache is written into the processor registers during ϕ_1 of the WB cycle but to allow more time for the hit detection, MIPS-X allows the *Miss* signal to fall after ϕ_1 of WB rises. This means that if Miss does not fall, the processor has written invalid data into its register. In this case, the processor continues to re-execute the load cycle while the Ecache fetches the correct data. When the correct data is being presented to the processor, the *Miss line* is released, and the processor continues with the correct data in the register.

To make this scheme work, the compiler must ensure that a load instruction does not use the same register for both the address calculation and the destination. This will insure that all loads are idempotent (can be repeated indefinitely and give the same result).

4.3. **Memory Read**

Figure 5 shows the basic timing for a load instruction. A normal read begins with the processor presenting the new address to the cache by the end of ϕ_2 of ALU of the load instruction. This address is latched in the external address register, and driven to the memory *array*. *MemCycle* and *Read/Write-b* will be high for this cycle. The memory system must present valid data to the processor before the falling edge of MEM ϕ_2. The *Miss* signal should be raised before ϕ_1 rising and lowered during ϕ_1 if the data is valid. If the memory system is not fast enough to keep up with the processor, then it must stall the processor by keeping the *Miss* signal high.

While the data is latched by the processor, the Ecache controller is performing the tag compare . The *Miss line* is pulled low during the ϕ_1 of WB if the reference hits in the cache. If the read generates an Ecache miss, then *Miss* is kept high and the data has to be retrieved from main memory. While the processor is stalled, the Ecache presents the address to main memory which will return the requested data to the Ecache. When the data is finally received from memory, the Ecache lowers *Miss* and the processor continues as usual.

Some complications arise in the Ecache control because of the late miss detection and the Ecache system having a two-stage pipeline. Every cycle, an address is latched externally during ϕ_2 falling and supplied to the Ecache. When there is an Ecache miss during the first load of a back-to-back load operation, the address of the second load will displace the address of the load that missed. To handle this event, the Ecache controller must keep the address of the first load in an internal register and then reload it in the address register. This is done by first tri-stating the processor address pins and then driving the address bus with the address that missed. Figure 6

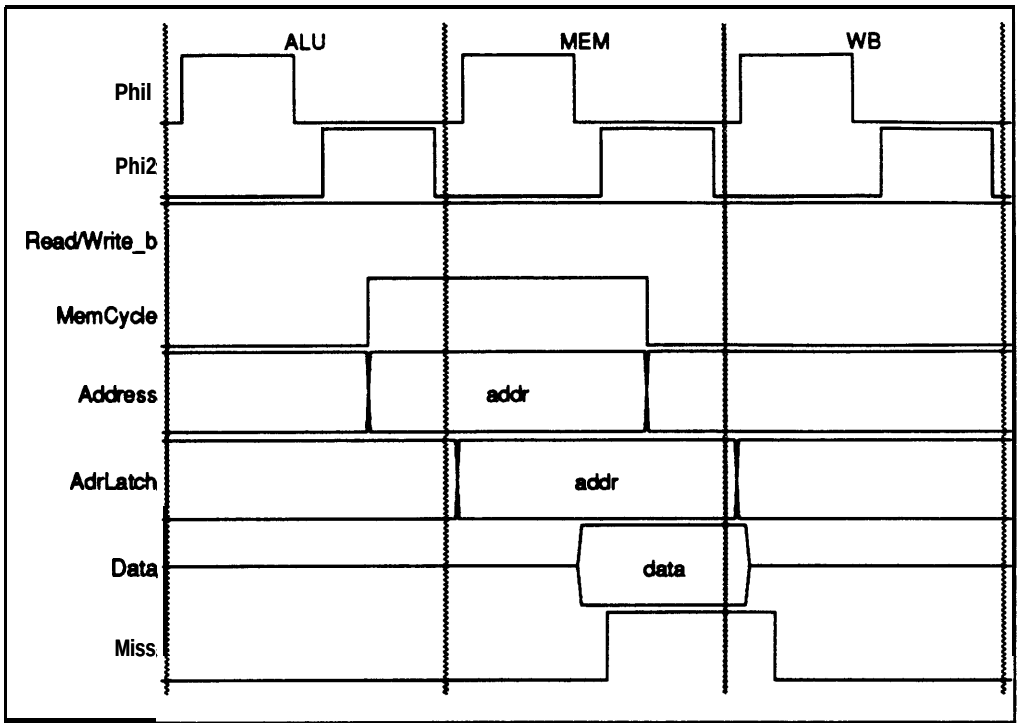


Figure 5: Timing diagram for a load instruction

shows this sequence.

4.4. Memory Write

To reduce the main memory bus traffic in a multiprocessor configuration **MIPS-X** assumes that a write-back scheme is used for the Ecache. Under this scheme, writes are only updated in memory when either the Ecache is flushed or a block that needs to be replaced is found dirty. Therefore, before writing data into the Ecache, the miss and *dirty* conditions must be determined. If a *miss-and-dirty* condition is encountered, the current cache entry must first be updated in memory and only then can the data be written into the cache. This means that a store instruction needs two memory cycles: one to probe the cache to check if the block exists and one to write the data. The instruction following the store instruction should not use its MEM cycle (the WB of the store instruction) so the Ecache can use this cycle to write the data without stalling the processor. The architecture disallows stores followed by another memory or coprocessor instruction but there is no hardware interlock to prevent this. The compiler reorganizer is responsible for scheduling the right instruction here[2].

[2]There is a hardware interlock that prevents an Icache miss from occurring during the second cycle of a store instruction. Section 5 explains this in more detail.

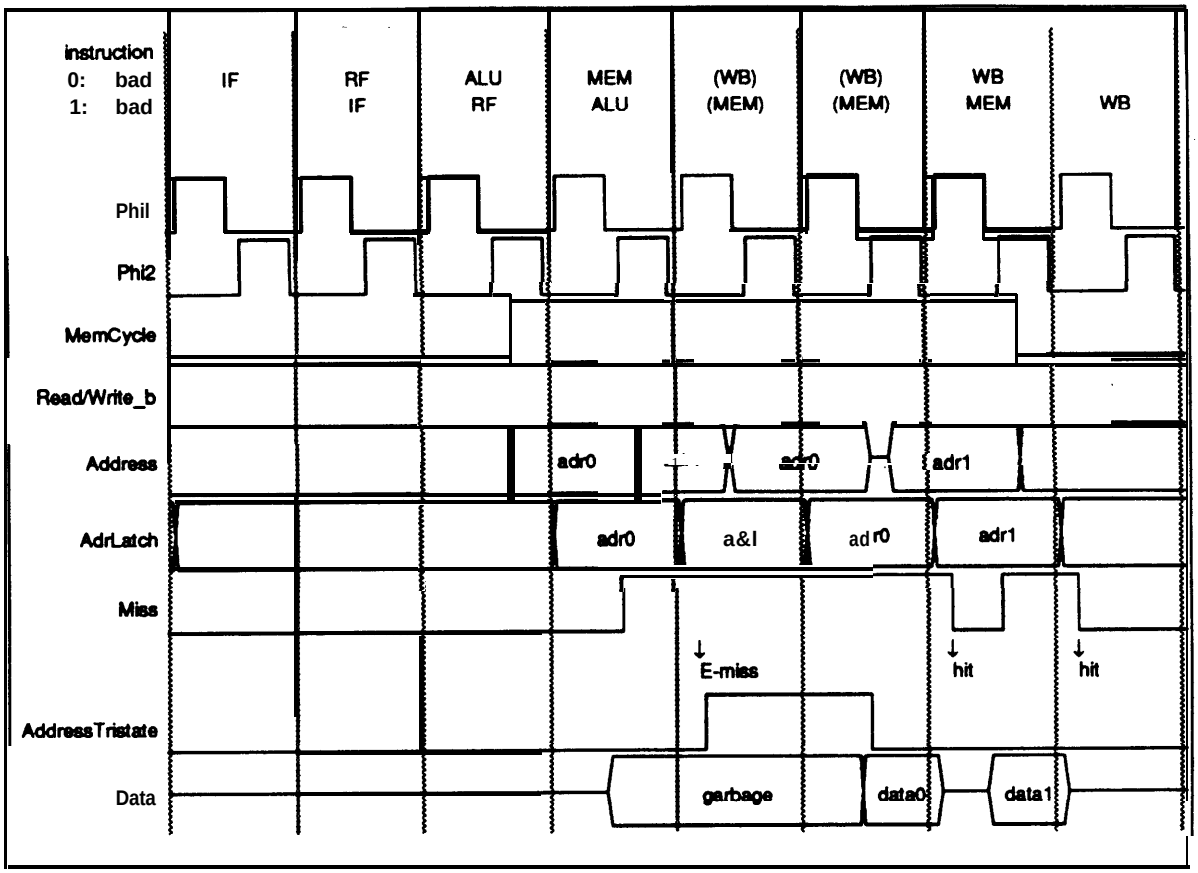


Figure 6: Timing diagram for Ecache miss during consecutive loads

Figure 7 shows the timing for a store instruction. The write operation begins like a read, except that *the **Read/Write_b*** signal is low. Data is driven out of the processor when Miss falls during ϕ_1 of WB and remains on the bus until ϕ_1 rising of the next cycle. The Ecache writes the data during WB (the MEM slot of the next instruction). This scheme is known as the ***delayed write.*** Note that raising Miss before WB will not stall the write; the data will be on the bus for exactly one cycle, independent of *Miss.*

A sequence of memory instructions is shown in Figure 8.

