MIPS-X: A 20-MIPS Peak, 32-bit Microprocessor with On-Chip Cache

MARK HOROWITZ, MEMBER, IEEE, PAUL CHOW, MEMBER, IEEE, DON STARK, STUDENT MEMBER, IEEE, RICHARD T. SIMONI, STUDENT MEMBER, IEEE, ARTURO SALZ, STEVEN PRZYBYLSKI, STUDENT MEMBER, IEEE, JOHN HENNESSY, MEMBER, IEEE, GLENN GULAK, MEMBER, IEEE, ANANT AGARWAL, STUDENT MEMBER, IEEE, AND JOHN M. ACKEN, MEMBER, IEEE

Abstract—**MIPS-X is a 32-bit RISC microprocessor implemented in a conservative 2-μm, two-level-metal, n-well CMOS technology. High performance is achieved by using a nonoverlapping two-phase 20-MHz clock and executing one instruction every cycle. To reduce its memory bandwidth requirements, MIPS-X includes a 2-kbyte on-chip instruction cache. This cache satisfies 90 percent of all instruction fetches, and reduces the memory bandwidth of the processor by a factor of 2.5. MIPS-X has a peak operating rate of 20 MIPS, and provides an effective throughput of 12 MIPS when the effects of the on-chip cache, external cache, and pipeline stalls are included. MIPS-X contains 150K devices in an 8×8.5-mm^2 die.**

To produce a high-speed computer system, MIPS-X uses a simple compute engine, a simple and fast clocking scheme, and a high-performance memory system. The simplicity of the basic processor allowed us to use a significant fraction of the design time and silicon area to integrate a part of the memory system on the processor. This paper provides an overview of MIPS-X, focusing on the techniques used to reduce the complexity of the processor and implement the on-chip instruction cache.

I. Introduction

THE MIPS-X project began in the Summer of 1984 with the goal of designing a second-generation RISC microprocessor that could be used as the processing nodes of a shared-memory multiprocessor. With the knowledge gained from early RISC designs [1]–[3] and the improved performance available from a 2-μm two-level-metal CMOS process we have designed a processor with a peak instruction rate of 20 MIPS. MIPS-X borrows from the original MIPS machine [1] the ideas of a simplified instruction set, pipelining, and a software code reorganizer to handle pipeline interlocks. However, to improve performance, MIPS-X uses a simpler instruction format, a deeper pipeline, an on-chip instruction cache, and a faster clock rate.

There are several areas that are important to consider when designing a high-speed processor, particularly one that is to be implemented in VLSI. These include the memory system design, the clocking methodology and the complexity of the resulting hardware. We feel that the most important factor is simplicity. For a high-speed processor, additional functionality should only be added when it significantly improves the overall performance of the machine. The design team has a certain amount of time and silicon area it can use to complete its task. Resources spent implementing a feature are resources that cannot be spent on other aspects of the design. In MIPS-X, the execution portion of the processor occupies a small fraction of the die area, allowing us to use the extra area to improve the performance of another critical element of the processor, the memory system.

As instruction rates increase, the bandwidth and latency of the memory system become important issues. This is evidenced by the greater use of on-chip caches and instruction prefetch queues to decrease the average time required to access instructions [4]–[10]. Crossing chip boundaries has become a limiting factor in high-speed processor systems; this makes it difficult to access instructions and data quickly if they have to be kept off-chip. MIPS-X uses both a large 2-kbyte on-chip instruction cache and an external interface optimized for high-speed cache access to provide the required memory bandwidth for the processor.

Increased performance also implies faster clock rates, and this makes the problem of clock distribution more difficult. Multiphase clocks exacerbate the situation because the time per phase is smaller and there are more phases to distribute. MIPS-X uses a simple two-phase clocking scheme, and locally generates additional clocks when necessary. Circuits using local clocks are often called self-timed because they derive the timing information from the delay of the circuit being controlled. The use of self-timed clocks makes the global clocking in MIPS-X simple, but does add some circuit complexity in the parts of the chip that require additional clocks.

The next section gives an overview of the MIPS-X architecture and the supporting memory structure. This is followed by a description of the pipeline in Section III. Sections IV and V present the hardware required to implement this machine. Section VI follows with a description of the design methodology used to keep the hardware

Manuscript received March 27, 1987; revised June 4, 1987. The MIPS-X research was supported by the Defense Advanced Project Research Agency under Contract MDA903-83-C-0335. P. Chow, S. Przybylski, and G. Gulak were supported in part by the Natural Sciences and Engineering Research Council of Canada.

The authors are with the Computer Systems Laboratory, Stanford University, Stanford, CA 94305.

IEEE Log Number 8716186.

relatively simple. The summary and current status of the project are given in Section VII.

II. Architectural Overview

The only instructions implemented in MIPS-X are those that contribute significantly to the performance of the machine. These instructions have been made to execute very quickly. The processor is a load–store machine; the only instructions that can access the word-addressed memory are explicit load and store instructions. All other instructions use the 32-word register file. There are three types of instructions: memory, branch, and compute. Memory instructions support a single addressing mode that adds an offset to the contents of a register to generate the effective address. Branch instructions contain an explicit comparison operation. This COMPARE AND BRANCH form was chosen to increase the speed of branches by removing the instructions that are normally needed to set the condition codes. Unlike some of the recently announced RISC machines [11], MIPS-X provides a full set of comparison operations for branches, rather than providing only simple (equality and sign) compares. Compute instructions are generally three-operand instructions with two sources and a destination. MIPS-X supports a wide variety of arithmetic, logical, and shift operations, including variable byte rotates to support character handling. A limited number of compute instructions include an immediate field, providing a simple way to generate and use short 17-bit constants.

The instruction format was optimized for simple decode. All 37 instructions are 32 bits and use a fixed format for the register specifiers. The four formats can be seen in Fig. 1. The comp func field in the compute instructions directly feeds control inputs in the execute unit making decoding very simple or nonexistent.

