The VLSI Circuitry of RISC I

James B. Peek

Computer Science Division
Department of Electrical Engineering and Computer Science
University of California, Berkeley
Berkeley, California 94720

ABSTRACT

This paper describes the very large scale integrated circuitry and chip level architecture of RISC I, a Reduced Instruction Set Computer. RISC I is a single chip 32-bit microprocessor, designed with a simple, yet powerful architecture. Its major features include a 3 bus data path, a simple controller, and a 3 phase clock. The RISC I microprocessor, also known as RISC Gold, was implemented as part of the RISC project at the University of California at Berkeley during the winter and spring of 1981. The chip contains 44,500 transistors, and has been fabricated using 4 micron (minimum gate length) NMOS depletion load technology. Testing of the chip has shown that it is operational. It has been demonstrated running small programs.

June 2, 1983

Table of Contents

INTRODUCTION

The Reduced Instruction Set Computer project at U.C. Berkeley combines research and development in the areas of computer architecture, computer aided design, and integrated circuit design. The primary goal of the RISC project is to investigate computer architectures that make effective use of the resources offered by Very Large Scale Integrated (VLSI) circuit technology.

A single chip VLSI circuit, the RISC I microprocessor, was designed and built as a part of the RISC project. This chip is intended to be a learning vehicle for those people involved in the project, and to demonstrate to others interested in this field that a RISC architecture is appropriate for VLSI implementation. [Patterson 81]

Software was written for RISC I. There is a C compiler with optimizer, assembler, and linker. An architecture simulator was written to predict execution times of RISC I programs. Benchmark programs run using this simulator showed that a RISC I computer, running C programs, would perform at speeds comparable to larger and more complicated computers. [Tamir 81]

One advantage of a RISC architecture is that its implementation can be more regular than that of a complex architecture and require less design and testing effort to build a machine with a high level of performance. [Fitzpatrick 81] The goal of this paper is to describe the architecture and implementation of RISC I so that it can be better understood and its advantages used by those working in the fields of computer architecture and integrated circuit design.

Mapping the architectural definition of RISC I onto silicon took approximately 2.5 quarters. Three courses offered during the 1980-81 school year provided the opportunity for graduate students from the Electrical Engineering and Computer Sciences Department to learn about and design the RISC I chip.

Courses for RISC I	
CS248	A structured design approach studied
CS292X	RISC I and RISC II datapaths designed
CS292Y	RISC I completed, simulated, sent for fabrication

Computer aided design (CAD) programs were devoloped that made it possible to layout, check, and simulate the RISC I design in a relatively short period of time. These CAD tools were essential in getting such a large chip working correctly the first time.

CAD Programs Used to Design RISC I	
Graphic Layout Editors	Caesar , Kic
Circuit Extractors	Mextra , Cifplot -X
Plotting programs	Cifplot
Design Checking	LRC, ERC
Simulators	Slang , Mossim , Spice

The chip was implemented by MOSIS (DARPA's MOS Implementation Service at the University of Southern California's Information Sciences Institute). We submitted a design file in Caltech intermediate format (CIF). This file contained the topological data for the 44,500 transistors of RISC I implemented in 4 micron NMOS depletion load technology with butting contacts. MOSIS fabricated the circuits, and returned wafers and bonded chips to us for testing.

The returned chips were sorted and tested. Some were found to operate correctly. These could be clocked at speeds up to 500 nano-seconds per active phase. This is approximately 1/2 the hoped for speed [Foderero 82]. One non-fatal design bug was found during testing (discussed below), and our compiler was modified to avoid this bug. The RISC I single board computer was demonstrated in the spring of 1982 with a RISC I chip running simple C programs.

CHAPTER 1 RISC I Architecture

The architecture for RISC I can be described as a high performance design that has been kept simple enough to fit onto one chip. The chip and its instruction set were designed to carry out each instruction in only one clock cycle. Two cycles are used for the load and store instructions.

RISC I uses registers for address and data manipulation, and a program counter (PC) for instruction fetching. The processor status word (PSW) holds the control state of the processor and can be written and read under program control. It contains the condition code information, and the window number (discussed below).

A single RISC I instruction can address 2 source registers, specify the operation to be performed, and indicate a destination register. RISC I makes exactly one memory reference each clock cycle. Instruction fetching, instruction decoding, and instruction execution all occur during the same cycle. While carrying out the current instruction, RISC I decodes the opcode of the next instruction, and sends to the memory the address for the instruction after the next instruction. Registers in the large register file are grouped into windows to achieve good performance in handling procedure calls and returns. [Halbert 80]

The RISC I microprocessor has a 32-bit byte address that can be mapped off chip to implement virtual memory. Data can be stored, transferred, and processed in 3 sizes: 8-bit bytes, 16-bit short words, and 32-bit long words. The two major data types supported by RISC I are signed two's complement integers and unsigned logicals.

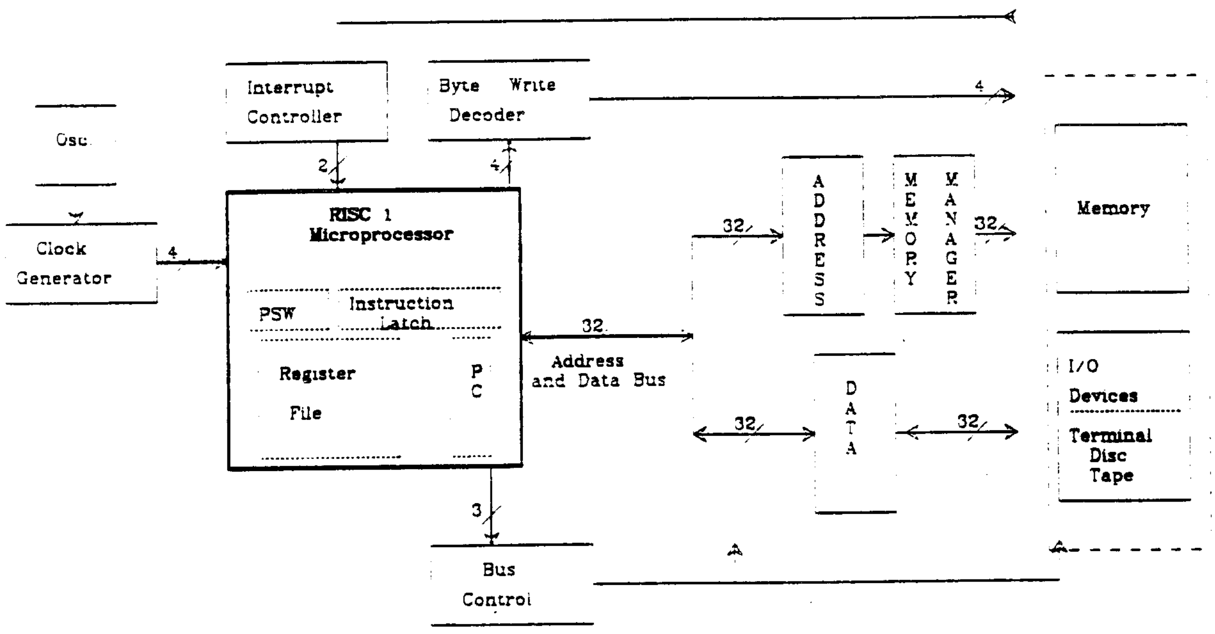


Figure 1: RISC I Single Board Computer

The diagram in figure 1 shows the RISC I chip with connections to the rest of the system as implemented in the RISC I Single Board Computer. This system was designed, built and demonstrated with a working RISC I chip during the winter and spring of 1982. Virtual memory management, bus control, serial and parallel ports, clocks and timers make up the rest of the system. [Van Dyke 82a]

Instructions

The complete set of reduced instructions for RISC I contains only 39 instructions. Complex instructions and addressing modes are executed as sequences and subroutines of these fast, simple RISC I instructions. [Patterson 80] It may seem that this approach would lead to very large programs, but our studies showed that RISC I program sizes are on average 0.8 to 1.4 times those of other comparable processors. [Patterson 82]

There are three types of instructions in RISC I's reduced instruction set:

[1] data operation instructions perform ALU and shift operations;

[2] control instructions include jumps, calls, and returns;

[3] memory access instructions include only single loads and stores.