4.5. Bus Locking

There is no provision in the MIPS-X processor for implementing synchronization primitives such as a read-modify-write instruction. Two schemes have been proposed to accomplish synchronization. One is to use strictly software synchronization such as Dijkstra's or Dekker's algorithm. The other is to implement coprocessor instructions in the Ecache controller. A possible coprocessor instruction is one that locks the bus on a read and unlocks it on the next

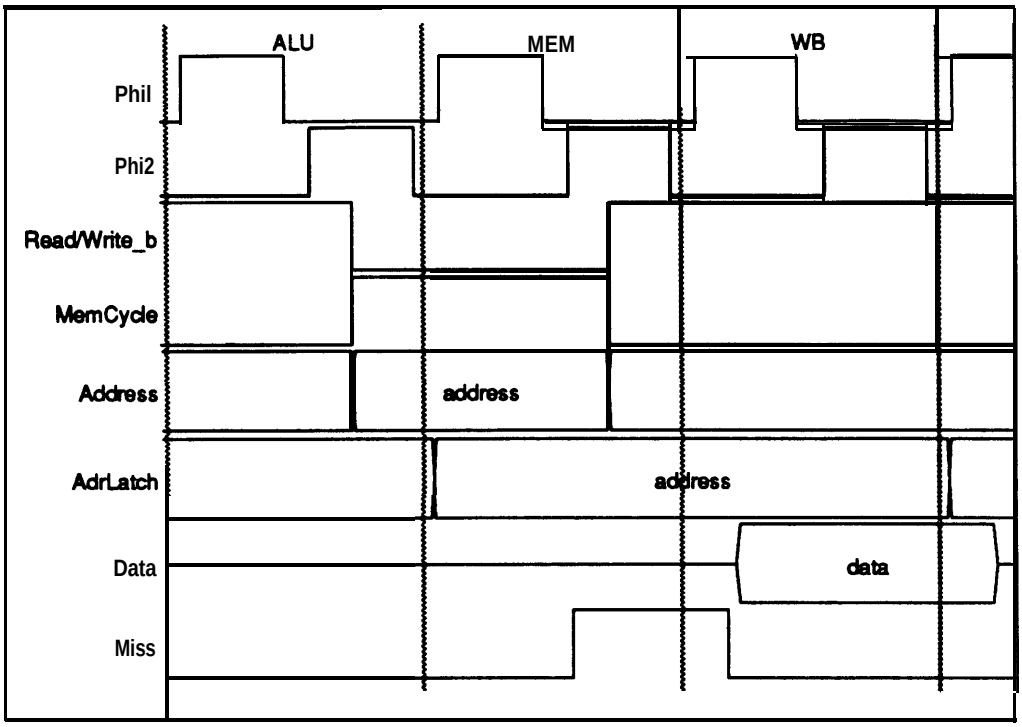


Figure 7: Timing diagram for a store instruction

store instruction. This is effectively a read-modify-write instruction.

5. Icache Misses

The timing for an **Icache** miss cycle looks exactly like a load instruction to the **Ecache.** During an **Icache** miss, two instructions are fetched during two cache miss cycles. Exceptions during an **Icache** miss are handled in the same way as those during normal operation. Figure 9 describes an **Icache** miss sequence for each of the pipeline stages. Figure 10 shows the timing of the external pins for a typical instruction sequence.

In Figure 9, assume that instruction 5 misses in the instruction cache. The following enumeration numbers correspond to the superscripts in Figure 9.

1. Instruction 5 misses in the **Icache.** The miss sequence starts from the next cycle.
2. Instruction 2 is closest to completion at this point. An instruction can commit only in its WB phase (stores are an exception) and special care needs to be taken to see that the instruction does not change processor state before the miss is handled.
   - If instruction 2 is an ALU operation then the WB is postponed until after the miss sequence.
   - If instruction 2 is a load, then the data from memory is latched into the input memory data register, the register that latches input data before it is written

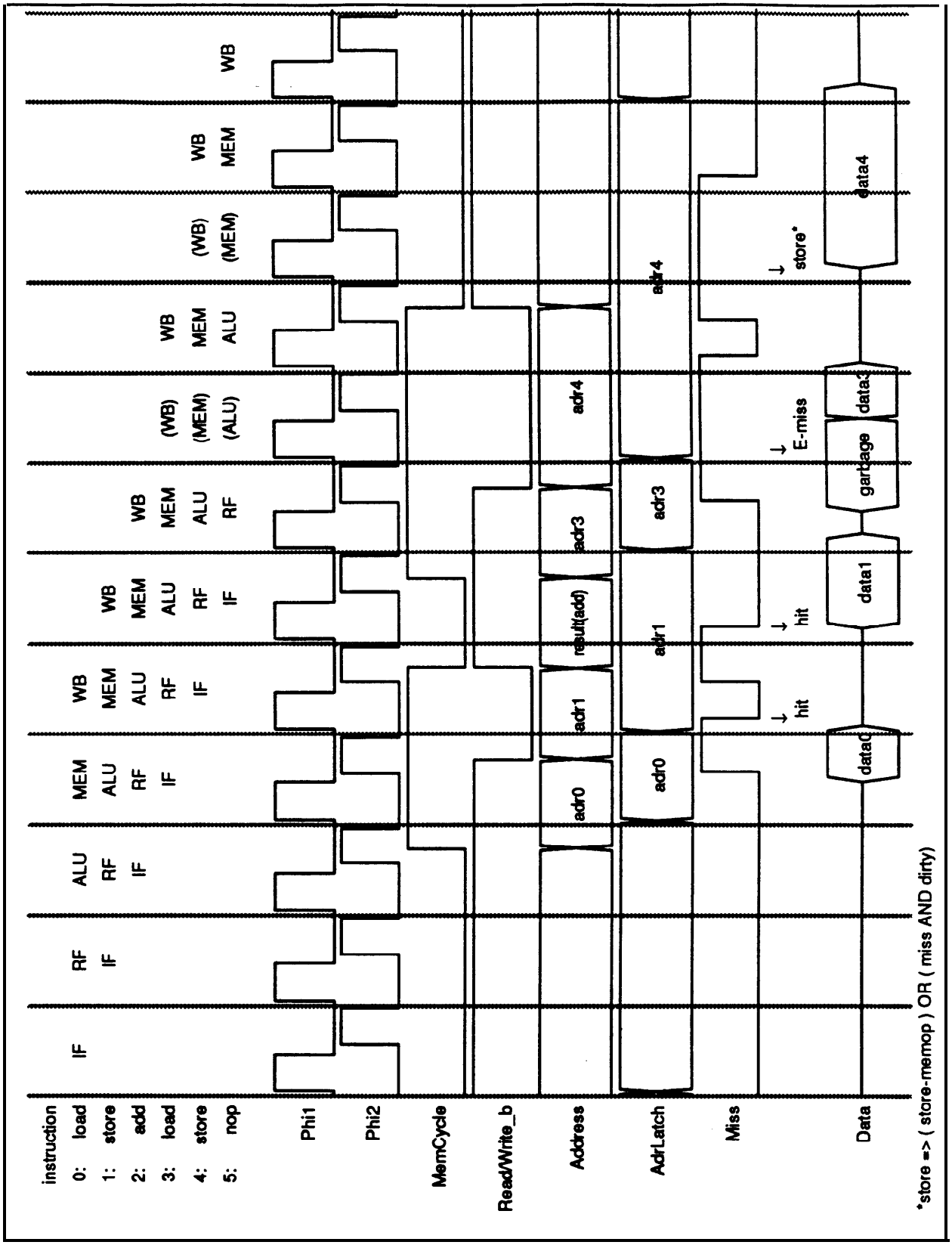


Figure 8: **Timing diagram** for sequence of memory instructions

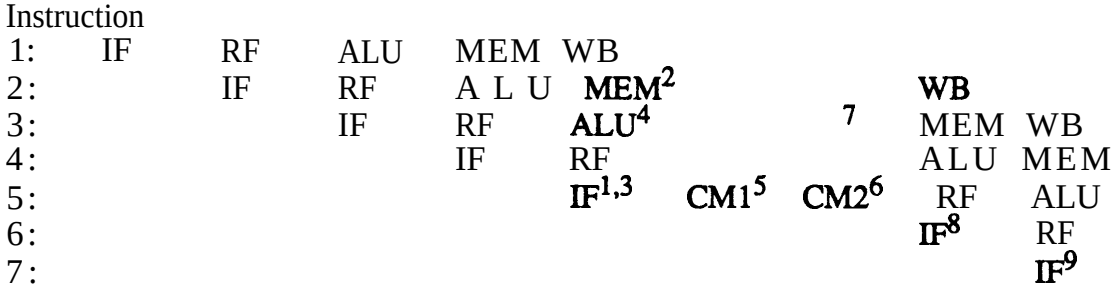


Figure 9: Icache miss timing

into the register **file.** If the memory operation corresponding to instruction 2 fails **(Miss** is active) then the sequence of events that follow at the Ecache is the same as an Ecache miss during the first load of a load-load sequence. The timing for this **sequence** was shown in Figure 6. The Ecache controller loads the address register with the load address of instruction 2 (note that this address was displaced from the address register by the address that came from the processor) and completes the load. The **Miss** signal stays high causing the processor to stall throughout the load miss sequence.

- If instruction 2 is a store then the only hardware interlock in the processor takes effect. Recall that store instructions require two cycles to complete and the software ensures that the instruction after the store does not use memory. However, there is no way to predict an **Icache** miss, which also needs to access memory, so a hardware interlock is implemented that inserts an extra cycle to complete the store before the **Icache** miss begins. After the **Icache** miss cycles are completed, instruction 2 will execute a null WB cycle.