MIPS-X requires a low-latency and high-bandwidth connection to memory. With single-cycle execution and a 20-MHz clock, the peak bandwidth required for instructions and data is 40 Mwords/s (160 Mbytes/s). Besides the difficulty in designing a memory system to support this data rate, transferring this amount of data across the pins of the package is extremely difficult. To reduce the large instruction bandwidth requirements, MIPS-X has a 2-kbyte instruction cache (ICache) on the processor. This cache occupies about one-half of the interior die area, satisfies roughly 90 percent of all instruction references [12], and reduces the instruction bandwidth requirements across the pins by a factor of six.[1] Missed instruction references and data references go off-chip to a large 64K-word external cache. The on-chip cache effectively dual ports the memory system, allowing the processor to simultaneously fetch data from the external cache and instructions from the internal cache. The large external cache is

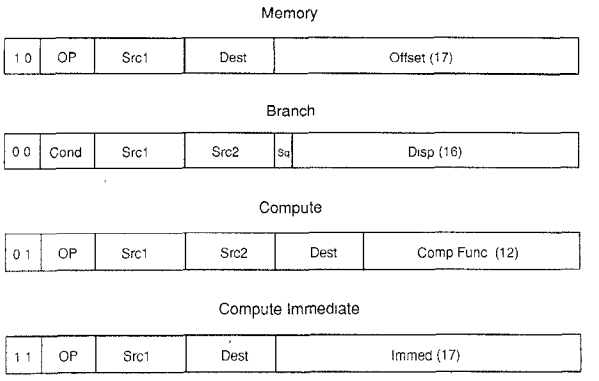


Fig. 1. MIPS-X instruction formats.

required to minimize the miss rate because accesses to the main memory take 15–20 processor cycles to fetch back four words. Low miss rates are also important to reduce bus contention in a shared-memory multiprocessor system.

The external interface is optimized for speed. It is designed to connect to a large cache memory, is fully synchronous, and can operate at a 50-ns cycle time. The interface is very simple; it presents an address by the beginning of a cycle and expects the data by the end of the same cycle. The general bus interface is placed on the other side of the external cache.

We realized early in the design that we would not be able to fit all the functionality needed for a high-speed computer onto a single die, so MIPS-X implements a simple, yet efficient coprocessor interface. This interface is made more difficult by the presence of the on-chip instruction cache which hides instructions from attached coprocessors. Instead of using valuable package pins to transfer the coprocessor instruction off the chip, MIPS-X uses the address and data bus for the coprocessor operations. During a coprocessor cycle an additional processor pin is asserted, indicating that the value on the address bus is a coprocessor instruction rather than a memory address. The coprocessors decode the instruction and determine their correct action. During these cycles the data bus can be used to transfer information between the coprocessor and MIPS-X. The inefficiency with this scheme is that all coprocessor-memory traffic must be transferred through the processor using extra instructions. We felt this would only be a significant problem for the floating-point processor. To improve the floating-point interface, two special memory instructions were added to MIPS-X that directly transfer data between one specific coprocessor and memory. With this minor addition we were able to provide a simple interface that supports high-performance coprocessors. One advantage of this interface is that coprocessor instructions look just like memory instructions and thus can be implemented easily.

MIPS-X provides separate system and user addresses. Programs running in user mode are prevented from accessing system addresses, while programs running in system mode can access either address space. The processor can enter system mode only by taking an interrupt or by

[1] 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.

executing a trap instruction. To support a dynamic paged virtual memory system, all instructions are restartable. The processor supports both maskable and nonmaskable interrupts. An interrupt causes the machine to flush the instructions in the execution pipeline, enter system mode, and jump to location zero. This simple support for exceptions provides the essential features needed to build an operating system for the processor.

III. Pipeline

Instructions in MIPS-X require five clock cycles to complete: instruction fetch (IF), register fetch (RF), execute (ALU), memory access (MEM), and write back of registers (WB). During IF, the instruction is fetched from the on-chip instruction cache and loaded into the instruction register. The RF cycle is used to drive the register specifiers from the instruction register to the register decoders and then to perform the actual register fetch. During ϕ_1 of the execute cycle either the ALU or the shifter evaluates, and during ϕ_2 this result is driven onto the result bus. For branch instructions, the ALU is used to evaluate the branch condition, and a separate adder in the program counter unit is used to compute the branch destination. This adder has the same timing as the ALU and evaluates on ϕ_1 of ALU. For memory instructions, the ALU is used to compute the effective address and during ϕ_2 this address is driven to the address pads. By having the ALU evaluate in a single phase, the address has enough time to be driven off the chip before the end of the ALU cycle. Thus the address is valid at the pins of the chip when the memory cycle begins. This predrive of the address gives the external cache memory a full cycle (MEM) to complete its access. The result of the instruction is written into the register file during ϕ_1 of WB.

The MIPS-X processor is pipelined so that a new instruction can be started every cycle. Starting the next instruction before the current instruction is completed gives rise to a number of pipeline dependencies as shown in Fig. 2. For example, the result of a branch instruction is not known until the end of the ALU cycle, too late to affect the IF of the next two instructions. Therefore, the two instructions following a branch will be fetched independent of the outcome of the branch; the branch delay is two cycles. The pipeline also has a delay slot associated with loads. Since the data from the load does not enter the chip until the end of the MEM cycle, it arrives too late to be used in the ALU of the next instruction. The instruction following a load cannot use the value just loaded. The processor does not contain pipeline interlocks in hardware so these pipeline interlocks are handled by a pass of the assembler called the reorganizer, a technique pioneered by the original MIPS processor [1]. The reorganizer is responsible for generating a code sequence that is free from pipeline dependencies. If the reorganizer cannot find a useful instruction to put into a delay slot, it fills the slot with a no-op instruction, effectively stalling the machine for a cycle at the cost of increased instruction bandwidth.

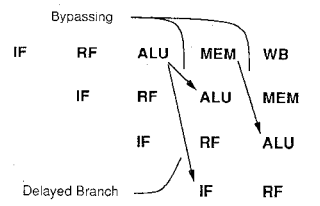


Fig. 2. Pipeline dependencies in MIPS-X.