Opcodes

The 39 RISC I instructions and their 7-bit opcodes are shown in figure 2 below.

Least Signif. 4 Bits \ Most Significant 3 Bits	0 000	1 001	2 010	3 011	4 100	5 101	6 110	7 111
0000 0	CALLI	SLL	ILLEGAL	STXH	AND	SUB	RET	LDHI
0001 1				STRH				
0010 2		GETPSW				SUBI		
0011 3								
0100 4			LDXBU	STXB	XOR	SUBC	RETI	
0101 5			LDRBU	STRB				
0110 6			LDXBS			SUBCI		
0111 7			LDRBS					
1000 8	CALLX	SRL	LDXW	STXW	OR	ADD	GTLPC	
1001 9			LDRW	STRW				
1010 A	JMPX	PUTPSW						
1011 B								
1100 C	CALLR	SRA	LDXHU			ADDI		
1101 D			LDRHU					
1110 E	JMPR		LDXHS					
1111 F			LDRHU					

Figure 2: RISC I Opcodes

The "RISC I ISP Description" contains a detailed description of what actions these instructions perform. [Corcoran 80] A table showing the major actions that occur on the chip for each instruction is presented in Appendix A of this paper. The only complete description of all the actions performed by RISC I for each instruction is the SLANG simulation description (discussed in chapter 3). [Foderaro 81]

There were 31 instructions in the original RISC I instruction set. Eight load and store relative instructions were added near the end of the design of RISC I. They were added because they can be useful to a compiler and required only a slight reprogramming of the PLA that decodes the opcodes.

Operands

Most operands for RISC I instructions come from on chip registers. The data operation instructions can access 3 registers, or 2 registers and one immediate operand. The condition codes can be set or not set, and they follow the DEC PDP-11 format.

[Register C] ← [Register A] + [Register B or 13-bit Immediate]

RISC I control instructions include offset-indexed jumps and PC-relative jumps. They can access 2 registers, or 1 register and one immediate operand to determine the jump address for the next instruction.

[PC] ← [Register A (index)] + [Register B or 13-bit Immediate (offset)]
[PC] ← [PC (pc relative)] + [13-bit Immediate (offset)]

The memory access instructions supported by RISC I use 2 basic addressing modes: offset-indexed and PC-relative. They can access 2 or 3 registers: one holds the data, and two other registers, or one register and immediate operand, are used to compute the effective address for the external memory.

[Effective Address] ← [Register A] + [Register B or 13-bit Immediate]
[Effective Address] ← [PC (pc relative)] + [13-bit Immediate]
[Register C (load)] ← [Memory contents of Effective Address]
[Register C (store)] → [Memory contents of Effective Address]

When bytes and short words are loaded into RISC I they are shifted from any position in memory to the least significant position in RISC I's registers. The sign bit of the data can be extended to fill the higher bits. Incrementing and decrementing of the addresses is done explicitly with data operation instructions (add and subtract).

Instruction Formats

Instruction Formats for RISC I					
OPCODE<31-25>	SCC<24>	DEST<23-19>	SORC1<18-14>	IMF<13>	SORC2<4-0>
OPCODE<31-25>	SCC<24>	DEST<23-19>	SORC1<18-14>	IMF<13>	IMM<12-0>
OPCODE<31-25>	SCC<24>	DEST<23-19>	LDHI IMMED<18-0>		

All RISC I instructions are 32 bits long. The bit fields for the instructions are shown above. These fields within the instructions are located in the same positions in all instructions to simplify the task of decoding them in hardware.

The most significant 7 bits always contain the code for the operation to be performed (OPCODE). The 24th bit tells whether the condition codes are to be set. This is followed by the 3 operand fields (DESTINATION, SOURCE1 and SOURCE2). The immediate flag bit (IMF) indicates whether the second source operand is contained in a register or the last 13 bits of the of the instruction. To allow the use of 32-bit immediates a "load high" instruction is supported by RISC I. It loads a 19-bit immediate into the upper 19 bits of the destination register.

Registers

All RISC I instructions access at least one on-chip register. The program counter (PC) and processor status word (PSW) are the two special purpose registers on RISC I. All other RISC I registers are located in the register file. They are general purpose registers to be used for handling data and addresses. All of these general purpose registers are 32 bits wide. In any procedure 32 such registers can be accessed.

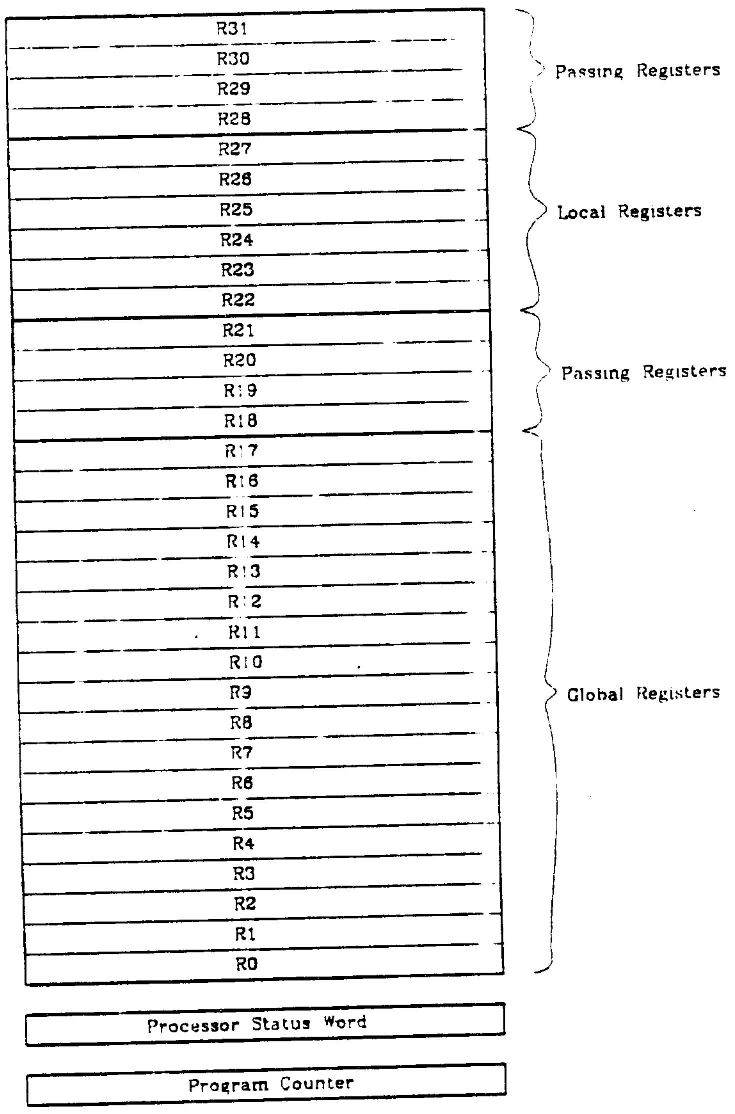


Figure 3: RISC I Register Programming Model

Figure 3 shows the registers that can be accessed by a RISC I program. Registers 0-17 are global registers; they can be accessed from any procedure (window). Register 0 (global) is wired to always contain the value zero. Registers 18-21 are 4 "next overlap" registers. These are useful for passing parameters with the next called procedure. Registers 22-27 are a group of 6 "local" registers that can be accessed only from the current procedure. Registers 28-31 are the 4 "previous overlap" registers. They can be used to pass parameters with the previous, or calling, procedure.