  If an exception occurs during the **Icache** miss cycles, the store instruction should be re-executed but the store has actually completed. Before returning from an interrupt, the interrupt handler must convert the **first** instruction to a ***nop*** if it is a store instruction. This is done by **changing** the restart address for that instruction to Address 0 where there is a ***nop***[3]. This action need not be done if it is **known** that all store instructions are idempotent. Writes to shared memory or device registers may have side effects that render them non-idempotent.

3. The address of instruction 5 is sent out.
4. The address of instruction 5 overrides a possible data address due to instruction 3. Essentially, the internal **Icache** miss signal causes the PC value to be selected instead of the address for the memory access.
5. Instruction 5 returns from the Ecache. The address of instruction 6 is sent out.
6. Instruction 5 is copied into the **Icache.** Instruction 6 comes back from the Ecache.
7. The address corresponding to a possible memory-reference by instruction 3 is

[3]The architecture specifies that a *nop* instruction be found at Address 0 in both system and user space. This allows instructions to be *squashed* on restart by setting their PC address to zero.

placed on the address bus by MIPS-X and latched into the address latch at the Ecache.

8. Instruction 5 is decoded to begin its RF phase. Instruction 6 is copied into the Icache. It is "fetched" at the same time.
9. The RF of instruction 6 is started. Instruction 7 is fetched from the Icache (unless another Icache miss occurs).

6. Coprocessors

Coprocessor instructions are cached in the main processor and hence have to be sent to the appropriate coprocessor for execution. Ideally, the coprocessor instructions should be sent over a separate bus, but because this requires a large number of pins a scheme that sends coprocessor instructions over the address bus is used. The coprocessor instructions are really memory type instructions with the offset field containing the instruction for the coprocessor. All loads and stores of coprocessor registers must go through the MIPS-X registers except for one special coprocessor which would usually be a floating-point unit.

There are three instructions that deal with the general coprocessor interface. They are move-to-coprocessor (*movtoc*), move-from-coprocessor (*movfrc*) and coprocessor-alu (*aluc*). The first two are used to transfer data between the coprocessor registers and MIPS-X and the last one is to initiate coprocessor operations. *The aluc* instruction is actually a *movfrc* instruction with the destination register set to register 0[4].

The floating-point load and store instructions are described later in Section 7. Detailed descriptions of these instructions are given in the Programmer's Manual [2].

The timing of the coprocessor instructions is similar to the timing for load and store instructions. During the ALU cycle, the coprocessor instruction is driven on the address bus as if it were an address. This is latched by the coprocessor latch on ALU ϕ_2 falling. The *CopCycle* signal is asserted at the same time, notifying the coprocessors that the address bus contains a coprocessor instruction; the *MemCycle* signal is inactive, so the memory subsystem will ignore the address bus during this cycle. The addressed coprocessor will then execute the instruction as follows:

- movtoc - The MIPS-X source register is placed on the data bus at MEM ϕ_1 rising or when *Miss* falls. If necessary, the coprocessor can stall MIPS-X until it has decoded the instruction by using the *Miss line. The data stays* driven until *Miss* is released during WB ϕ_1. The data is latched by the coprocessor on MEM ϕ_2 falling. The coprocessor should only write the destination register if the *WBEnable signal* is

[4]The current software system generates *movfrc* instructions but *movtoc* instructions could also be used if the coprocessor ignores the data on the data bus during this cycle. The timing diagrams in this document assume that *movtoc* instructions are used.

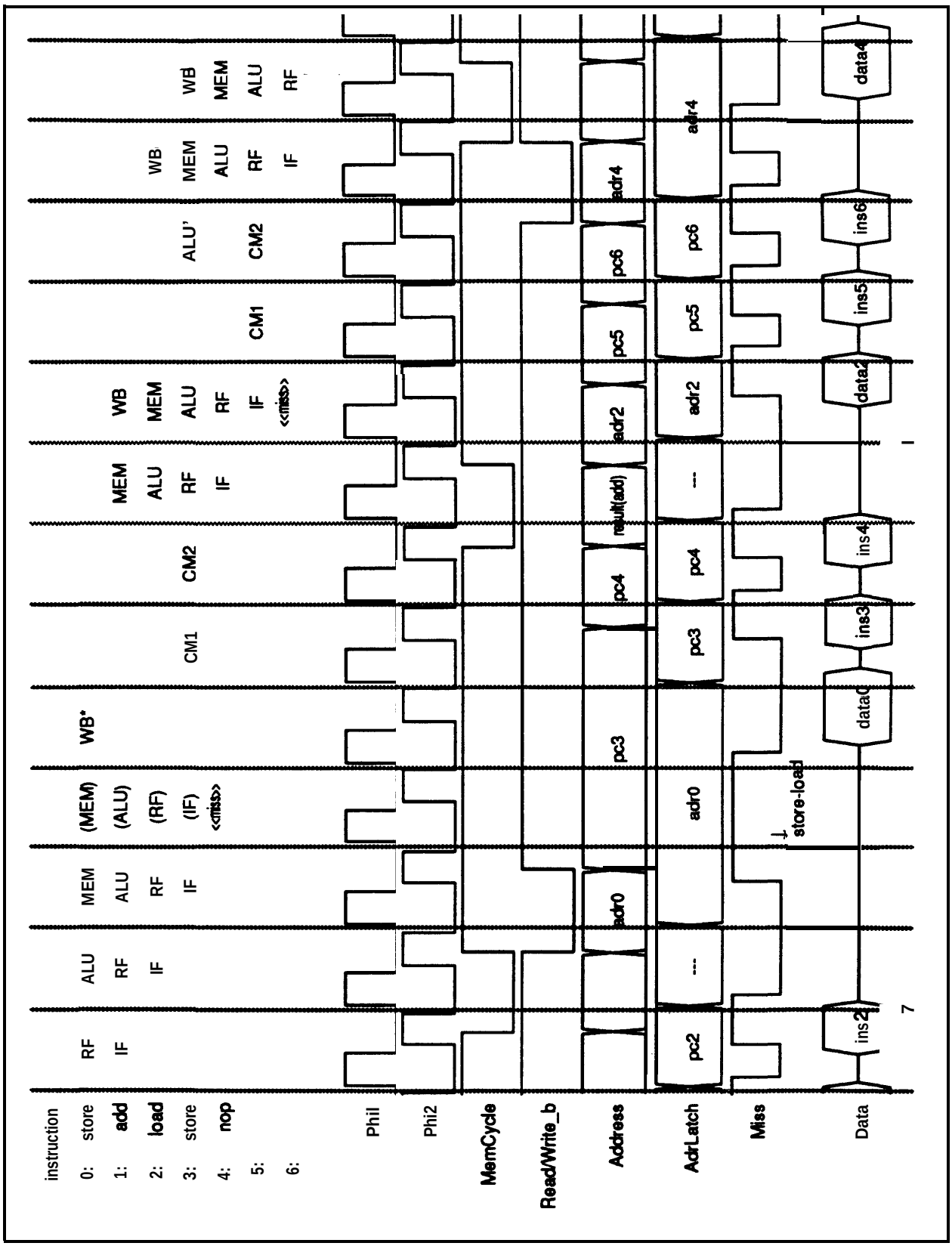


Figure 10: Timing diagram for an Icache miss

activated by MIPS-X as explained later in Section 6.1. Note that this instruction is like a store instruction except that only one memory cycle is needed because a cache probe is not necessary.

- **movfrc -** The coprocessor places the contents of the appropriate register on the data bus before MEM ϕ_2 falling. If necessary, MIPS-X can be stalled if this cannot be done in time since the coprocessor only has one cycle to do this? MIPS-X will latch the contents of the data bus on MEM ϕ_2 falling.
- **aluc -** The coprocessor decodes the instruction and starts executing the instruction. If movfrc instructions are used, then the data on the data bus is read by MIPS-X and dumped into register 0. If *movtoc* instructions are used, then MIPS-X places register 0 on the data bus. This should be ignored by the coprocessor.

Coprocessors should stall the main processor by keeping *Miss* high if they require more time to finish an operation .

6.1. Write-Back Enable

The external interface of MIPS-X provides for restartability of coprocessor instructions. This is necessary if the processor has to operate in a system where interrupts or other exceptions may occur. An instruction should not change processor state until it has reached the WB stage on the main processor. To tell a coprocessor that an instruction can complete, a signal *called* ***WBEnable*** is asserted by MIPS-X on the WB cycle of the coprocessor instruction. Before this signal is received, the coprocessor must *retain the* result of *the* operation internally *until WBEnabZe* is activated (if at all). If the coprocessor implements idempotent operations then it may ignore the ***WBEnable*** signal and commit as soon as it has finished execution.

Figure 11 shows the timing for the execution of four consecutive coprocessor instructions: *movtoc*, ***movfrc***, ***aluc*** and ***aluc***. Instruction 4 stalls MIPS-X; this is to show what happens when an instruction is sent to a coprocessor that can execute only one instruction at a time, and has not finished executing the previous instruction sent to it. Also, note that when the coprocessor stalls MIPS-X, the Ecache must also stall. Otherwise, it would read the incorrect data from the data bus. This shows that all devices must look at the ***Miss*** signal.

If the MEM cycle of a coprocessor instruction coincides with an interrupt or page fault, the ***WBEnable*** signal will not be generated and the instruction will be re-executed after the interrupt has been serviced. When a coprocessor is waiting for ***WBEnable*** to complete its current instruction and *another* coprocessor instruction is issued before ***WBEnable***, *the* current

---

[5] This will probably be the case since a 50ns clock cycle will not be enough time for the coprocessor to read the instruction, decode it, decode the register number, index into its register file, and place the contents on the data bus. An alternative solution that would have provided the coprocessor more time to decode the register number would have been to send out the register number in the RF phase on the register number lines provided for the floating-point unit, but this meant a non-trivial change in the MIPS-X processor implementation and was discarded.