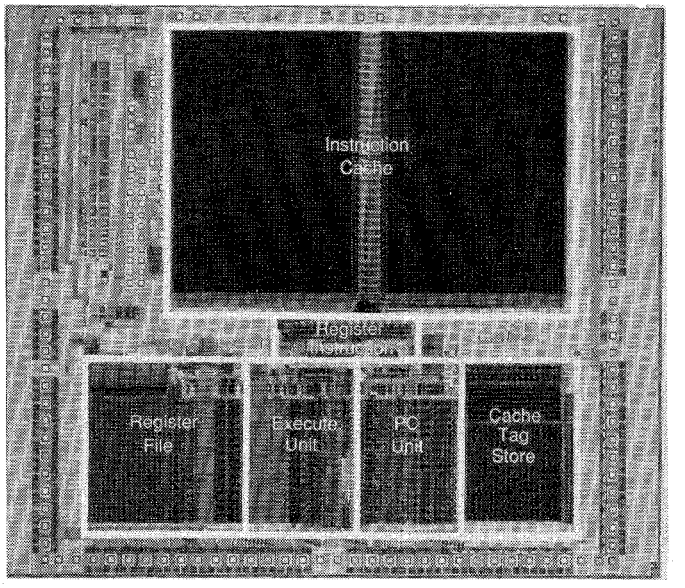


Fig. 3. Die photo of MIPS-X.

To help the software system use the two slots associated with a branch, MIPS-X can optionally squash (turn into no-ops) the instructions in the slots if the branch is not taken. This allows the reorganizer to predict that the branch will go and put the first two instructions of the branch destination after the branch. In this case the machine effectively starts executing the code at the branch destination right after the branch instruction. Only if the branch is not taken are these instructions turned into no-ops and the resulting cycles wasted.

To avoid having additional pipeline constraints, MIPS-X has two levels of internal forwarding or bypassing. The bypassing allows the result of one instruction to be used as input for the next instruction and is needed because the actual WRITE into the register file occurs late in the instruction, too late to be directly used in the next two instructions. The bypass logic slightly complicates the design of the register file, but greatly reduces the number of no-ops needed to eliminate interlocks.

IV. Hardware Resources

A microphotograph of the processor with the major functional blocks outlined is shown in Fig. 3. The on-chip instruction cache dominates the die, occupying the upper half of the chip. The data path of the processor runs under the cache, and can be divided into four major sections. The register file contains 32 general-purpose 32-bit registers,

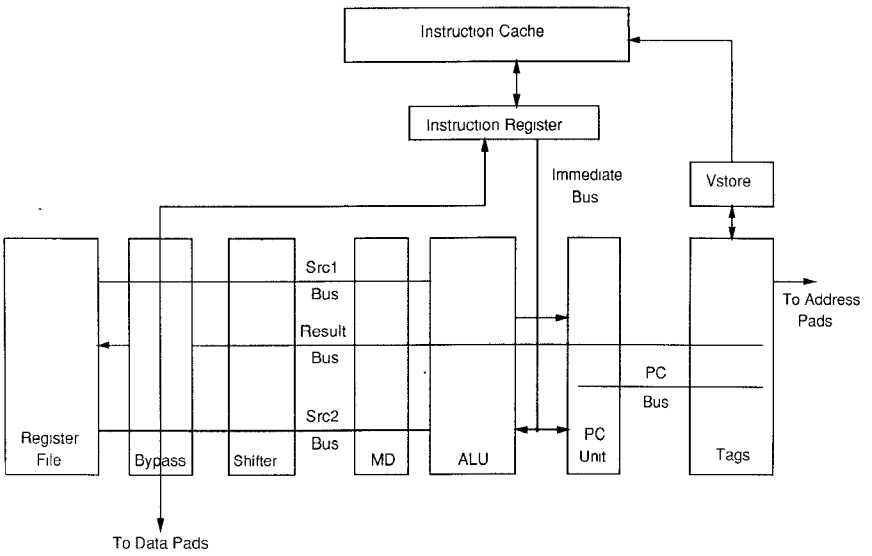


Fig. 4. MIPS-X hardware resources.

the pipeline bypass registers, and the registers associated with the external memory interface. The execute unit contains a 32-bit funnel shifter, a 32-bit ALU, registers to support single-bit multiplication and division (MD), and the processor status word. In the program counter unit (PC unit), there are two 32-bit adders, one used as an incrementer to calculate the next instruction address and the other used to compute the destinations of branches, and a chain of shift registers (PC chain) that is used to hold the addresses of the instructions currently in execution. These addresses are needed to restart the machine in the case of an interrupt. The tag section contains the tags and valid bits for the on-chip instruction cache. Located between the data path and the ICache is the instruction register, which contains a set of pipeline staging registers, and a small amount of instruction decode logic. The instruction register is also responsible for writing instructions into the cache during an internal cache miss.

Fig. 4 shows the hardware and the major buses. Data are read from the register file on the *Src*1 bus and *Src*2 bus. Data are written to the register file from the bypass block. The result bus carries values to the bypass block and to the tag section where it is multiplexed with the PC bus and used as an address for memory instructions.

A. Instruction Cache

Much of the design effort of MIPS-X was spent implementing the on-chip instruction cache. The goal was to design a simple cache that provided a high hit rate and a low cache-miss penalty. The on-chip cache is organized as 32 blocks of 16 32-bit words. Each block has a tag indicating the part of memory that is currently stored in it, and each word in a block has a valid bit indicating whether this word is currently stored in the cache. The use of valid bits allows the cache to have a large block size but use subblock replacement. The large block size was chosen to minimize the amount of storage required for tags, allowing a full 512-word instruction cache to be placed on the die. The small number of tags also allowed the tag memory array to be placed in the data path, reducing the amount of wiring needed for the cache. With the tag array in the data path, the large ICache above the data path becomes a 512×32-bit static RAM.

The cache system has a full cycle for its access, but needs to determine whether the instruction will hit in the cache in a single phase. The early hit detect is needed to be able to use the next cycle to fetch the missed instruction from the external cache as shown in Fig. 5. The root of the problem is that external memory accesses really take one and a half cycles; the processor must drive the address pads on ϕ_2 of the cycle before the memory access. To fetch the missed instruction by the end of the first cache-miss cycle, the processor must drive the instruction address off chip during ϕ_2 of the IF that misses, and thus we need the hit signal by the end of ϕ_1. Using the early hit detect, internal cache misses stall the machine for two cycles. The first cycle is used to fetch the missed instruction from the external cache, and the second cycle is used to write this value into the instruction cache. Since we assumed that the data from an external cache fetch are valid just before the end of the cycle, to reduce the miss delay to a single cycle we would need to extend the cycle time to provide sufficient time for a cache write to complete after the data become valid. Instead, MIPS-X uses the second cache-miss cycle to fetch from the external cache the next instruction that will be executed. Therefore, ICache misses have a penalty of two cycles, but fetch back two words. This fetch of two words halves the miss rate of the cache and provides roughly the same system performance as a cache with a single-cycle miss penalty, but accomplishes this performance without influencing the cycle time of the processor.