Windows

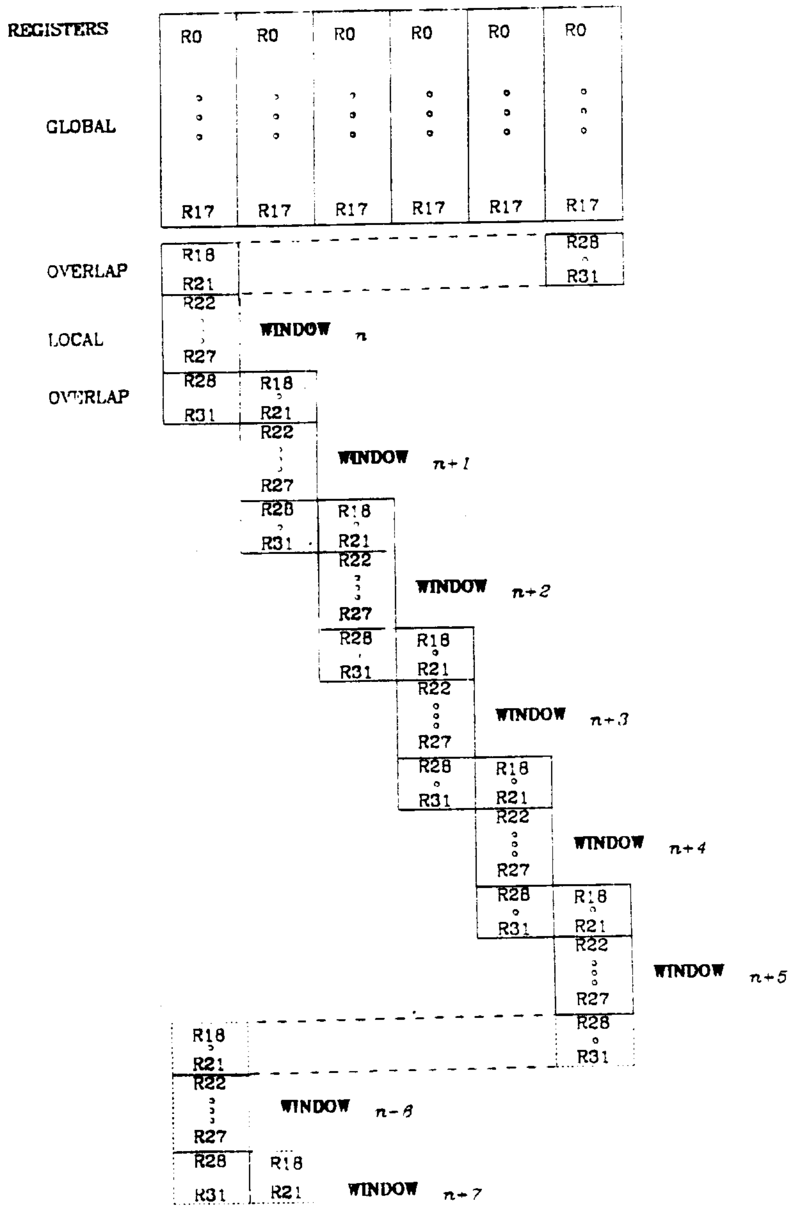


Figure 4: RISC I Windows of Registers

RISC I allocates a window of registers to each procedure. The window contains those registers that can be accessed within a proceedure. The arrangement of the windows is shown by figure 4.

There are 6 windows on the RISC I chip; each window is pointed to on the chip by a number from 0 to 5. Procedures are entered by call instructions which decrement the current window number. Return instructions increment the current window number. When all 6 windows on chip are either emptied or fully used, a trap (interrupt) occurs to force the processor to jump to a window handling routine.

Chip Architecture

The RISC I microprocessor chip was designed as three major functional blocks. These blocks are the the clock circuitry, the control unit, and the data path.

The clock circuitry synchronizes the control signals and provides order for the events on chip. The RISC I control unit is much simpler than most microprocessor controllers. Its job is to handle the PSW, and latch and decode the incoming opcode. The bits of the decoded opcode are latched and gated with clock signals, then sent to the data path.

The RISC I data path occupies most of the chip. It can operate on 32-bit quantities and consists of 4 major modules. These are the register file, shifter, ALU, and PC. Data and pointers (data-addresses) are stored in the register file. The PC contains the addresses used to fetch instructions. The shifter and ALU perform operations on data in the register file or PC. In order to facilitate the conditional jump instructions, the condition code information from the ALU is latched into the PSW (part of the control unit) where the desired condition can be tested.

The diagram in figure 5 below shows the major components on the chip.

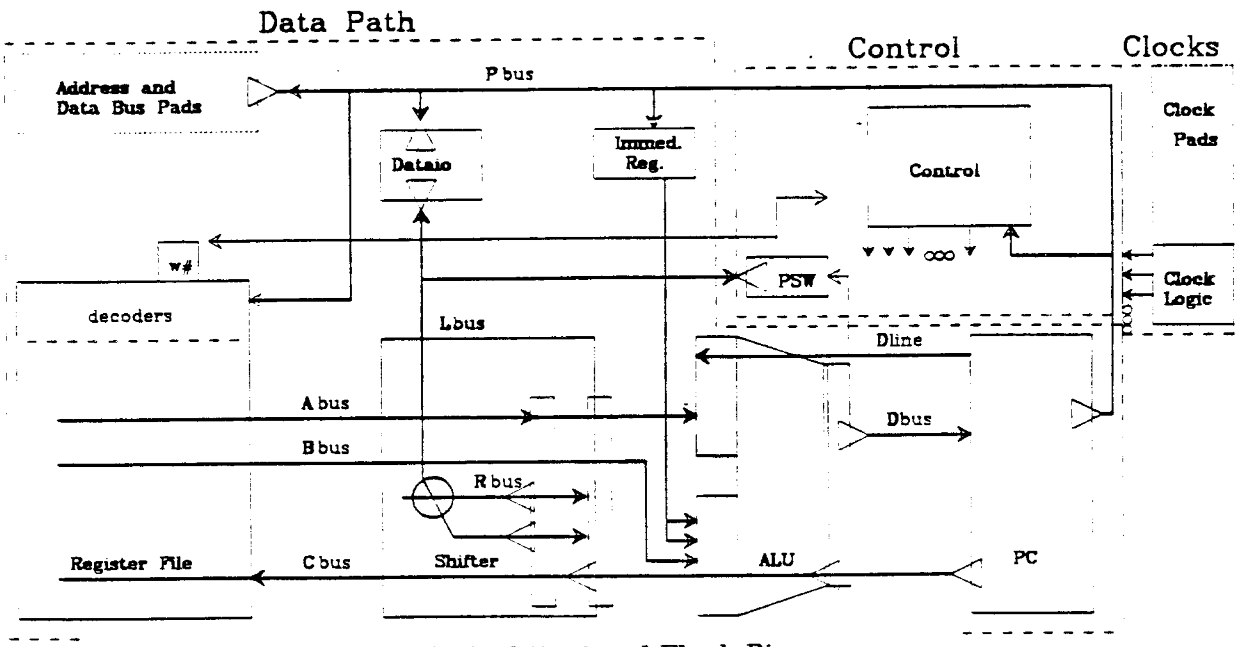


Figure 5: RISC I Architectural Block Diagram

The RISC I design makes effective use of some of the circuit configuations associated with VLSI. The shifter on RISC I uses a 32-bit cross bar which can shift data by an arbitrary amount in one clock cycle [Sherburne 82]. All of the control logic is implemented in programmable logic arrays (PLA's). A large register file (2496 bits) is used like a data cache. The chip generates 32-bit byte addresses and processes 32-bit data. Data and addresses are multiplexed across the same 32 pins so that only 48 pins are required to connect RISC I to the external world.

CHAPTER 2 RISC I Timing

The timing cycle for RISC I is composed of a cycle of three non-overlapping clock phases. The data operation instructions occur in 3 phases: the operands are read; the modification is performed; and the result is stored. The basic idea behind the 3-phase clock is that one clock phase is used to drive each part of an instruction's execution cycle [Katevenis 80]. The three phases are generated by an external oscillator and enter the chip via three input pins (one for each clock phase). This allows for easy external synchronization and testing. All RISC I instructions fit into some sort of 3 phase sequence as described below.

In RISC I data operation instructions the following three things occur:		
1] data operands read	2] modification is performed	3] result is saved

In RISC I control instructions the following three things occur:		
1] address operands read	2] jump address calculated	3] jump is taken or not

In RISC I load and store instructions the following six things occur:		
1] address operands read	2] effective address calculated	3] effective address sent out
4] data loaded on chip	5] byte alignment is performed	6] data is saved in reg.
4] reg. data is read	5] data is sent off chip	6] data stored in mem.