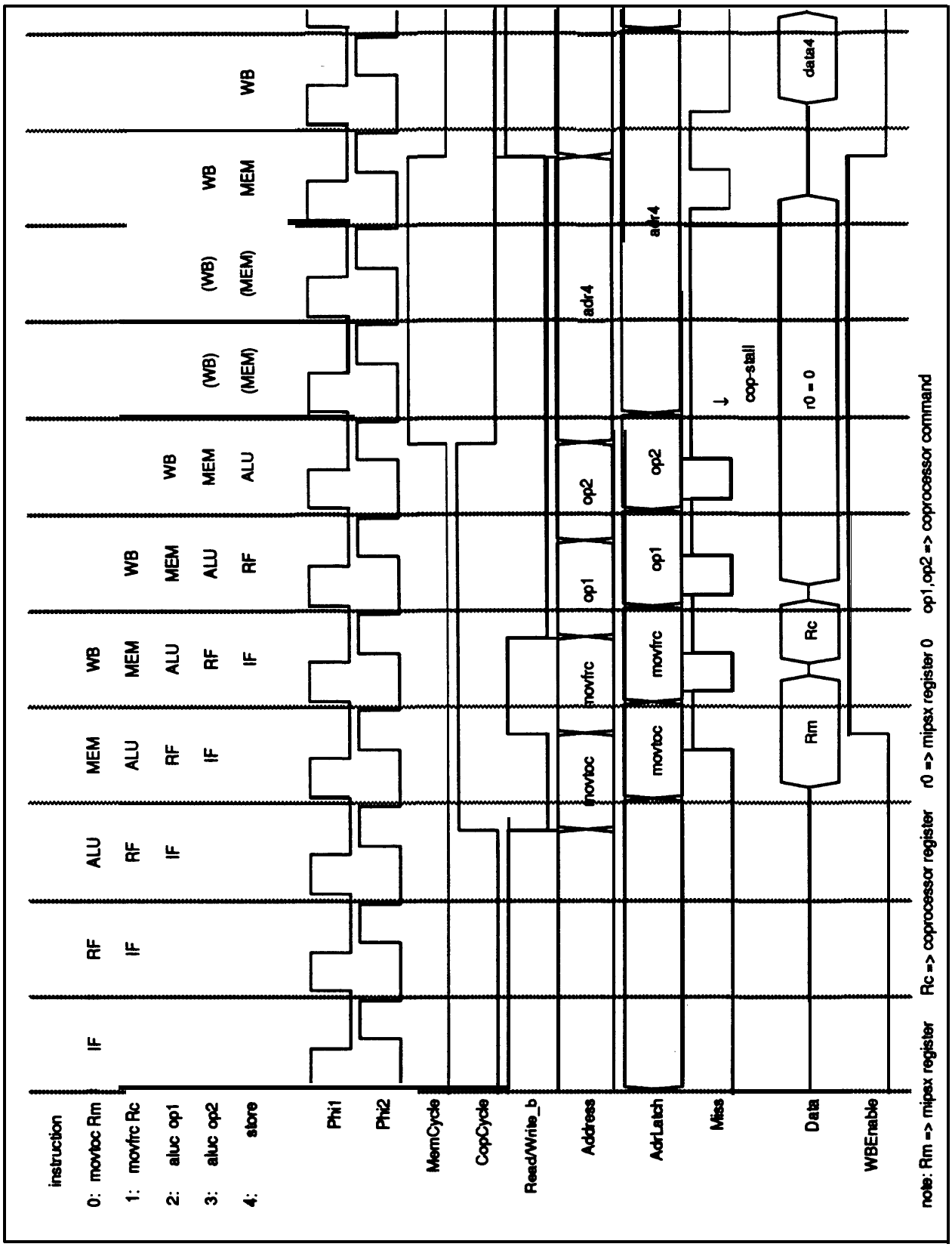


Figure 11: Timing diagram for coprocessor instructions

instruction is squashed. The coprocessor should start the new instruction if it was the coprocessor addressed. If a coprocessor *sees* a ***WBEnable*** before or concurrent with any other coprocessor instructions issued, then it should complete its current instruction.

If an **Icache** miss occurs during the MEM cycle of a coprocessor instruction, activation of the ***WBEnable*** signal is delayed until after the CM1 and CM2 cycles because the WB cycle is delayed. *If* no exception *occurs* during these *two* cycles then ***WBEnable*** *will be* activated. When *an* exception *occurs* during *the* CM cycles, ***WBEnable*** *will* not be generated and the instruction will be restarted later. Figure 12 shows the delaying of *the* ***WBEnable*** during an **Icache** miss followed by a coprocessor instruction that gets squashed due to an interrupt.

There are at least two ways the coprocessor can behave if it is sent back-to-back instructions:

- The coprocessor can be pipelined at least two stages: One for executing and one for **performing** the write-back. This allows the main processor to send data and instructions to the coprocessors one immediately after another as shown in Figure 12. It may be necessary to bypass the coprocessor Result register because the next instruction may need to use it before it has been written back.
- The coprocessor first performs the write-back and then starts executing the new instruction. In this case, the main processor is stalled until the pending write-back is performed and the coprocessor has had time to decode the new instruction.

A better option would be to resolve these interlocks by software, but this may not be possible for all coprocessors, because coprocessor instruction execution times will vary from coprocessor to coprocessor, from system to system, and even from instruction to instruction on the same coprocessor. Nevertheless, it is still worthwhile to try to schedule useful operations between coprocessor instructions when the execution time can be estimated.

The need to look at *the* ***WBEnable*** signal before completing the execution of instructions on coprocessors can complicate the coprocessor control. An alternative solution that obviates the ***WBEnable*** signal is to guarantee that non-idempotent instructions are not restarted on the coprocessors. Then the coprocessor instruction that is in its MEM phase when an exception occurs can be allowed to complete in the coprocessor. Such a feature can be implemented in the software interrupt handler. During the return from interrupt, the handler should check to see if the first instruction to be restarted is a non-idempotent coprocessor instruction and convert it to be the ***nop*** instruction at Address 0. This is similar to the scheme used to prevent stores from **re-executing** after an interrupt.

7. Floating-Point Unit

The special interface for the floating-point unit tries to optimize floating-point loads and stores. The fact that all loads and stores of coprocessor registers must go through the general purpose registers of the main processor was considered unsatisfactory far the floating-point unit

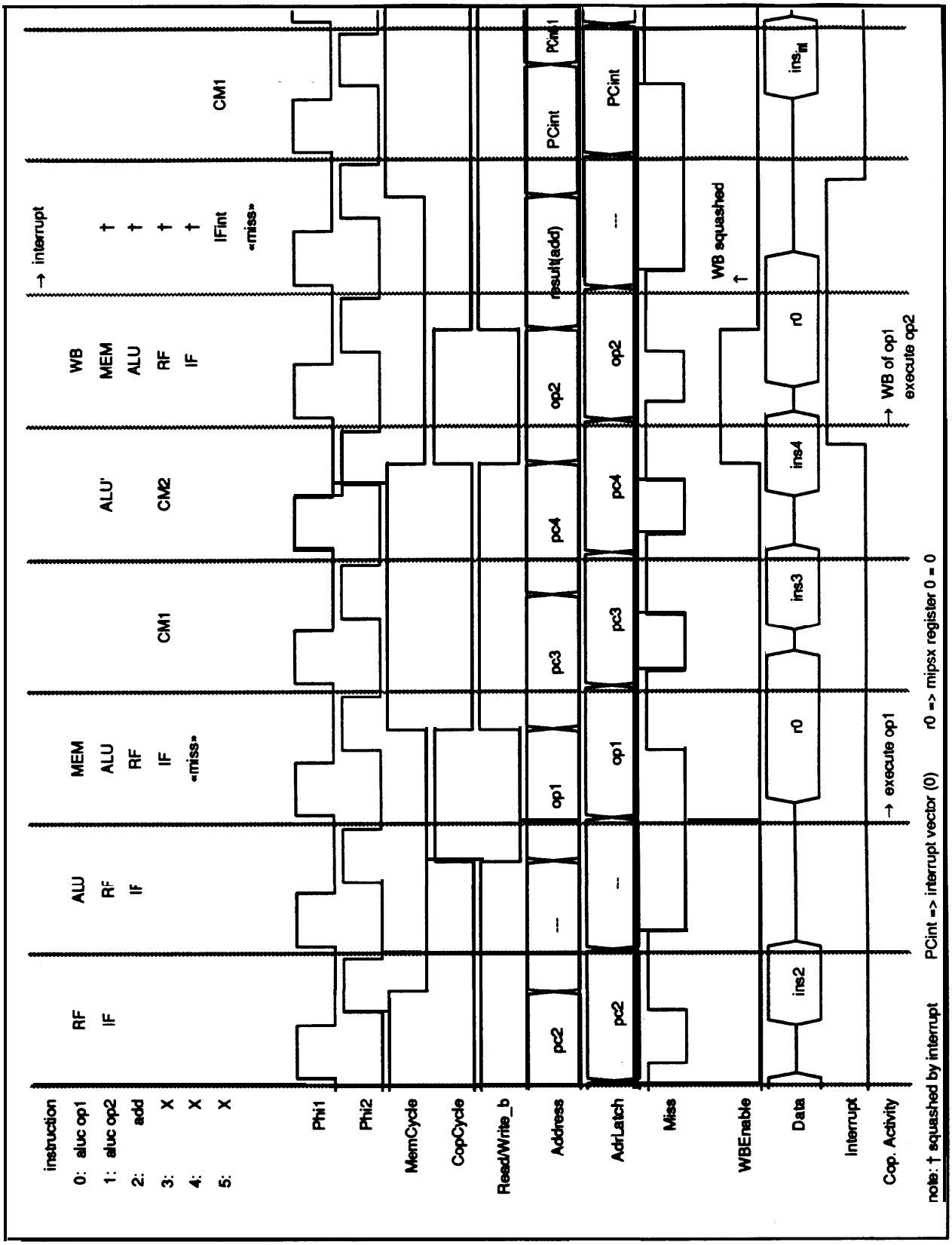


Figure 12: Timing diagram of *the* ***WBEnable*** signal for coprocessors

(FPU) since it could be used heavily. The coprocessor interface was extended to allow loads and stores directly between memory and the FPU by including two more instructions for the floating-point coprocessor. Load float (*ldf*) loads a floating-point register from memory and store float (*stf*) stores the contents of a floating-point register into memory.