The tags are stored in a content-addressable memory using a standard ten-transistor CAM cell so they can be quickly compared against the current instruction address. Fig. 6 shows the tag array. The 32 tags are placed in the

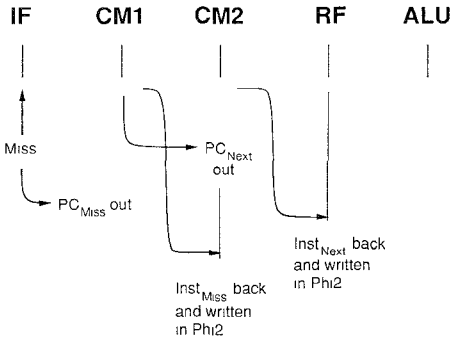


Fig. 5. MIPS-X instruction cache-miss timing.

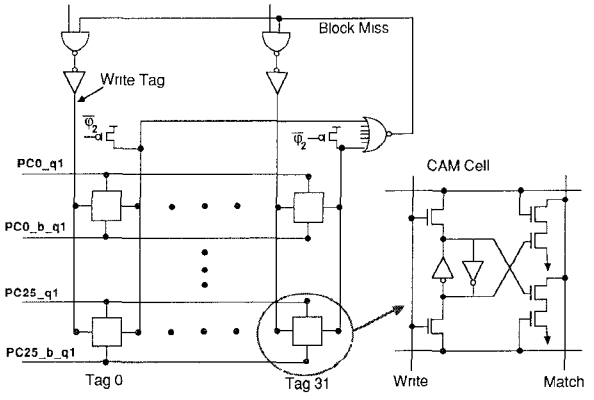


Fig. 6. Tag array showing tag cell and self-timed WRITE.

data path and match its pitch. Located above each tag are the 16 valid bits associated with that tag. The valid bit cells are the same width as the tag cells which means that the valid bit store (Vstore) fits directly on top of the tags.

Logically, the 32 tags are broken into four different sets of eight entries each. Low-order bits of the instruction address select a set and the associative compare is used to find the correct entry in the set. The most significant 24 bits of the instruction address are compared against the tag entries. The least significant 2 bits are the byte selector and are always zero for instructions; the next four bits select the correct word in each cache block, and the next two bits select the correct set.

Hit detection requires first comparing the current instruction address against the values stored in the CAM array, and then fetching the correct valid bit for the block that matches. To generate the hit information in one phase, the tag compare and valid bit fetch are performed simultaneously. The Vstore is logically organized as 64 words of eight bits. During the tag compare the low-order bits of the instruction address are used to index into the Vstore to fetch the eight possible valid bits, a bit for each tag that could match. Next these output lines are ANDed with the output of the tag comparison, and then ORed together to generate the cache hit signal. Since the tag compare and the Vstore access both require roughly 15 ns, it is easy to generate the hit signal in a single phase.

There are two types of internal cache misses: block miss and word miss, depending on whether the block for the

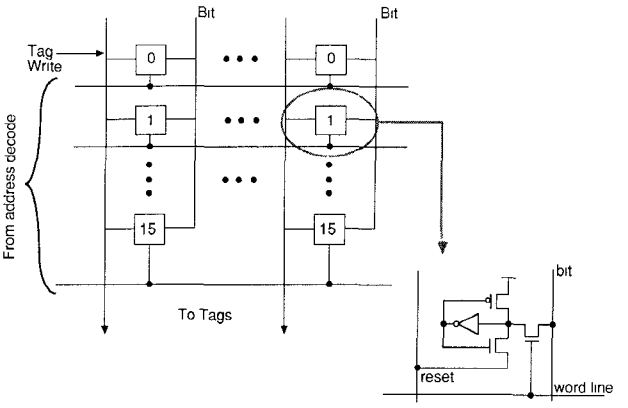


Fig. 7. Valid store circuit.

desired instruction is already in the cache. To make the cache-miss sequencer simpler, it only handles word misses. When a block miss occurs, a tag is written with the new instruction address and the valid bits for that tag are flushed in the same phase that the block miss was detected (Fig. 6). The tag write allocates a block for the instruction, and makes block misses look like normal word misses. To generate the write signal, we make use of the monotonic nature of the tag comparison logic. The match output of each CAM word is precharged on ϕ_2 and falls during ϕ_1 if the instruction address does not match. The outputs of all the match lines are NORed together forming the block-miss signal. This signal starts low at the beginning of ϕ_1 and rises only if a block miss occurs. It is used to drive the write line of the selected tag high, writing the current value of the program counter into the tag. Fig. 7 shows how the tag write line also serves as a virtual ground for the valid bits associated with that tag. When the write line is pulled high it forces all the cells to reset, clearing the valid bits for that tag.

MIPS-X uses a simple ring counter algorithm for selecting the tag to be replaced during a block miss. The ring counter is located above the Vstore, and is incremented after each block miss. The fetch of two instructions during a cache miss means that the ring counter must also increment when there is a block hit and word miss, and the ring counter points to the block where the hit occurred. This prevents a block miss during the fetch of the second instruction from clobbering a block that only had a word miss during the fetch of the first instruction.

The data portion of the instruction cache uses a fairly conventional static RAM design that has been optimized for synchronous operation. During ϕ_1 of IF, the bit lines and sense circuit of the RAM are precharged, and the low-order six bits of the instruction address are driven to the RAM and decoded. These six bits form the row address. Near the end of this phase, the tag comparison information is available and is sent to the RAM. This information is used for the column select. During ϕ_2 of IF the selected word line is driven and the outputs of the sense amplifiers are latched into the instruction register. Because of the short bit lines and relatively large cell transistors, MIPS-X uses a simple unclocked sense circuit

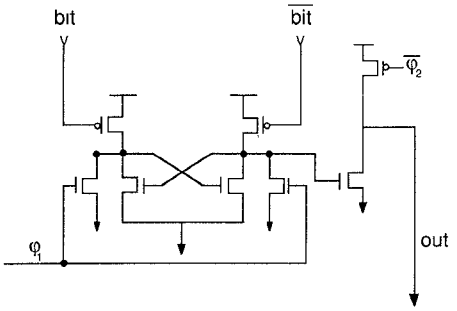


Fig. 8. RAM sense amplifier.