The External Clock

The timing diagram for the external clock is shown in figure 6 below. It is composed of three separate active periods.

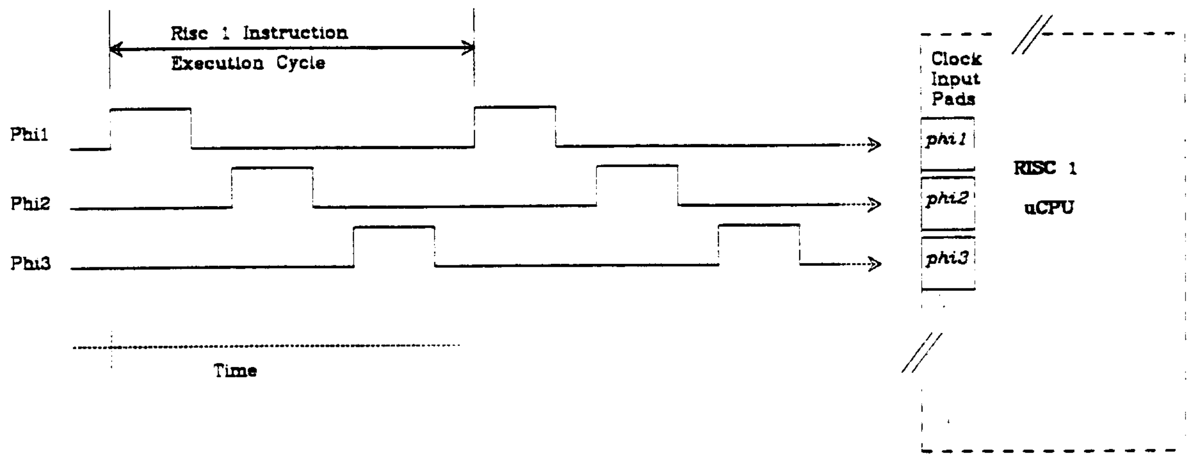


Figure 6: RISC I 3-Phase clocks

It is important that none of the phases overlap. This is to insure that one internal event is completed before another is begun. These clock phases are used internally by RISC I to gate the control signals which read and write to internal latches. Non-overlap time is essential for digital latches as it allows for the set up and hold times needed to reliably latch data. In some designs this delay between phases is handled by circuitry on the chip. For RISC I we chose to keep this constraint external to simplify the design and allow detailed performance testing. If the non-overlap delay is to be handled on chip, the clock skews and time delays of certain critical signals need to be known and accounted for by on-chip clock circuits.

On Chip Sequencing

The clock circuitry on RISC I cycles through sequences of 3 out of 8 possible phases, synchronously with the 3 external phases. The selection of the phases depends on whether the instruction is a "normal" instruction, a memory access instruction, or an interrupt.

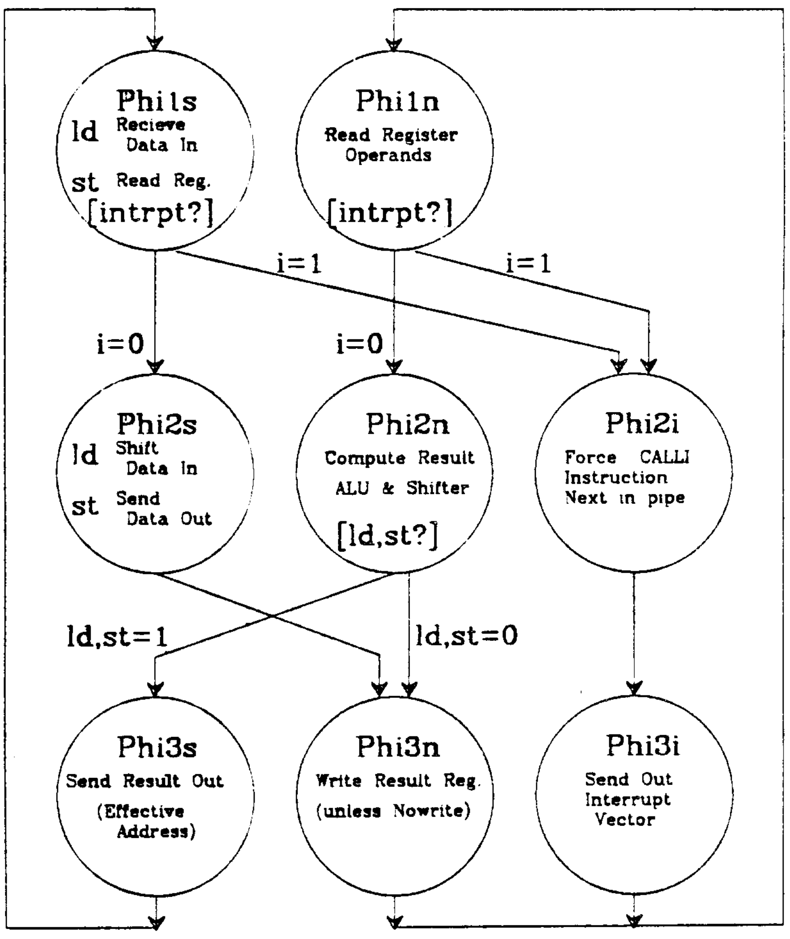


Figure 7: RISC I Clock-State Sequencing

The 8 phases and their transitions are shown in figure 7 above as a timing state diagram. Once on chip, each of the 3 external phases is read and used to generate 1 of 8 different internal clock phases. These 8 phases (or time-states) are used to sequence all of the circuitry through each instruction and interrupt. The clock circuitry generating these 8 phases is the heart of the machine. It can be described as a hardwired, time-state controller. This sequence of state transitions is analogous to the state transitions of the micro sequencer found in most computers, but the RISC I sequencer requires no microcode, and no micro-instruction sequencer. On most microprocessors approximately 50% of the chip area is devoted to the microcode memory and sequencer. The RISC I clock circuitry occupies less than 1% of the chip. The clock and control circuitry combined occupy only 6% [Sequin 82]. This may be the major advantage of this architecture for VLSI implementation.

Most RISC I instructions execute in the 3 phases: Phi1n, Phi2n, Phi3n. Only the load and store instructions require more time, and they execute in 6 phases. They require one cycle of the 3 normal "n" phases with an additional cycle of 3 secondary "s" phases. This extra time used by load and store instructions is required because the data path must do 2 major things for these instructions:

[1] Calculate the effective address and send it out.

[2] Send data to, or receive data from the external memory.

Interrupt Handling

Interrupted instructions are handled in 3 phases. The interrupt is accepted during phase 1 (Phi1n or Phi1s). The clock circuitry issues 2 subsequent interrupt "i" phases which abort the current instruction as if it had never occured. After accepting an interrupt, Phi2i and Phi3i are issued instead of the normal Phi2n and Phi3n or secondary Phi2s or Phi3s. Interrupts after Phi1s are allowed so that page faults and addressing errors can be handled.

During Phi3i the interrupt vector is sent out to allow RISC-I to jump to its interrupt handling routine.

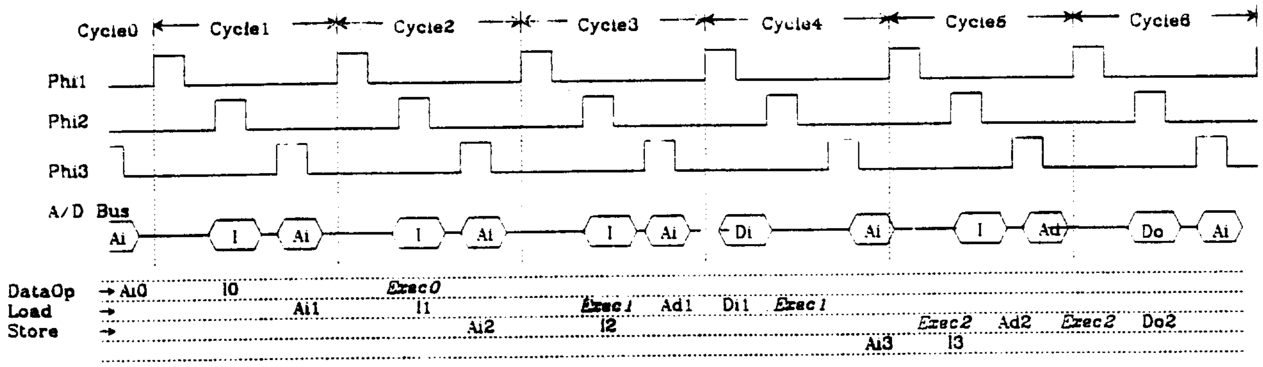
The following things occur during an interrupt:		
1] accept interrupt	2] force calli instruction next	3] send out interrupt vector

In RISC I the interrupt acceptance bit is part of the clock circuitry, and is not part of the PSW. It allows interrupts to occur or be ignored. It is set to allow interrupts after a "return from interrupt" instruction, and reset to ignore interrupts after an interrupt has occured. Interrupts occur due to window over or underflow, external signals, or receiving a "call interrupt" instruction. The reset pin causes an interrupt whether the interrupt acceptance bit is on or off.