The two instructions require four pins dedicated to *the* FPU: ***FPReg1 - FPReg4***. *The timing* of these two instructions is identical to memory loads and stores. For a ***stf***, MIPS-X performs a store cycle by generating the memory address and signaling the FPU to provide the data word. For a ***ldf***, MIPS-X performs a load cycle by generating the memory address and signaling the FPU to get the data from the data bus. The FPU source or destination register for a ***ldf*** or ***stf*** operation is sent to *the* FPU by means of the ***FPReg1 -*** *FPReg4* signals, whose timing is identical to the address lines.

Figure 13 shows the timing of the various signals required for a stream of floating-point instructions: two ***ldf***'s to load the operands followed by an ***aluc*** (the floating-point operation) to be performed on the two operands and a ***movtoc*** to move the result into a MIPS-X register, followed by a stfto store the result into memory.

It is important to note that when a ***ldf*** or ***stf*** is being performed, it may be necessary for either the FPU or the Ecache to wait for one another. The solution adopted is to require each of these to remain in the same state until ϕ_2 has fallen; the required synchronization is accomplished by having the FPU and the Ecache stall MIPS-X using the Miss signal until they have both finished. The data should then be latched on the falling edge of ϕ_2.

8. Exceptions

Exceptions such as interrupts and page-faults are asynchronous events. The ***Interrupt*** line is maskable and latched by the processor during ϕ_1. The *Exception* line is not maskable and not latched. Figures 14 and 15 *give the timing* for *the Interrupt* and the *Exception lines* respectively. Exceptions and Interrupts can occur while the processor is being stalled by the Miss signal. When the *Miss* signal is released, execution will begin at the start of the exception handling routine. Section 3.1 explains the timing for these signals in more detail.

When MIPS-X recognizes an interrupt or exception, execution jumps to location 0 in system space. At this point, the exception routine should save the state of the processor and any coprocessors that are to be used before the interrupt returns. Details of this are explained in the Programmer's Manual **[2].**

Table 3 shows the action taken by MIPS-X for various instructions when an interrupt or exception occurs during the ALU, MEM and WB stages. If an interrupt occurs before the WB *stage* of a coprocessor instruction, *the* processor does not activate *the* ***WBEnable*** *line. The* coprocessor remains in a suspended state until it receives the next coprocessor cycle request,