(see Fig. 8). This circuit is relatively slow since one bit line must fall below a p-FET threshold before it begins to sense, but has the advantages of not requiring a sense clock and not dissipating any static power. The measured access time of the RAM is about 18 ns, well within the single-phase access requirements.

B. Register File

The MIPS-X architecture requires a dual-READ single-WRITE register file, with support for double bypassing. The register file is time multiplexed, with WRITE's occurring on ϕ_1 and READ's on ϕ_2. To reduce the access time, three sets of decoders are placed above the register array, one for each access port. The inputs to the decoders are driven on the phase before their output is used so the decode time is not on the critical path for accessing the registers.

The initial design of the register cell used dual-differential buses, but this was dropped because the short bit lines made sense amplifiers unnecessary. Instead we used a CMOS version of the six-transistor RAM cell with split word lines described by Sherburne *et al.* [2]. Fig. 9 shows the CMOS cell. Time multiplexing the register array did pose a minor problem, since both bit lines must start at 5 V for a READ. The self-timed circuit shown in Fig. 10 was used to solve this problem. This circuit detects when a WRITE has completed, turns off the WRITE and then restores both bit lines high. A row of dummy cells was placed above the register array; these cells are hardwired to always contain a zero. Thus, after a READ the dummy *bit* line is always low and the $\overline{bit}$ line is high. The write drive for the dummy row input is tied high, so it always tries to write a ONE into the cells. Transistor M_{sense} detects when the bit lines have crossed by enough to write the register. This transistor discharges the precharged node $\overline{Done}$, causing *Done* to rise, and forcing the write drivers to recover the bit lines for the following READ. Transistor M_{once} is needed to prevent the circuit from oscillating. If it is deleted, then the recovery of the $\overline{bit}$ line will cause $\overline{Done}$ to rise, and the write will restart. The write and recovery is quite fast, requiring less than 20 ns to complete.

To remove many potential pipeline interlocks, the register file is double bypassed. This requires adding bus drivers to two latches in the data path, and adding four comparators in the control as shown in Fig. 11. The comparators check the destination of the previous two instructions against the two register sources of the current instruction to see if bypassing is required. If a match occurs, the correct latch output is driven onto the source bus instead of the data from the register file. The comparators are built around the set of latches needed to delay the destination specifier, which is driven into the register file on ϕ_1 of RF but not used until ϕ_1 of WB. The comparators use a precharged gate of n transistors and a predis-

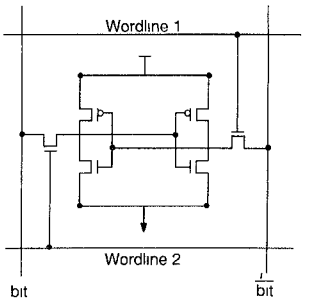


Fig. 9. CMOS dual-port register cell.

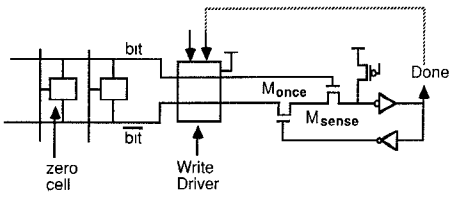


Fig. 10. Schematic of the self-timed bit-line WRITE circuit.

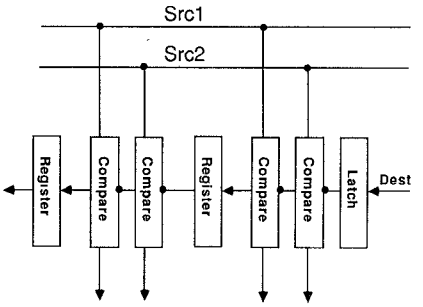


Fig. 11. Register bypass logic showing the comparators.

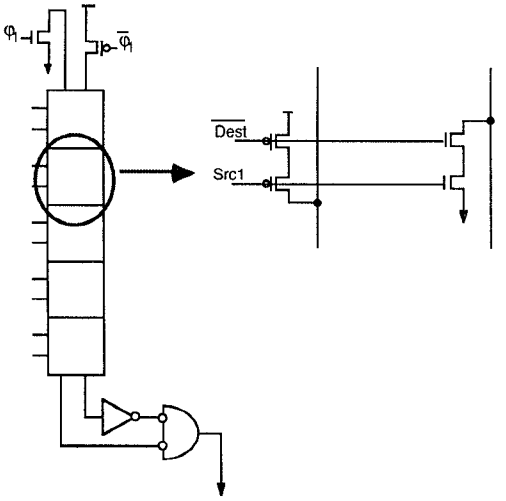


Fig. 12. Schematic of comparator circuit.

charged gate of p transistors to avoid requiring both the true and complement versions of the register specifiers. Fig. 12 shows the comparator circuit.

V. Control

The simple instruction format and the use of a lock-step pipeline make the control for the processor relatively simple. Most of this control is implemented in simple decoders and PLA's located above the part of the data path being controlled. To keep the complexity of this logic low, each designer was responsible for the design of a section of the data path and its control. This organization provided incentive to arrange the overall design to minimize the amount of random logic needed.

A. Global Control

Care was taken to keep the global control for the machine extremely simple. There are only three types of pipeline interruptions possible—exceptions, external cache misses, and internal cache misses—and of these only the first one requires the pipeline to be flushed. The cache misses only cause the processor to stall until the required data become available. In the case of an exception (either an interrupt, external fault, or internal fault) MIPS-X holds the instruction addresses of the last four instructions in the PC chain, squashes the instructions in the pipeline by preventing them from writing their results back into the register file, and jumps to system address 0. No attempt is made to complete instructions in the pipeline that occur before the instruction that caused the exception. The uniform effect of an exception makes the controller quite simple. The exception signal directly no-ops the instructions in the MEM and ALU phase of the pipe, and also sets up a small finite state machine (FSM) that causes the instructions in IF and RF to be converted to no-ops. The machine can be restarted by simply jumping to the addresses of the instructions stored in the PC chain.