Diagnostic Stopping

The RISC I chip is designed so that, during testing, it can be stopped after any clock phase to allow diagnostic read-out of its internal state. Serial scan-in-scan-out loops exist in the latches of the shifter, ALU, PC, and control circuits. When the chip is stopped the tester's clock must hold all three clock phases low. A special refresh pin (PhiR) can be asserted to refresh all memory on chip. The PhiR pin plus nine other special test pins are used to scan data in and out of the chip. When the chip is running normally all on chip memory gets refreshed by the three phase clocks.

Instruction Fetch Timing



Ai → Address for instruction out from RISC 1
Ad → Address for data out from RISC 1
I → Instruction into RISC 1
Do → Data out from RISC 1
Di → Data into RISC 1

Figure 8: RISC I Instruction Timing

The diagram in figure 8 above shows the timing of three complete instructions and part of a fourth. Fetching the first instruction (Ai0) occurs during phase 3 of cycle0. This first instruction (I0), a simple data operation instruction, is received from memory during phase 2 of cycle1. It is executed during cycle 2 (Exec 0). This instruction is followed by a load instruction, and then a store instruction.

RISC I prefetches instructions and thus has a delayed jump. The jump takes effect only after the next instruction, fetched after the jump, is executed. A special pipeline flush circuit could correct this by flushing the pipe after all jumps, but this would not make it any faster. Instead the compiler inserts no-op instructions after jump instructions. The optimizer then looks over this code to reassemble the order of instructions, eliminating most of the no-ops, to make the code run faster. No-op instructions are left only after jumps that cannot be optimized. [Patterson 82]

Clock Circuitry

There are six input pads for the clock phases and clock controls. These are labeled: Phi1, Phi2, Phi3, Interrupt, Reset, and PhiR. The circuitry is shown in figure 9 below.

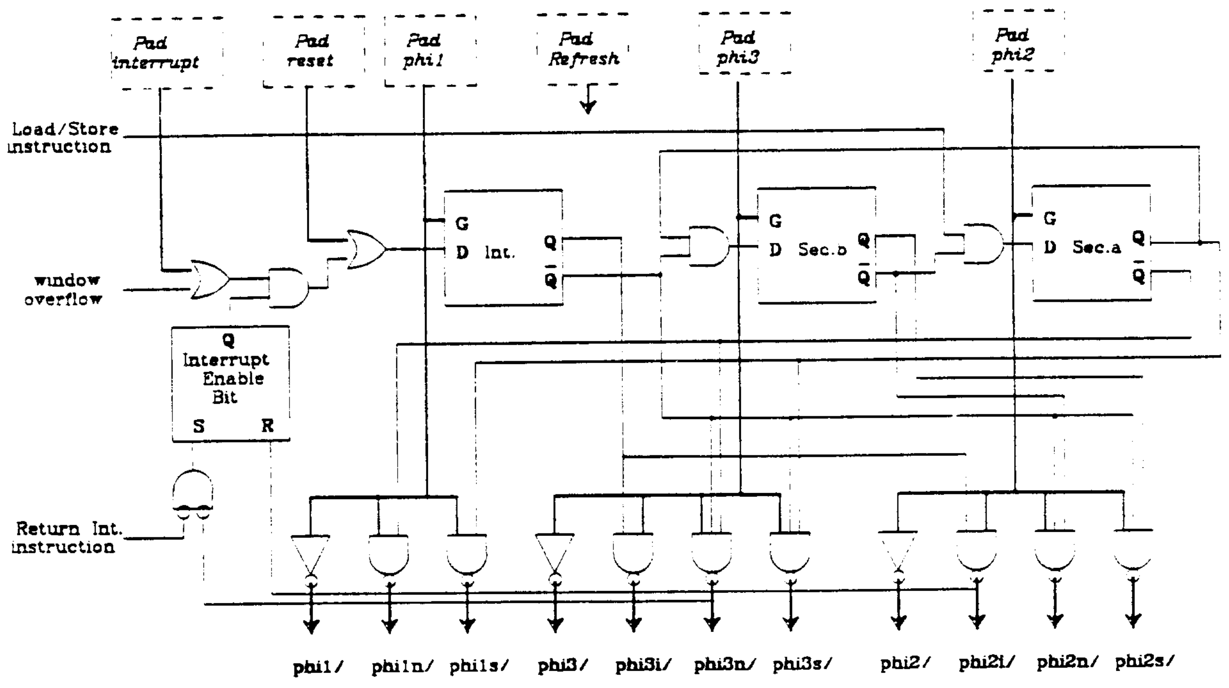


Figure 9: RISC I Clock Circuit Diagram

The logic that implements the 8 clock phases contains 4 bits of state. A set-reset latch and a D-type latch allow the interrupt phases to occur. Two other

D-type latches sequence the load and store instructions through their secondary phases.

The following diagram, figure 10, shows the layout of the clock circuit.

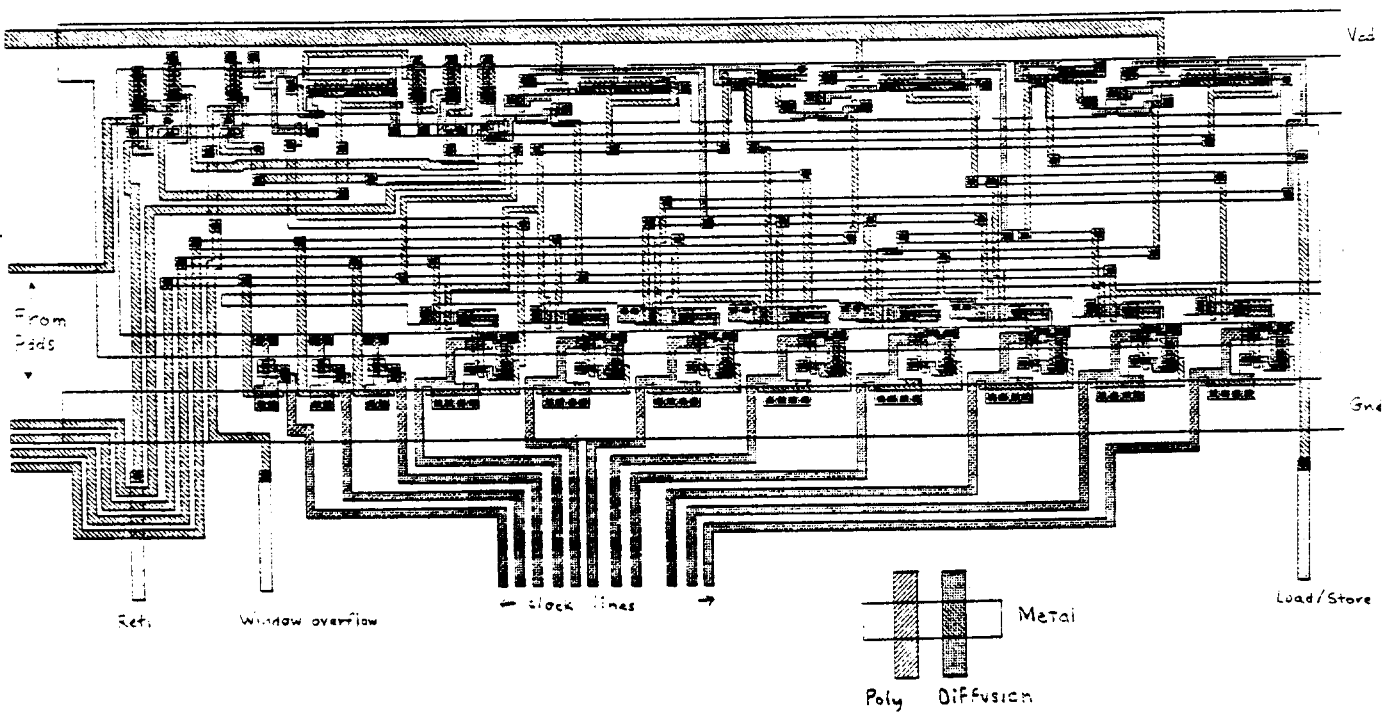


Figure 10: RISC I Clock Circuit Layout Plot

The clock circuit's latches, NAND gates, and drivers are located on the upper right edge of the chip next to the 6 clock input pads. This location was chosen to minimize the distance from clock pads to clock-control gates in the datapath. Approximately 100 transistors are used in this circuit and it occupies the amount of area required for 3 input pads.