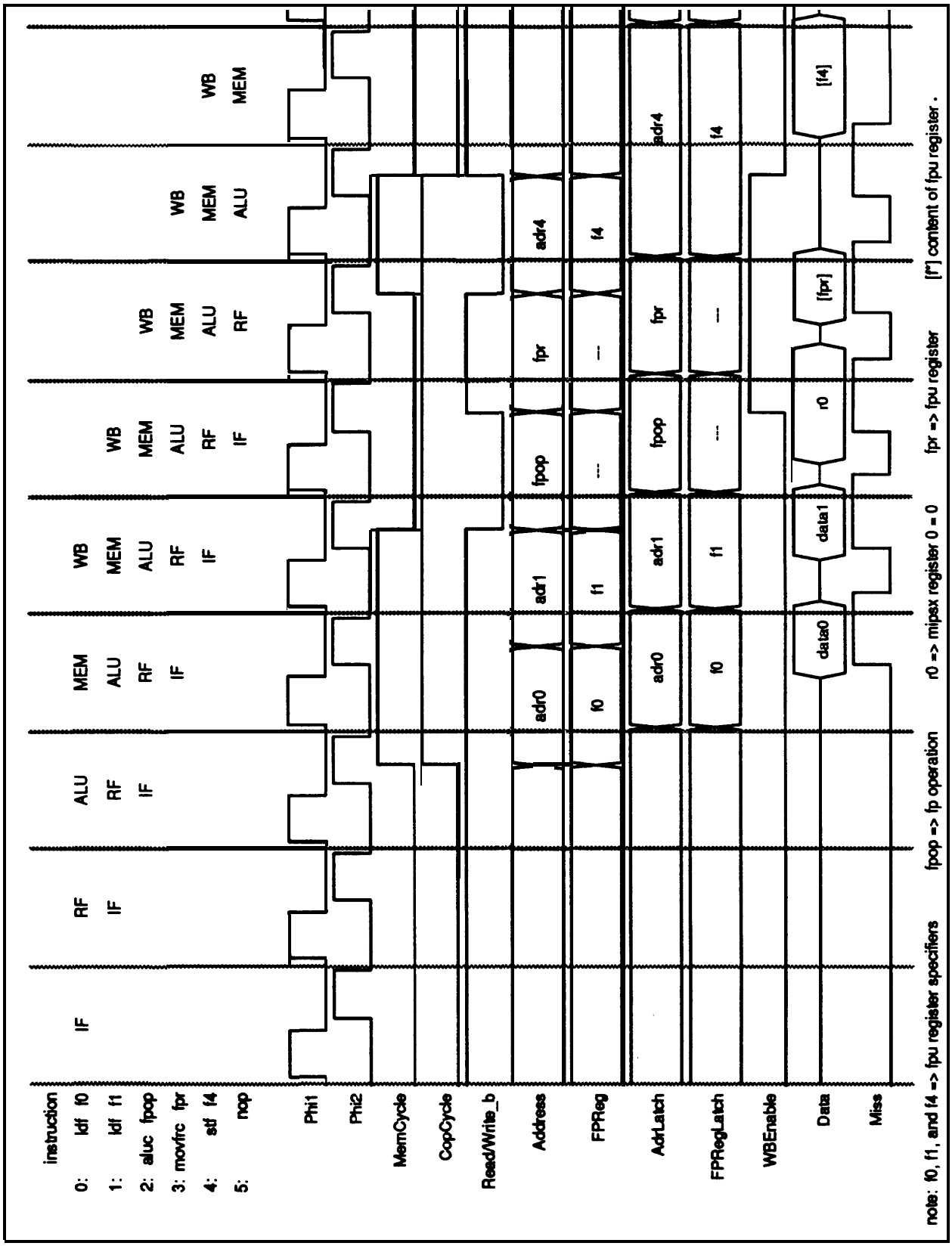


Figure 13: Timing diagram for floating-point operations

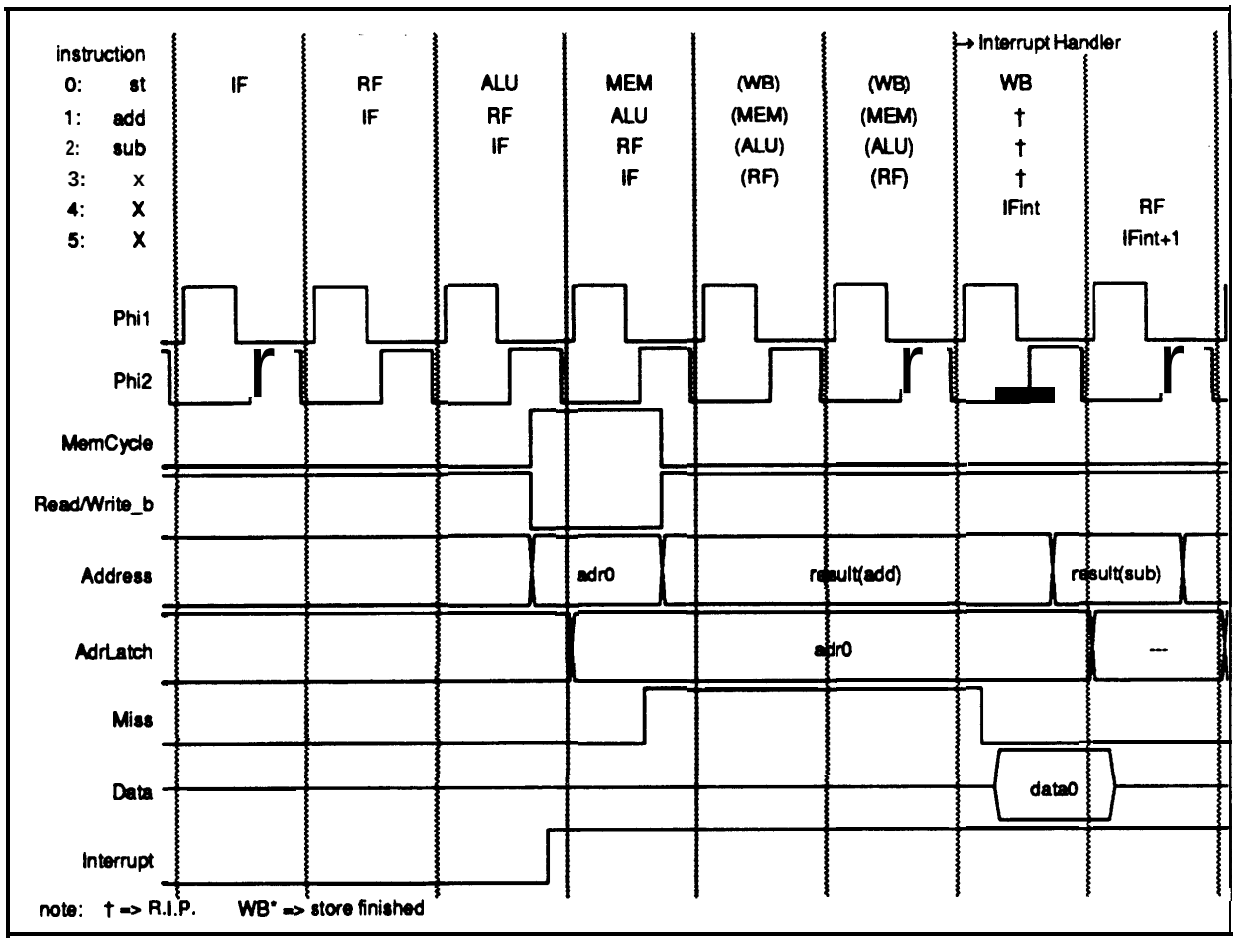


Figure 14: Interrupt timing

signaling a new instruction. In this case it starts off on' the new instruction, neglecting to do the writeback of the previous one. This effectively squashes the interrupted instruction. Interrupts which *arrive* after the ***WBEnable*** signal has been asserted have no effect on the coprocessor.

9. Reset

To reset the processor, this pin should be held high for at least 4 cycles and should only be released when ϕ_1 is active or during the rising edge of ϕ_1. Internally, Reset is expected to be a stable ϕ_2 signal.

During reset, the address **0x7fffff80**[6] is forced onto the address pins and the processor will begin executing at this location when *Reset* is released. This location should be the start of the

[6]This is the byte address that appears on the pins of the chip. Internally, the PC Unit is word addressed and it uses 0x7ffffe0 as its starting address. This is reflected in the simulators and the assemblers.

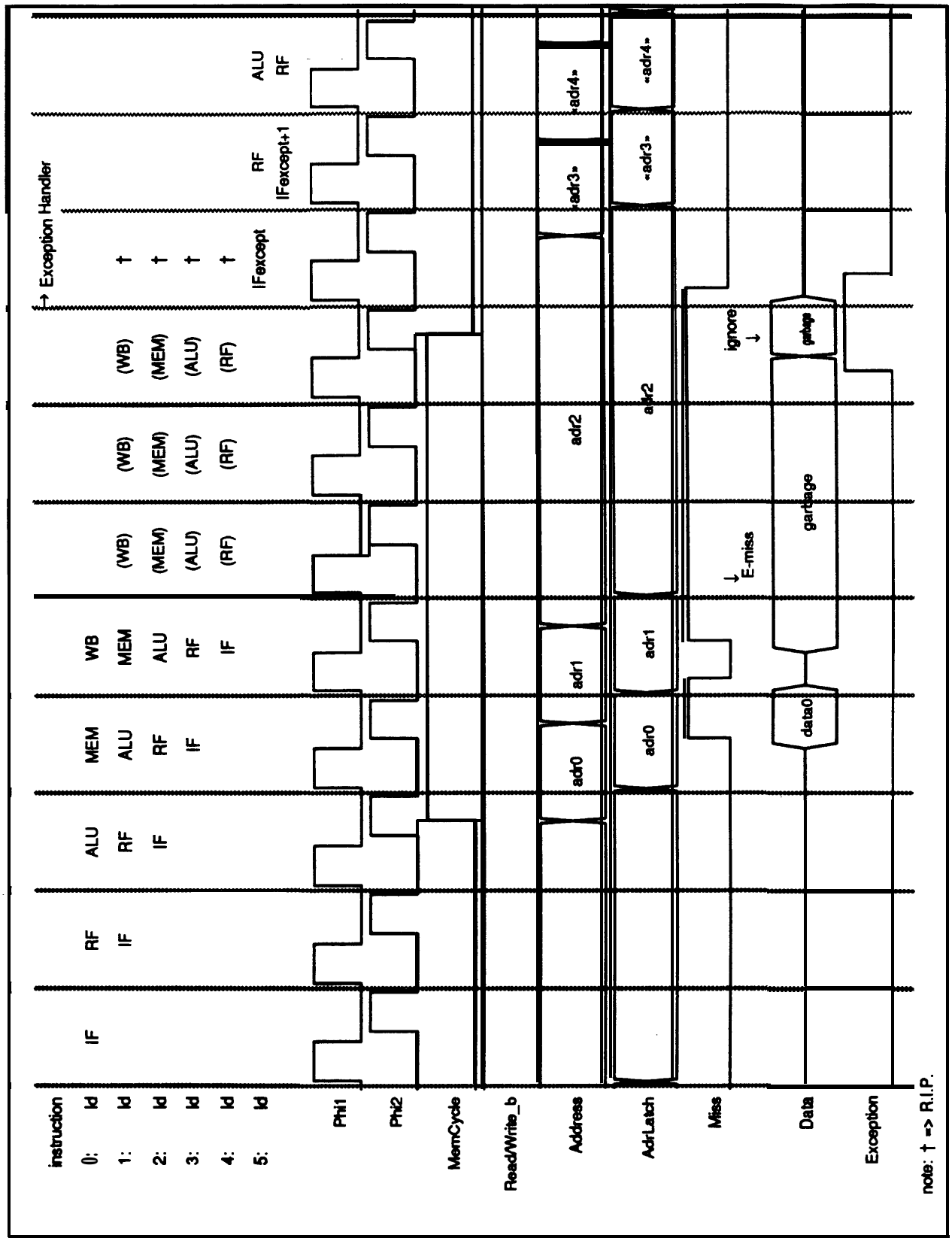


Figure 15: Exception timing during page fault

	ALU	MEM	WB
Compute instruction	Squashed	Squashed	Completes
Load instruction	Squashed (never leaves chip)	Squashed	Completes
Store instruction	Squashed (never leaves chip)	Completes next cycle	Completes
COP instruction	Squashed (never leaves chip)	WBEnable not generated	Completes if WBEnable seen

Table 3: MIPS-X actions for various instruction types when an exception occurs during their ALU, MEM and WB pipestages.

reset sequence.

10. Instruction Cache Testing

The instruction cache can be read and written to test whether the bits in the instruction cache are working. The sequence is:

1. Reset the processor by holding the *Reset* pin high for at least 4 cycles to clear the tags. Before releasing *Reset*, set *ICacheTest* high and do 4 more cycles. Then release *Reset* as described in Section 9.

2. *While ICacheTest* is high the PC Unit is forced to increment starting from the Reset vector. Since the Icache is empty, the processor will now do Icache miss cycles to fill the Icache with the data presented at the data pins. The data should be available at the pins while the *MemCycle* pin is high but can be left driven on the pins during this part of the test if only one pattern is to be loaded in all locations. With the PC Unit forced to be in increment mode, all locations in the cache will be filled with the data at the data pins after 1024 clock cycles. This number is not 512 cycles (the Icache can hold 512 instructions) because the processor is executing Icache miss cycles. You will see a pattern of 2 Icache miss cycles and 2 execute cycles on the address bus.

3. During ICacheTest mode the datapath of the processor is still interpreting the data at the data pins as instructions. This means that some care must be taken if the top two bits of the data (corresponding to the TY field of an instruction) are I *0* meaning that the instruction is a memory instruction. If the data can be interpreted as a store instruction, the data loaded into the Icache will be corrupted because the datapath will be driving the internal data bus. If the data can be interpreted as a load instruction, then the *MemCycle* pin cannot be used to determine when to put the next data pattern on the pins unless you know when it is for an Icache miss and when it is for the load instruction.

4. After *filling the* Icache, Reset *the processor while* keeping *ICacheTest high. This* will farce the PC Unit back to the Reset vector but it will not invalidate the contents of the tags. When *Reset* is released, the processor will sequence through the 512 Icache locations in 512 cycles. The data on the data pins will be data from the Icache.

5. If none of the patterns loaded into the instruction cache correspond to memory instructions, then the *PadMem* pin should always stay low because all instructions should be in the Icache and no data will be accessed. If *PadMem* does go high it is

an indication of data coming from the **Icache** being interpreted as a memory instruction or a failure in the tags causing an **Icache** miss cycle.