The FSM used for exceptions is also used to implement the conditional evaluation of the two instructions that follow a branch. If the branch does not go, then during its ALU cycle the input to this FSM is set converting the instructions in RF and IF (the two slots of the branch) into no-ops. The squashing of branch slots does not need any additional logic; the same hardware is used to implement exceptions.

An FSM for handling internal cache misses is the only other global control that MIPS-X requires. During an ICache miss this controller sequences the machine through the two cache-miss states before resuming the execution pipe. The two FSM's are shown in Figs. 13 and 14. Collectively, these two controllers use less than 0.2 percent of the total chip area and are built with standard cells.

B. Stalling the Processor

The pipeline is stalled by using a set of qualified ϕ_1 clocks, ψ_1 and ψ_{1PC}. These clocks are used to latch all the state information in MIPS-X, and the machine is stalled by simply preventing these clocks from rising. This scheme is similar to one used to stall a floating-point unit in the case of an unusually long carry [13]. If ψ_1 does not rise, the machine throws away the results computed on ϕ_1 and during the next ϕ_2 repeats the ϕ_2 operation of the previous cycle.

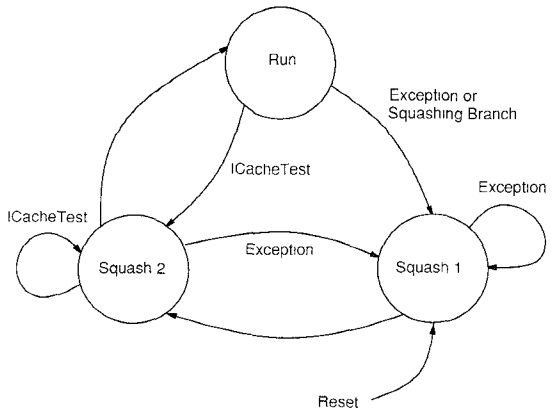


Fig. 13. Squash FSM.

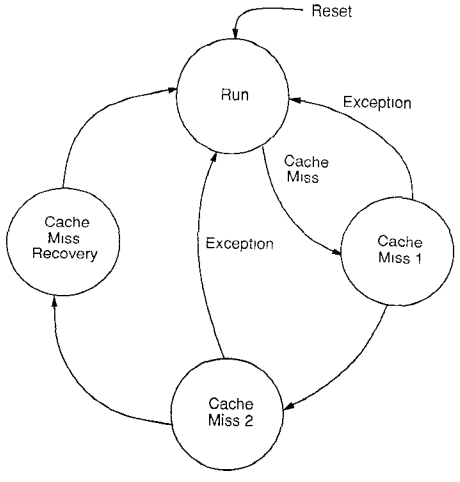


Fig. 14. Cache-miss FSM.

The ψ_1 clock is ϕ_1 qualified by *external cache miss* and *internal cache miss*. The ψ_{1PC} clock is ϕ_1 qualified by only *external cache miss*. Two clocks are needed since part of the processor must be clocked during internal cache misses, in particular the cache-miss FSM. This set of logic uses ψ_{1PC} while the rest of the chip uses ψ_1.

The ψ_1 clocks can only be used as an input to a latch; the clocking of functional units is always done on the true clocks ϕ_1 and ϕ_2. This allows the ψ_1 clocks to be slightly shorter than ϕ_1, or said a different way, it means that the external cache-miss signal can arrive a little late. As long as the external cache-miss signal monotonically falls, it can actually arrive at the processor after the end of the MEM cycle, during ϕ_1 of WB. The external miss signal can arrive up to 10 ns late and still provide a valid ψ_1 clock. This gives the external cache about 10 ns to generate the miss signal after the data fetch, and prevents the cache tag comparison from being on the critical path for memory accesses.

VI. Design Methodology and Testing

The basic MIPS-X architecture and pipeline structure were developed during the first six to nine months of the project. During this time an instruction level simulator for the machine was developed and used to evaluate the effects of different architectural features. We also investigated many organizations for the internal cache memory before settling on the one described in this paper. The different architectural trade-offs are described in more detail in [14]. In parallel with the architectural definition, we began investigating different implementations, and about a year after the project started we had a paper design of the hardware needed to implement the processor. The paper design included the layout of a number of the large structures MIPS-X would need. The layouts were done to get a better feel of the density and performance of the CMOS technology that we would use. At this point the chip was partitioned into several major functional units: the register file, the execute unit, the PC unit, the tag store, the instruction cache, the instruction register, and the external interface. Each of these sections, including its control, was designed by a single person. There was a total of six people. We used a tall thin design style; each designer was responsible for the design of a section, from writing the functional description for simulation, to generating the layout. Interface signals between the sections were fixed at the start of this design phase and then negotiated between the various parties if changes were required.

The first step of the detailed design was to write a functional description for the machine. We chose to write a custom simulator in Modula-2 because we lacked a good functional simulator at the time. The individual sections were written first, debugged, and then put together when everyone was satisfied that their sections were working correctly. This functional simulator became the *de facto* definition of the machine and was used quite extensively in the verification of the layout and testing of the silicon.

Once the functional definition was complete, the layout effort was started in earnest, using the Magic [15] layout system. This system has incremental design-rule checking and hierarchical extraction. Each section was extracted and then simulated using RSIM [16], a switch-level simulator. The functional simulator was modified so that it could be used to drive the RSIM simulations, making the verification of the circuits much easier. The functional simulator would provide the input vectors to a switch-level model of a subsection of the chip, and check the outputs of the switch-level simulator. This proved to be a powerful tool because it made it very easy to find differences between the functional and circuit representations. Using this method each designer was able to verify his section against the functional simulator before releasing it to the full chip simulation. On a MicroVax II, simulation of the entire chip (without the cache array) took about one minute per clock cycle and only found a few errors. Most were subtle timing errors that the functional simulator could not catch. Five machines were kept busy for about two weeks to do the final simulations before tape-out.

To simplify the testing of MIPS-X, we included the ability to separately test the processor and the large instruction RAM. By asserting an external pin the cache can be disabled, allowing the processor to run even if the cache is not functional. This feature simply forces a cache miss on every cycle so that the cache is never accessed. Asserting another pin puts the processor in the cache test mode. In this mode, the PC unit generates sequential addresses while the data bus is connected to the cache so that the cache can be directly read and written. These testing pins were used quite extensively during testing.