CHAPTER 3 RISC I Data Path

The data path in RISC I is designed to process 32-bit numbers. It stores, computes, and transfers 32-bit numbers as the basic units of data. It occupies approximately 69% of the area of the chip. External routing and pads occupy 25% of the chip, with control using the remaining 6%. The data path contains 42,000 transistors and consists of 4 major modules and 2 minor modules. The four major modules are the register file, shifter, arithmetic-logic unit, and the program counter. The two minor modules are the data input-output port, and the immediate latch.

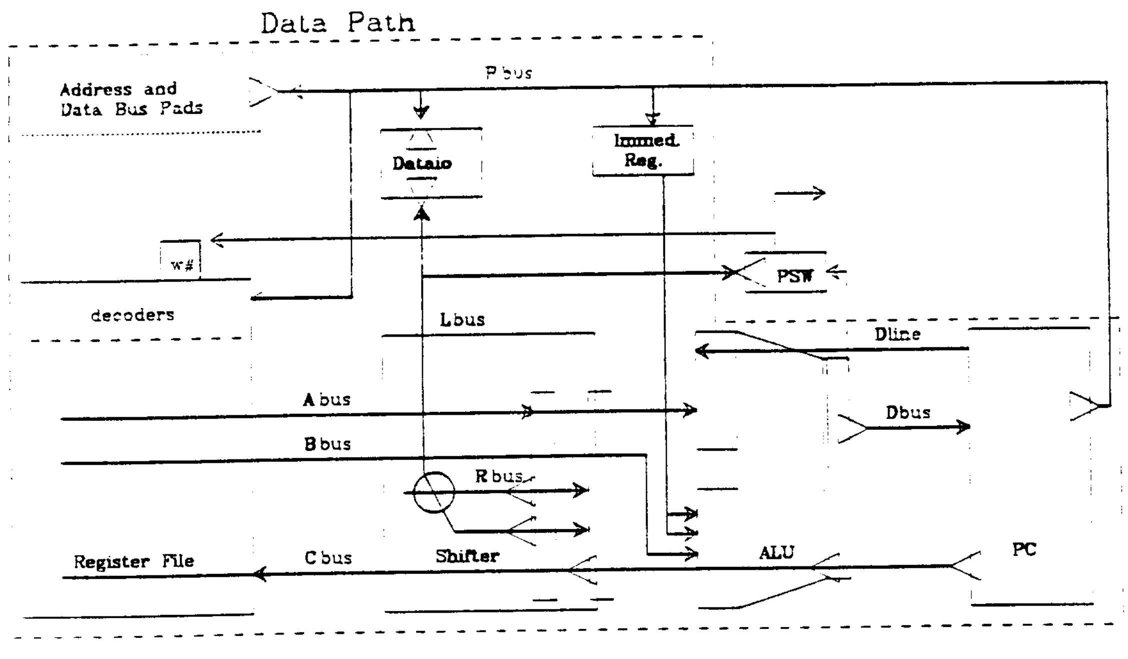


Figure 11: RISC I Data Path Block Diagram

Figure 11 above shows the components of the RISC I data path. All data processing operations occur in the data path. Data and data-addresses (pointers) are stored in the register file. Instruction addresses are contained in the PC. The ALU can be used to modify data in the register file and the PC. The shifter can be used on data in the register file as well as to shift data during load instructions.

Technology and Layout

Three layers are used to form transistors and interconnections in the NMOS technology used for RISC I. The first layer consists of the substrate silicon (single crystal, p-type), with specially patterned diffused regions of n-type silicon. The next layer up is a layer of poly-crystaline silicon (n-type). Above both of these layers is a layer of metallic aluminum. Between each of these three layers insulating silicon dioxide is deposited. Contact cuts can be placed to allow electrical connections between metal and diffusion or metal and poly.

Transistors are formed by the overlap of a poly-silicon region over a diffused region. Two types of transistors can be made: enhancement transistors that can be turned on or off, depletion mode transistors that are always on and act as resistive loads. Transistors are interconnected by lengths of metal, poly, and diffusion.

Figure 12 below shows a photo micrograph of the RISC I chip.

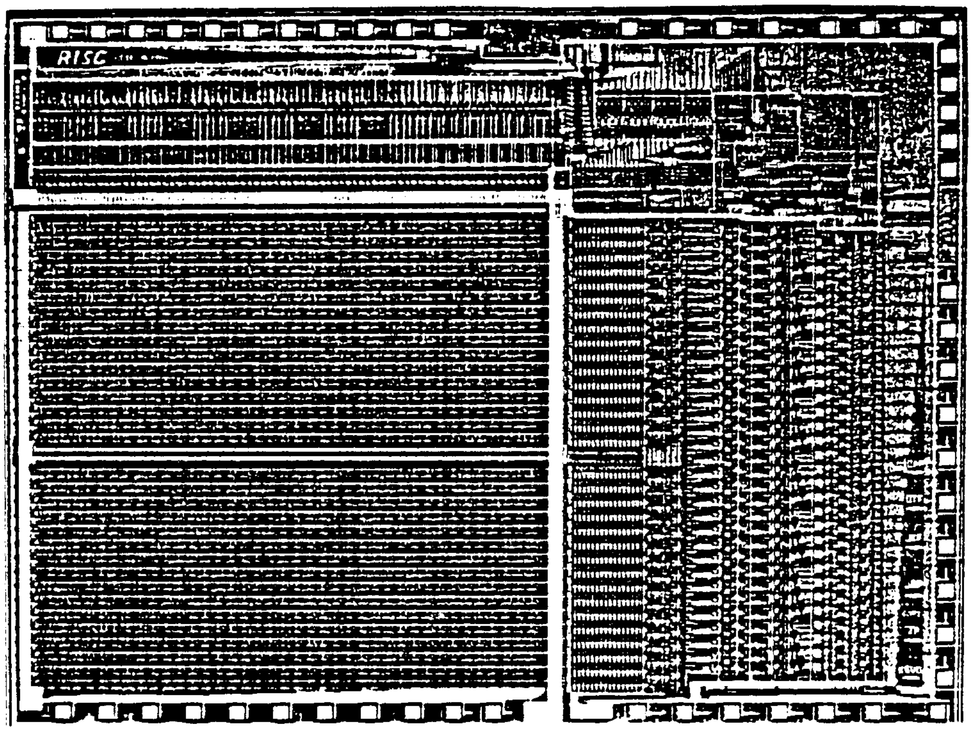


Figure 12: RISC I Chip Photo-micrograph

The metal layer has the lowest resistance for a given length and width of interconnection. It is especially important that the ground connections are routed in metal. This is because the basic circuits operate by amplifying the voltage difference between the gate (input), and source (ground) of an enhancement transistor, and through this ground connection flows all of the current in the depletion transistor load and the transient current required to drive the external capacitive loads.