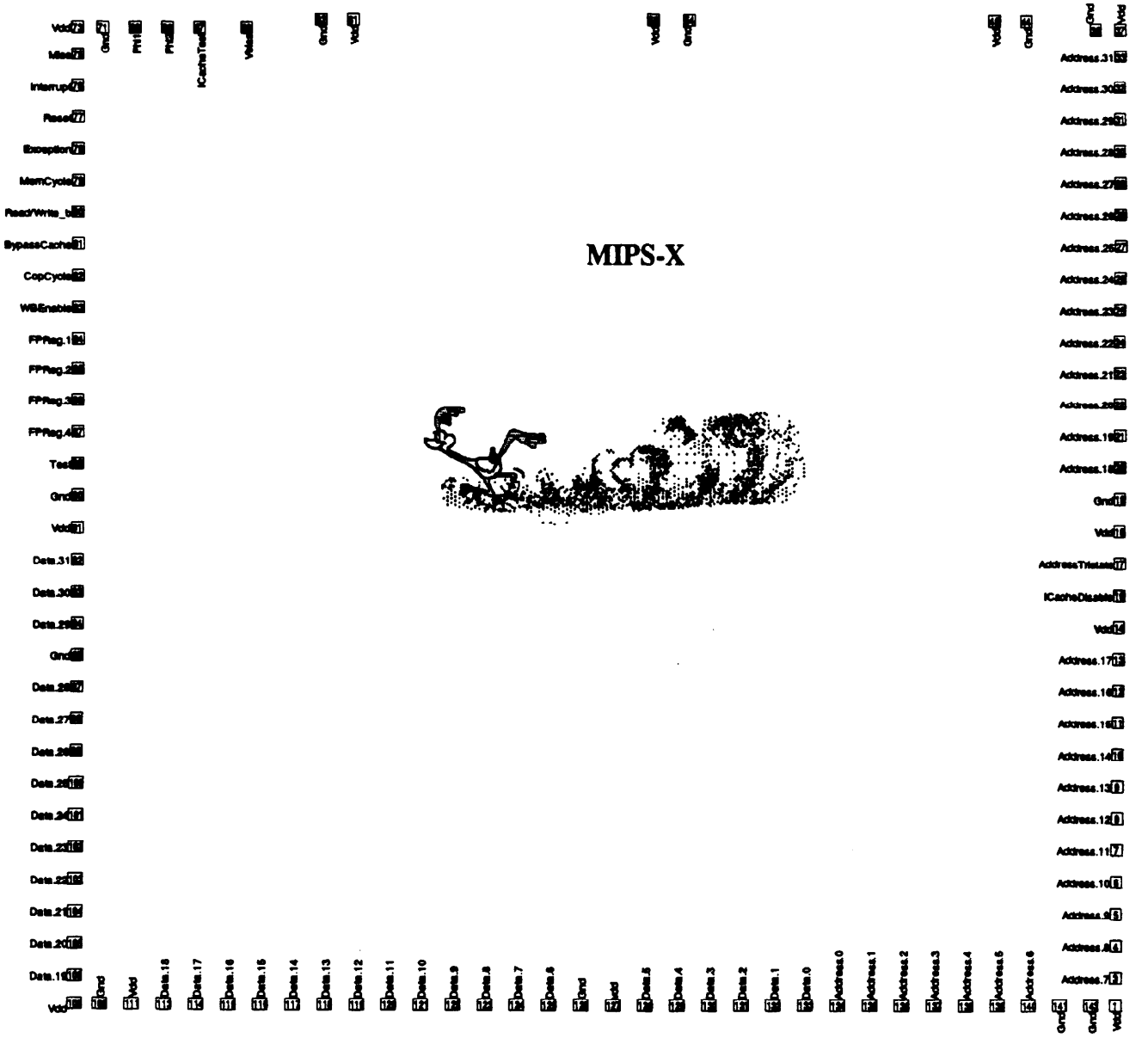
11. Data Path Testing

A useful feature for testing the data path is to observe the contents of the address bus. **Internally,** the address bus is the output of either the Result Bus or the PC Bus. During **Icache** miss cycles, the address bus will have the contents of the PC Bus. Otherwise, it will show the value of the Result Bus which usually has the output of the ALU or the shifter, depending on the instruction being executed.

Appendix I
MIPS-X Revision 1 and 2 Pin Numbers

The pin assignments for both Revision 1 and Revision 2 are the same except that the Revision 2 part does not have a *Test* pin. This pin is not connected (NC) on the Revision 2 part.

Two lists are given. The **first** one gives the correspondence between the pin numbers and names for the medium tester jumpers, the probe card, the layout and the functional simulator (Funsim). The names given are the names that are used in the layout and may not be exactly the same as the names used elsewhere in this document. The second list maps the pin number of the 144 pin PGA and the signal names to the medium tester jumpers and 64-pin connectors. A map of the pads is shown below. The numbers on the pins are the bonding pad numbers for the **144-pin** ceramic PGA from Kyoto Ceramic Co., Ltd. **(Kyocera).**



1.1. Pin Mapping for Probe Card and Funsim

This is the list of the pad names and their corresponding pin numbers for the tester, probe card and the layout. For the layout the numbering of the pads starts with Pin 1 in the top left comer of the chip and going counter-clockwise. For each pin, the name used in the layout and the name in the functional simulator are given.

*The **ClockTri*** pin is used to switch the state of the multiplexer that selects between using the tester to stimulate the clock pins and using an external clock for speed testing. This is only a connection between the tester and the probe card pins. A low value on this pin selects the tester as the clock source.

Tester Jumper #	Probe Card #	Layout Pin #	Pad Name	Funsim Name
120	B 56	1	Vdd - Pad	
56	A 56	2	PadMiss	PadMiss_v1
119	B 55	3	PadInterrupt_v1	PadInterrupt_v1
55	A 55	4	PadReset	PadReset
118	B 54	5	PadException_v2	PadException_v2
54	A 54	6	PadMem	PadMem
117	B 53	7	PadRead	PadReadWrite
53	A 53	8	PadBypass	PadBypass
116	B 52	9	PadCop	PadCop
52	A 52	10	PadWBen	PadWBen
115	B 51	11	PadFPReg.1	PadFPReg
51	A 51	12	PadFPReg.2	"
114	B 50	13	PadFPReg.3	"
50	A 50	14	PadFPReg.4	"
113	B 49	15	PadTest(NC in Rev.2) not in funsim	
49	A 49	16	Gnd - Pad	
112	B 48	17	Vdd - Pad	
48	A 48	18	PadDataBus.31	PadDataBus_s1
111	B 47	19	PadDataBus.30	"
47	A 47	20	PadDataBus.29	"
110	B 46	21	Gnd - Reg Array,Pad	
46	A 46	22	PadDataBus.28	"
109	B 45	23	PadDataBus.27	"
45	A 45	24	PadDataBus.26	"
108	B 44	25	PadDataBus.25	"
44	A 44	26	PadDataBus.24	"
107	B 43	27	PadDataBus.23	n
43	A 43	28	PadDataBus.22	"
106	B 42	29	PadDataBus.21	"
42	A 42	30	PadDataBus.20	"
105	B 41	31	PadDataBus.19	"
41	A 41	32	Vdd - Pad	
104	B 40	33	Gnd - Pad	
40	A 40	34	Vdd - Reg Array	
103	B 39	35	PadDataBus.18	
39	A 39	36	PadDataBus.17	"
102	B 38	37	PadDataBus.16	"
38	A 38	38	PadDataBus.15	"
101	B 37	39	PadDataBus.14	"
37	A 37	40	PadDataBus.13	"
100	B 36	41	PadDataBus.12	"
36	A 36	42	PadDataBus.11	"

Tester Jumper #	Probe Card #	Layout Pin #	Pad Name	Funsim Name
99	B 35	43	PadDataBus.10	"
35	A 35	44	PadDataBus.9	"
98	B 34	45	PadDataBus.8	"
34	A 34	46	PadDataBus.7	"
97	B 33	47	PadDataBus.6	"
33	A 33	48	Gnd - Pad	
96	B 32	49	Vdd - Pad	
32	A 32	50	PadDataBus.5	"
95	B 31	51	PadDataBus.4	"
31	A 31	52	PadDataBus.3	"
94	B 30	53	PadDataBus.2	"
30	A 30	54	PadDataBus.1	"
93	B 29	55	PadDataBus.0	"
29	A 29	56	PadAddressBus.0	AddressBus
92	B 28	57	PadAddressBus.1	"
28	A 28	58	PadAddressBus.2	"
91	B 27	59	PadAddressBus.3	"
27	A 27	60	PadAddressBus.4	"
90	B 26	61	PadAddressBus.5	"
26	A 26	62	PadAddressBus.6	"
89	B 25	63	Gnd - Tags,Pad	
25	A 25	64	Gnd - Pad	
88	B 24	65	Vdd - Pad	
24	A 24	66	PadAddressBus.7	"
87	B 23	67	PadAddressBus.8	"
23	A 23	68	PadAddressBus.9	"
86	B 22	69	PadAddressBus.10	"
22	A 22	70	PadAddressBus.11	"
85	B 21	71	PadAddressBus.12	"
21	A 21	72	PadAddressBus.13	"
84	B 20	73	PadAddressBus.14	"
20	A 20	74	PadAddressBus.15	"
83	B 19	75	PadAddressBus.16	"
19	A 19	76	PadAddressBus.17	"
82	B 18	77	Vdd - Tags	
18	A 18	78	PadICacheDisable	PadCacheDisable
81	B 17	79	PadAddressTristate	PadTriState
17	A 17	80	Vdd - Pad	
80	B 16	81	Gnd - Pad	
16	A 16	82	PadAddressBus.18	AddressBus
79	B 15	83	PadAddressBus.19	"
15	A 15	84	PadAddressBus.20	"
78	B 14	85	PadAddressBus.21	"
14	A 14	86	PadAddressBus.22	"
77	B 13	87	PadAddressBus.23	"
13	A 13	88	PadAddressBus.24	"
76	B 12	89	PadAddressBus.25	"
12	A 12	90	PadAddressBus.26	"
75	B 11	91	PadAddressBus.27	"
11	A 11	92	PadAddressBus.28	"
74	B 10	93	PadAddressBus.29	"
10	A 10	94	PadAddresaBus.30	"
9	A 9	95	PadAddressBus.31	"

Tester Jumper #	Probe Card #	Layout Pin #	Pad Name	Funsim Name
8	A 8	96	Vdd - Pad	
7	A 7	97	Gnd - Pad	
69	B 5	98	Gnd - ICache	
68	B 4	99	Vdd - ICache	
65	B 1	100	Gnd - ICache Sense Amp	
128	B 64	101	Vdd - ICache	
127	B 63	102	Vdd - ICache	
126	B 62	103	Gnd - ICache	
125	B 61	104	Vbias	
60	A 60	105	PadICacheTest_s1	PadTest
59	A 59	106	PadPhi2	
58	A 58	107	PadPhi1	
57	A 57	108	Gnd - Pad	
3	A 3		ClockTri	

1.2. Pin Map for 144 Pin PGA

The mapping in the MIPS-X column is between the signal names and the pin number on the 144 pin PGA. The numbers in the TESTER column is for the **144-pin** test adaptor that plugs into the medium tester and gives the tester jumper number and the number on the **64-pin** connectors.