No special hardware was needed to test the data path of the processor. Whenever the processor is not handling an internal cache miss, the address pins are driven to be the value of the result bus. This makes it easy to observe the result of compute instructions and check the functionality of the execute unit and the register file.

Some hardware was added to make testing of the datapath control easier. By placing a small amount of logic under the data bus we could directly observe groups of control pins on the data pads by asserting a test pin. This can be done in the middle of any clock phase to allow direct observation of the internal control state of the machine. So far, this feature has not been used because no problems have been found in the control.

VII. Summary and Status

The first version of the MIPS-X processor was sent out for fabrication in May of 1986, and silicon was returned at the beginning of October. The functional simulator was used to generate test vectors for a low-speed functional tester developed t Stanford University. Simple speed testing was done by loading a small program into the instruction cache with the tester and then turning up the clock speed while observing the address bus. These parts are fully functional, run at 16 MHz, and dissipate less than 1 W at nominal operating conditions. Although the parts did not meet our ultimate cycle-time specification, they did run as fast as the simulation predicted. These die have been probed using low-capacitance probes and the waveforms match the simulation results quite well. The slower speed is caused by a slow path involving branches that has been fixed on the next revision of the part. This revision also includes a number of other small changes to improve other slow paths, and to make the external interface easier to use. We fully expect this version of the part to meet the 50-ns cycle time. We are also working on a simple shrink of the part to a 1.6-μm CMOS technology. This will yield a die of under 6.5 mm on a side, with a cycle time of over 25 MHz.

MIPS-X demonstrates the power of keeping VLSI processors simple, obtaining an effective throughput of over 10 MIPS while using a conservative technology and a relatively small die size. The key was to use the silicon

where it made the most difference: in the memory system design.

ACKNOWLEDGMENT

The authors gratefully acknowledge J. Gasbarro and E. McCreight of Xerox for providing fabrication support. C. Y. Chu, S. McFarling, S. Richardson, P. Steenkiste, and S. Tjiang deserve special thanks for their contributions, and thanks to M. Rowland for doing some of the drawings.

REFERENCES

[1] C. Rowen, S. A. Przybylski, N. P. Jouppi, T. R. Gross, J. D. Shott, and J. L. Hennessy, "A pipelined 32b NMOS microprocessor," in *ISSCC Dig. Tech. Papers*, Feb. 1984, pp. 180–181.

[2] R. W. Sherburne Jr., M. G. H. Katevenis, D. A. Patterson, and C. H. Sequin, "A 32b NMOS microprocessor with a large register file," *IEEE J. Solid-State Circuits*, vol. SC-19, pp. 682–689, Oct. 1984.

[3] G. Radin, "The 801 minicomputer," in *Proc. SIGARCH/SIGPLAN Symp. Architectural Support for Programming Languages and Operating Systems, ACM* (Palo Alto, CA), Mar. 1982, pp. 39–47.

[4] M. Hill *et al.*, "Design decisions in SPUR," *Computer*, vol. 19, no. 10, pp. 8–22, Oct. 1986.

[5] D. MacGregor, D. Mothersole, and B. Moyer, "The Motorola MC68020," *IEEE Micro*, vol. 4, no. 4, pp. 101–118, Aug. 1984.

[6] K. A. El-Ayat and R. K. Agarwal, "The Intel 80386—Architecture and implementation," *IEEE Micro*, vol. 5, no. 6, pp. 4–22, Dec. 1985.

[7] D. Phillips, "The Z80000 microprocessor," *IEEE Micro*, vol. 5, no. 6, pp. 23–36, Dec. 1985.

[8] A. Berenbaum, B. W. Colbry, D. R. Ditzel, R. D. Freeman, H. R. McLellan, M. Shoji, and K. J. O'Connor, "A pipelined 32b microprocessor with 13Kb of cache memory," in *ISSCC Dig. Tech. Papers*, Feb. 1987, pp. 34–35, 331.

[9] P. W. Bosshart *et al.*, "A 553K-transistor LISP processor chip," in *ISSCC Dig. Tech. Papers*, Feb. 1987, pp. 202–203, 402.

[10] D. Archer *et al.*, "A 32b CMOS microprocessor with on-chip instruction and data caching and memory management," in *ISSCC Dig. Tech. Papers*, Feb. 1987, pp. 32–33, 329.

[11] J. Moussouris *et al.*, "A CMOS RISC processor with integrated system functions," in *Compcon Dig. Papers*, Mar. 1986, pp. 126–131.

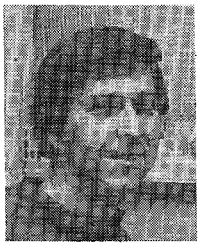
[12] A. Agarwal, P. Chow, M. Horowitz, J. Acken, A. Salz, and J. Hennessy, "On-chip instruction caches for high-performance processors," in *Proc. 1987 Stanford Conf. Very Large Scale Integration* (Stanford, CA), Mar. 1987, pp. 1–24.

[13] G. Wolrich, E. McLellan, L. Harada, J. Montanaro, and R. Yodlowski, "A high performance floating point coprocessor," *IEEE J. Solid-State Circuits*, vol. SC-19, pp. 690–696, Oct. 1984.

[14] P. Chow and M. Horowitz, "Architectural tradeoffs in the design of MIPS-X," in *14th Annual Int. Symp. Computer Architecture Conf. Proc.* (Pittsburg, PA), June 1987, pp. 300–308.

[15] J. K. Ousterhout, G. T. Hamachi, R. N. Mayo, W. S. Scott, and G. S. Taylor, "The Magic VLSI layout system," *IEEE Design Test Comput.* vol. 2, no. 1, pp. 19–30, Feb. 1985.

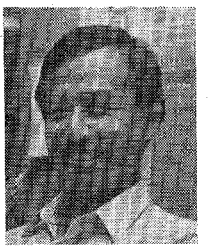
[16] C. Terman, "Simulation tools for digital LSI design," Massachusetts Inst. Technol., Cambridge, Tech. Rep. MIT/LCS/TR-304, Sept. 1983.