Five 32-bit busses are used to carry data between modules in RISC I. Two different types of busses are used. The bus that connects the Dataio and PC to the I-O Pads is a tristate bus. It must be driven both high and low; it is labled "P bus". The other four busses are precharged busses. They are the A, B, C, and D busses. These are used to pass data between the modules of the data path. In the clock phase before they are used all wires are charged to a logical high voltage. This voltage is approximately 3 volts. Reading out onto these busses is done by discharging only the bus wires that should be zeroes.

In the RISC I design, the control signals in the data path run perpendicular to the bussed data signals. Metal is used for the signals in one direction, and poly used for signals in the other direction. The register file and cross-bar have metal data busses and poly control lines, whereas the rest of the data path has poly data busses and metal control lines. Clock signals were routed in metal as much as possible to minimize clock skew.

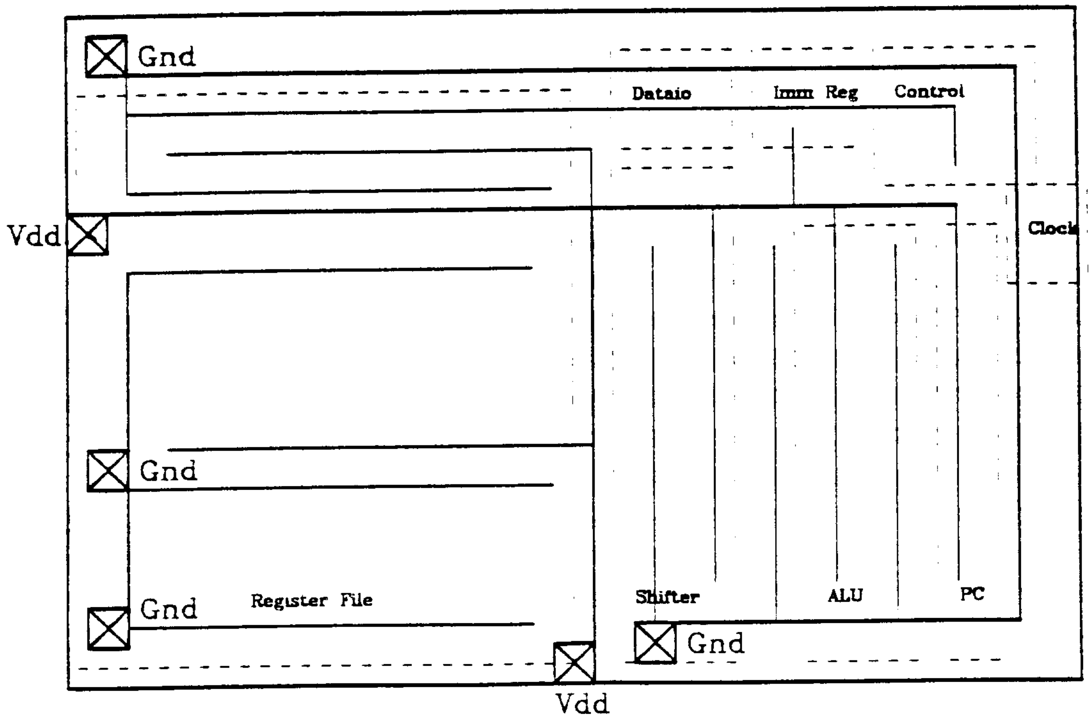


Figure 13: RISC I Metal Routing

The circuits in RISC I were layed out as cells which were placed and routed using the Caesar graphics editing program. [Ousterhout 81] The data path was designed using a bit slice approach. Each 1-bit slice of the data path was stacked 32 high. All cells in the major data path are of the same height, with differing widths. Some routing was required between cells, but almost all of the connections were made within the cells which greatly simplified the routing job.

Design rule checking was carried out by the program "LRC". [Baker 80] Static logic interconnections were checked by the "ERC" program. The circuits were simulated by the logic simulator "ESIM" [Terman 82] as simple cells, as parts of the modules (e.g. as the ALU), and as the entire chip with all 44500 transistors. Slang was used as the user interface for ESIM. Some of the cells were simulated with the circuit simulator SPICE. [Nagel 75]

Register File

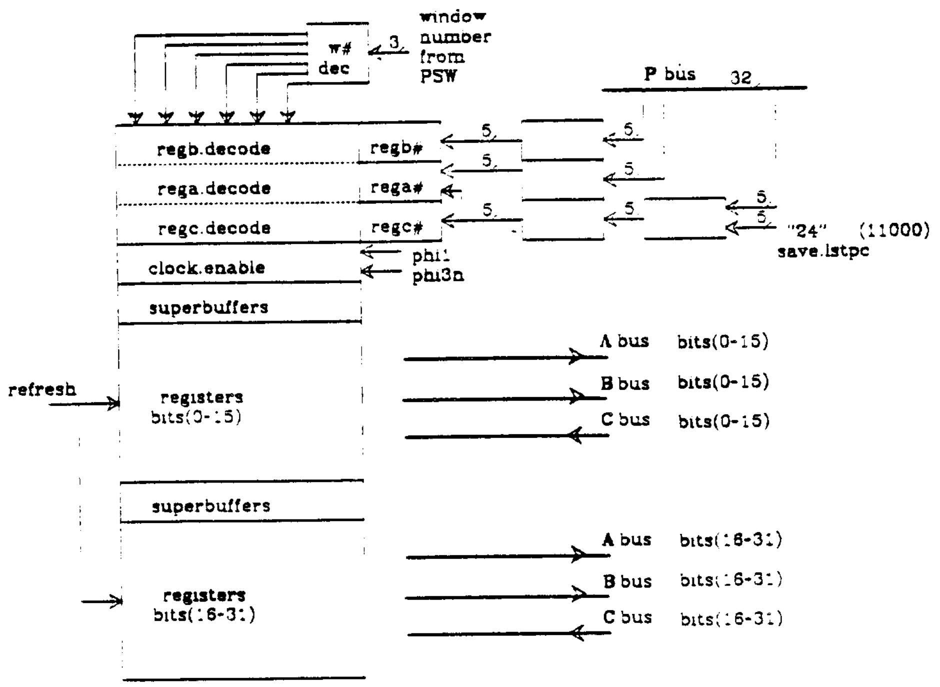


Figure 14: RISC I Register File Block Diagram

The register file in RISC I is shown above in figure 14. It contains 78 registers; each 32 bits wide. The supporting circuitry consists of: seven 5-bit register address latches; three register number decoders (1 of 78); three sets of 78 synchronizing gates; and six groups of 78 super buffers.

Register Address Latches

The register address latches receive the source and destination fields during an instruction prefetch. They drive the inputs to the decoders later during that instruction's execution cycle. The values are obtained from the P bus during phi2n. One of the fields must be changed on store instructions where the "dest" field is used to address the register with the data to be stored. These latches drive each decoder at least one clock phase before the outputs are needed.

The diagram below in figure 15 shows the circuitry and layout of the 1-bit latch cell used for these latches. This cell is loaded from the right side, with buffered Q and Q/ outputs to the left.