MIPS-X			TESTER		
Bonding Pad No.	Pin No.	Signal Name	Jumper	64-pin conn.	Notes
1	D 3	Vdd	8	A 8	
3	B 1	Address.7	24	A 24	
4	D 2	Address.8	87	B 23	
5	E 3	Address.9	23	A 23	
6	C 1	Address.10	86	B 22	
7	E 2	Address.11	22	A 22	
8	D 1	Address.12	85	B 21	
9	F 3	Address.13	21	A 21	
10	F 2	Address.14	84	B 20	
11	E 1	Address.15	20	A 20	
12	G 2	Address.16	83	B 19	
13	G 3	Address.17	19	A 19	
14	F 1	Vdd	17	A 17	
16	H 2	ICacheDisable	18	A 18	
17	H 1	AddressTristate	81	B 17	
18	H 3	Vdd	40	A 40	
19	J 3	Gnd	7	A 7	
20	J 1	Address.18	16	A 16	
21	K 1	Address.19	79	B 15	
22	J 2	Address.20	15	A 15	
23	K 2	Address.21	78	B 14	
24	K 3	Address.22	14	A 14	
25	L 1	Address.23	77	B 13	
26	L 2	Address.24	13	A 13	
27	M 1	Address.25	76	B 12	
28	N 1	Address.26	12	A 12	
29	M 2	Address.27	75	B 11	
30	L 3	Address.28	11	A 11	
31	N 2	Address.29	74	B 10	
32	P 1	Address.30	10	A 10	
33	M 3	Address.31	9	A 9	
37	N 4	Vdd	41	A 41	
39	Q 2	Gnd	25	A 25	
44	Q 4	Gnd	33	A 33	
45	N 6	Vdd	68	B 4	
54	N 8	Gnd	49	A 49	
55	N 9	Vdd	82	B 18	
61	Q 11	Vdd	88	B 24	
62	P 11	Gnd	57	A 57	
66	N 11	Vbias	125	B 61	
67	P 13	ICacheTest	60	A 60	
68	Q 14	Phi2	59	A 59	(clock line)
69	N 12	Phil	58	A 58	(clock line)
71	P 14	Gnd	65	B 1	
73	M 13	Vdd	96	B 32	
75	P 15	Miss	56	A 56	
76	M 14	Interrupt	119	B 55	
77	L 13	Reset	55	A 55	
78	N 15	Exception	118	B 54	
79	L 14	MemCycle	54	A 54	
80	M 15	Read/Write-b	117	B 53	

Bonding Pad No.	Pin No.	Signal Name	I	Jumper	64-pin conn.	Notes
81	K 13	BypassCache		53	A 53	
82	K 14	CopCycle		116	B 52	
83	L 15	WBEnable		52	A 52	
84	J 14	FPReg.1		115	B 51	
85	J 13	FPReg.2		51	A 51	
86	K 15	FPReg.3		114	B 50	
87	J 15	FPReg.4		50	A 50	
88	H 14	Test		113	B 49	(NC on Rev.2)
90	H 13	Gnd		69	B 5	
91	G 13	Vdd		112	B 48	
92	G 15	Data.31		48	A 48	
93	F 15	Data.30		111	B 47	
94	G 14	Data.29		47	A 47	
95	F 14	Gnd		80	B 16	
97	E 15	Data.28		46	A 46	
98	E 14	Data.27		109	B 45	
99	D 15	Data.26		45	A 45	
100	c 15	Data.25		108	B 44	
101	D 14	Data.24		44	A 44	
102	E 13	Data.23		107	B 43	
103	c 14	Data.22		43	A 43	
104	B 15	Data.21		106	B 42	
105	D 13	Data.20		42	A 42	
106	c 13	Data.19		105	B 41	
108	A 15	Vdd		120	B 56	
109	c 12	Gnd		89	B 25	
111	A 14	Vdd		127	B 63	
113	c 11	Data.18		103	B 39	
114	A 13	Data.17		39	A 39	
115	B 11	Data.16		102	B 38	
116	A 12	Data.15		38	A 38	
117	c 10	Data.14		101	B 37	
118	B 10	Data.13		37	A 37	
119	A 11	Data.12		100	B 36	
120	B 9	Data.11		36	A 36	
121	c 9	Data.10		99	B 35	
122	A 10	Data.9		35	A 35	
123	A 9	Data.8		98	B 34	
124	B 8	Data.7		34	A 34	
. 125	A 8	Data.6		97	B 33	
126	C 8	Gnd		104	B 40	
127	c 7	Vdd		128	B 64	
128	A 7	Data.5		32	A 32	
129	A 6	Data.4		95	B 31	
130	B 7	Data.3		31	A 31	
131	B 6	Data.2		94	B 30	
132	C 6	Data.1		30	A 30	
133	A 5	Data.0		93	B 29	
134	B 5	Address.0		29	A 29	
135	A 4	Address.1		92	B 28	
136	A 3	Address.2		28	A 28	
137	B 4	Address.3		91	B 27	
138	c 5	Address.4		27	A 27	
139	B 3	Address.5		90	B 26	
140	A 2	Address.6		26	A 26	
141	c 4	Gnd		110	B 46	
143	B 2	Gnd		126	B 62	

Appendix II
Revision 1 and Revision 2 Differences

There are a few differences between the Revision 1 and Revision 2 MIPS-X parts. Most have to do with making the external interface easier to use. Some are internal changes that do not affect the external interface logic but are mentioned here for archival purposes.

1. On the Revision 1 parts, the instruction fetched from the Ecache during an **Icache** miss must be valid until ϕ_1 falling of the **"WB"** cycle of the **Icache** miss cycle. For example, the first instruction fetched back must be stable until ϕ_1 falling of **CM2.** For Revision 2, the timing is the same as a normal load sequence; the data must be valid **until ϕ_2 falls** during the **"MEM"** cycle of an **Icache** miss cycle. For example, the **first** instruction fetched back must be valid until CM1 ϕ_2 falls.

2. The address pins of the Revision 1 parts are set to zero on every ϕ_1, reflecting the **precharging** of the internal Result Bus. These pins are valid ϕ_2, stable ϕ_1 on the Revision 2 parts.

3. The *Miss* pin cannot be used to stall the processor on the Revision 1 parts. The processor must be stalled by stretching ϕ_2. The *Miss* pin stalls the processor correctly on the Revision 2 parts.

4. When an **Icache** miss occurs between the probe in the Ecache for a store instruction and the actual store, the store must be allowed to complete before the **Icache** miss sequence can begin. On the Revision 1 parts this interlock must be done by asserting Miss on the processor. For Revision 2 parts, the processor does the interlock and stalls itself until the store completes.

5. Because of the change to the store interlock described in the previous point, the timing of when data appears on the data pins has changed. For Revision 1 parts, data is asserted on the pins during MEM ϕ_2. The Revision 2 parts do different actions for stores and *movtoc* instructions. For store instructions, the data is driven off the chip when *Miss* falls during **WB** ϕ_1 and stays valid until ϕ_1 rises. Data for a *movtoc* instruction is driven off chip when *Miss* **falls** during MEM ϕ_1 and stays valid until Miss falls during WB ϕ_1. If Miss is not high when ϕ_1 goes high, then all actions will occur on the rising edge of ϕ_1.

6. There is only one difference in the **pinouts** of the two versions. The Revision 1 part has a *Test* pin that is used to enable observation of control lines in the data path. This feature was never used so the pin is not **connected** in the Revision 2 **part.**

The following changes are internal changes **only** and do not affect the external interface.

1. The testing multiplexers have been removed on the Revision 2 parts. This is why *the Test* pin has gone away.

2. The PLA in the PC Unit has been replaced by standard cell logic. This is to speed up the critical paths for branches and exceptions.

3. *The* branch path logic has been sped up.

4. The Tag section has been changed to speed up the valid-bit store access.

References

[1] Anant Agarwal, Paul Chow, Mark Horowitz, John Acken, Arturo Salz and John Hennessy.
On-Chip Instruction Caches for High-Performance Processors.
In 1987 Stanford Conference on Advanced Research in VLSI, pages l-24. Stanford, California, March, 1987.

[2] Paul Chow.
MIPS-X Instruction Set and Programmer's Manual.
Technical Report CSL-86-289, Stanford University, May, 1986.

[3] Mark Horowitz, Paul Chow, Don Stark, Richard Simoni, Arturo Salz, Steven Przybylski, John Hennessy, Glenn Gulak, Anant Agarwal and John Acken.
MIPS-X: A 20 MIPS Peak, 32-Bit Microprocessor with On-Chip Cache.
IEEE Journal of Solid-State Circuits SC-22(5):790-799, October, 1987.