Mark Horowitz (S'77–M'78) received the B.S. and M.S. degrees in electrical engineering from the Massachusetts Institute of Technology, Cambridge, in 1978, and the Ph.D. degree in electrical engineering from Stanford University, Stanford, CA, in 1983.

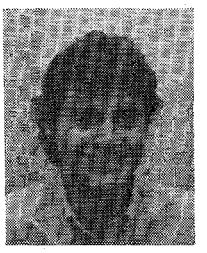
Since September of 1984, he has been an Assistant Professor in the Department of Electrical Engineering at Stanford University. His current research interests are in the area of high-performance digital system design, especially integrated circuit design, and computer tools to aid in the design process. His research projects have ranged from high-speed ECL RAM design to designing a 20-MIPS CMOS processor to improved models for switch-level simulation.

In 1984 Dr. Horowitz was given a Presidential Young Investigator Award, and an IBM New Faculty Development Award.



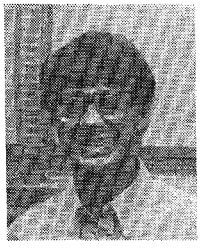
Paul Chow (S'79–M'83) is from Peterborough, Ont., Canada. He received the B.A.Sc. degree with honours in engineering science, and the M.A.Sc. and Ph.D. degrees in electrical engineering from the University of Toronto, Toronto, Ont., in 1977, 1979, and 1984, respectively.

In 1984 he joined the Computer Systems Laboratory at Stanford University, Stanford, CA, as a Postdoctoral Fellow and became a Research Associate in April of 1987. He has been working on the MIPS-X project. His current research interests include computer architecture, the design of large VLSI systems and the CAD tools involved.



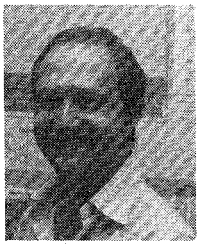
Don Stark received the B.S. degree in electrical engineering from the Massachusetts Institute of Technology, Cambridge, in 1985 and the M.S. degree in electrical engineering from Stanford University, Stanford, CA, in 1987. Currently a graduate student at Stanford, his interests include computer architecture, VLSI, and CAD tools.

Mr. Stark is a member of Tau Beta Pi and Eta Kappa Nu.



Richard T. Simoni received the B.S. degree in electrical engineering from Rice University, Houston, TX, in 1984 and the M.S. degree in electrical engineering from Stanford University, Stanford, CA, in 1985. He is now a graduate student in the Department of Electrical Engineering at Stanford University, where his research interests include multiprocessor memory hierarchies, process scheduling algorithms for multiprocessors, and system-level computer-aided design.

Mr. Simoni is currently a National Science Foundation Fellow.



Arturo Salz received the B.S. degree with honours in electronic and communication engineering from Universidad Iberoamericana, Mexico, in 1982 and the M.S. degree in electrical engineering from Stanford University, Stanford, CA, in 1985.

He is currently a Ph.D. student in the Department of Electrical Engineering at Stanford University. His research interests include high-performance processors, multiprocessor architectures, computer-aided design, and user interfaces.

From 1981 to 1983 he was Director of Technology Applications at Microsist S.A.

Steven Przybylski was born in February 1959 in Ottawa, Ont., Canada. In 1980 he received the B.A.Sc. degree with honours upon completing the Engineering Science programme at the University of Toronto. He received the M.S.E.E. degree from Stanford University, Stanford, CA, in 1982 and is currently working towards the Ph.D. degree, also at Stanford University. Between 1982 and 1984 he was a major contributor to the MIPS project at Stanford.

During 1985 he was with MIPS Computer Systems, of Sunnyvale CA, where he participated in the architectural definition and VLSI implementation of the R2000 CPU.

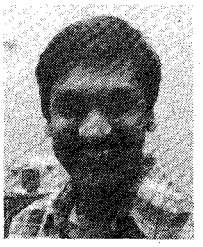
John Hennessy (S'72–M'77) received the B.E. degree in electrical engineering from Villanova University, Villanova, PA, in 1973. He received the Masters and Ph.D. degrees in computer science from the State University of New York at Stony Brook in 1975 and 1977, respectively.

Since September 1977, he has been in the Computer Systems Laboratory at Stanford University, Stanford, CA, where he is currently a Professor of Electrical Engineering and Computer Science, and Director of the Computer Systems Laboratory. He initiated the MIPS project at Stanford in 1981. During a leave from the academic life in 1984–1985, he cofounded MIPS Computer Systems and acted as Chief Scientist. His current research is in the area of high-performance architectures and software for such machines.

Dr. Hennessy is the recipient of the 1983 John J. Gallen Memorial Award and a 1983 Presidential Young Investigator Award.

Glenn Gulak received the B.A.Sc. degree in electrical engineering from the University of Windsor, Windsor, Ont., Canada, in 1978. While on an NSERC postgraduate scholarship, he received the M.Sc. and Ph.D. degrees in electrical engineering from the University of Manitoba, Winnipeg, in 1980 and 1984, respectively.

From February 1985 to January 1986 he worked on the MIPS-X project while on an NSERC University Research Fellowship. Since April 1987 he has been a Research Associate in the Information Systems Laboratory at Stanford University, Stanford, CA. His research interests are in digital communications, signal processing algorithms, computer architecture, and VLSI.



Anant Agarwal received the B.Tech. degree in electrical engineering from the Indian Institute of Technology, Madras, India, in 1982, and the Masters and Ph.D. degrees in electrical engineering from Stanford University, Stanford, CA, in 1984 and 1987, respectively. He is currently a post-doctoral research affiliate with the Computer Systems Laboratory at Stanford, where he worked on the architecture and design of the MIPS-X processor. His research interests include high-performance computer architectures, performance analysis and modeling, VLSI, and CAD tools.

John M. Acken (S'75–M'78) received the B.S. and M.S. degrees in electrical engineering from Oklahoma State University, Stillwater, and is currently working on the Ph.D. degree in electrical engineering at Leland Stanford Junior University, Stanford, CA.

From 1970 through 1973 he served in the armed forces as an electronics repairman. From 1978 to 1983 he was a member of the Computer-Aided Design Division of Sandia National Laboratories. Since 1983 he has been at Stanford University working on fault modeling for CMOS custom logic integrated circuits.