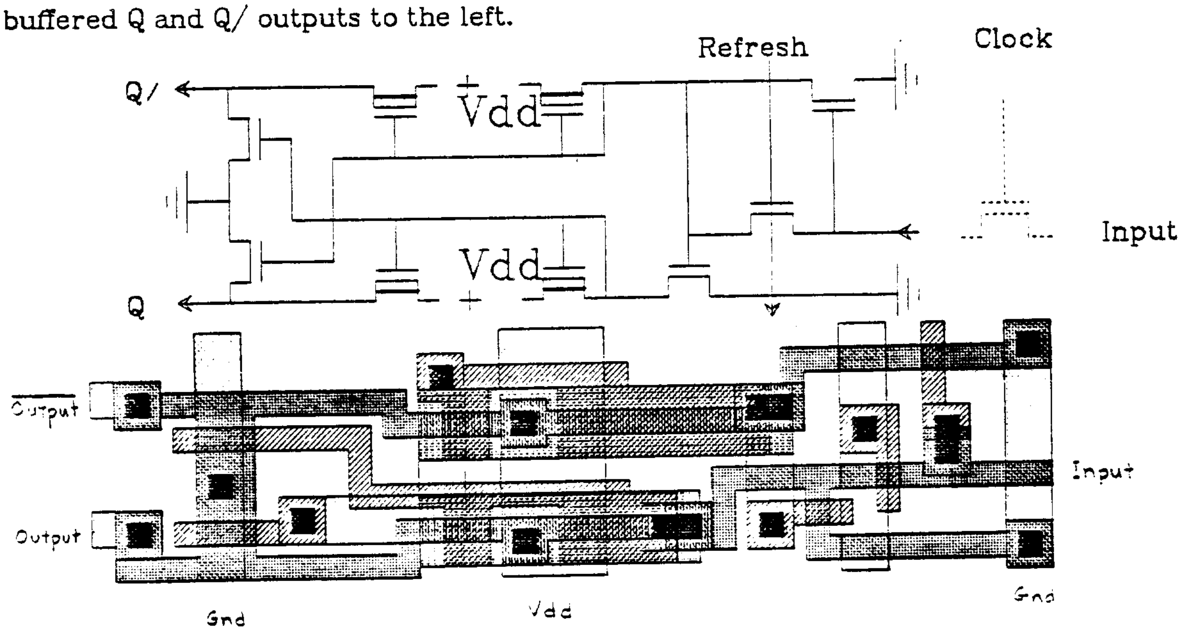


Figure 15: RISC I 1-bit Register Address Latch

Window Decoder

Individual registers are selected with a 5-bit register address number and a 3-bit window number. The window number latch and decoder circuitry is located on top of the register file decoders. It drives six window enable signals into the decoders, one for each window. One of the six lines is driven low to select the current window. The window control circuitry is discussed in chapter 4.

Register Decoders

In RISC I there are three register address decoders. The destination address is decoded by the top decoder. In the middle is the decoder for source2. The source1 decoder is on the bottom. The three decoders each activate 1 out of 78 outputs. The outputs of the register address decoders are active high.

The decoders are implemented as 6 input negative input AND (NOR) gates. Each gate is followed either by an inverter for a register with one address, or a 2 input NOR gate for the window overlapping registers which have two different addresses.

Enable Gates and Super Buffers

Under the decoders is a row of clock enable gates which synchronize the negative outputs of the decoder with the internal timing phases. They also allow the decoder outputs to be ignored during transitions. These gates are implemented as two input negative input AND (NOR) gates. The outputs of these enable gates are positive true signals which drive a row of non-inverting buffers which then drive the control lines into the register file.

The Registers

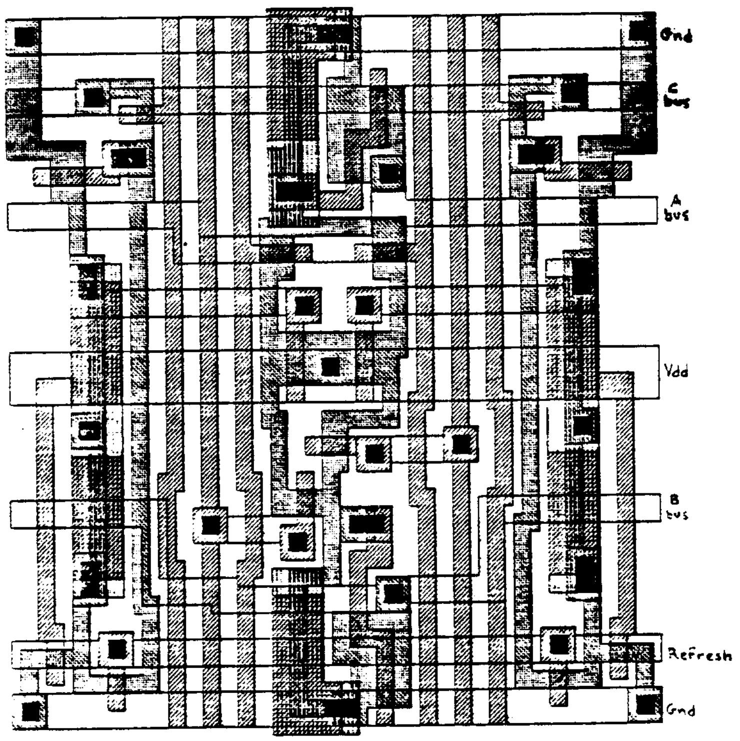
The RISC I register array was designed as two blocks of 78 16-bit registers. These two blocks are placed one on top of the other, with a row of non-inverting super buffers placed to drive the control lines between them to help speed up the read and write times. Bit 0 of the register file is on top nearest the decoders with bit 31 on the bottom.

Three 32-bit data busses connect to and pass through the register file. These are the A, B, and C busses. The A bus and B bus are used for reading the source1 and source2 data out from the register file. They can read from the same or 2 different registers. During every phi1n the register file reads data onto these busses.

The C bus is used to bring data into the register file. The ALU or Shifter can write onto this bus during phi3n and the register file will read from it unless prevented by the controller via the "nowrite" control line.

These busses are precharged to a logical high voltage level in the phase before they are used. Reading onto these busses is accomplished by discharging the bus lines that are to be read as zeroes and leaving the other lines high to be read as ones.

Register 0 (global) is internally wired to read out the value 0. It can be written into, but will always contain the value 0. This simplifies the addressing modes in RISC I. [Patterson 81]



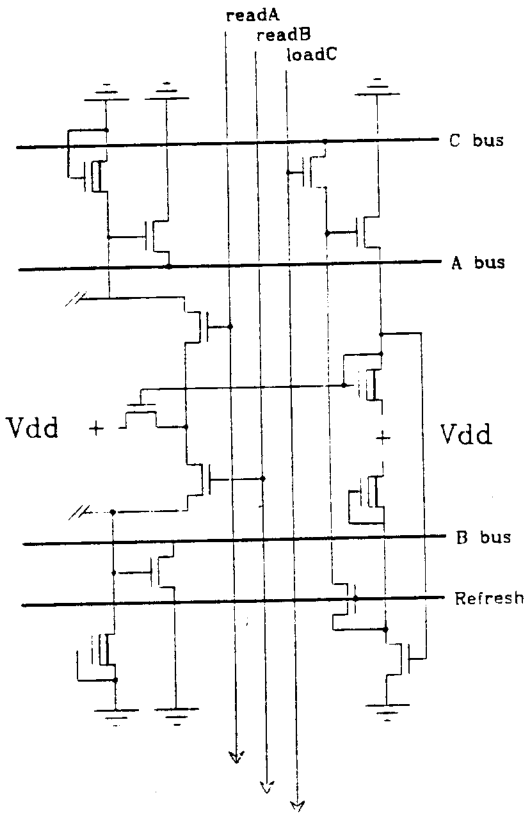


Figure 16: RISC I Register Cell

Register Operation

The register cell layout and circuit diagram are shown above in figure 16. Each register cell is designed as a 22 transistor 2-bit circuit. It is designed with 2 bits so that adjacent registers can share bus read out drivers. Each register consists of a semi-static latch with circuitry to load from the C bus and read to the A and B busses.

The semi-static latch is made up of 2 series connected inverters with an enhancement (or refresh) transistor connecting the output of the second inverter to the input of the first. This connection forms a bistable network which can either latch to a high or low voltage depending on its "initial conditions".

Two sets of 3 poly-silicon control lines run vertically inside each cell. They are used to load from the C bus, read to the A bus, and read to the B bus. Registers are loaded from the C bus through a pass transistor activated by the load control line during Phi3n. During this phase the refresh transistor is off. This action sets up the initial condition for latching to the desired state during the next Phi2n when the refresh transistor is activated.

The registers read data out to the A bus and B bus when the read control lines are activated during Phi1. The circuitry that reads out to the busses for this cell has some unusual characteristics. It buffers the latch from the busses being read to so that even with the feedback transistor on, the data in the latch will be undisturbed.(A simple pass transistor output to a precharged bus can upset data in a latch if the feedback transistor is still active at the start of reading out to the bus.) The transistor connected to the control line is a small one; only twice minimum size. Pass transistors used to read to a bus are typically much larger than this. Because the circuitry works as a non inverting gate only the circuit with a logical high output (reading a register) dissipates any power. This is important where there are 2496 such circuits and only 64 ever need to be active at the same time.

One refresh control line runs horizontally through each register cell. It is activated once each clock cycle during the Phi2n clock phase, and during PhiR.

Shifter

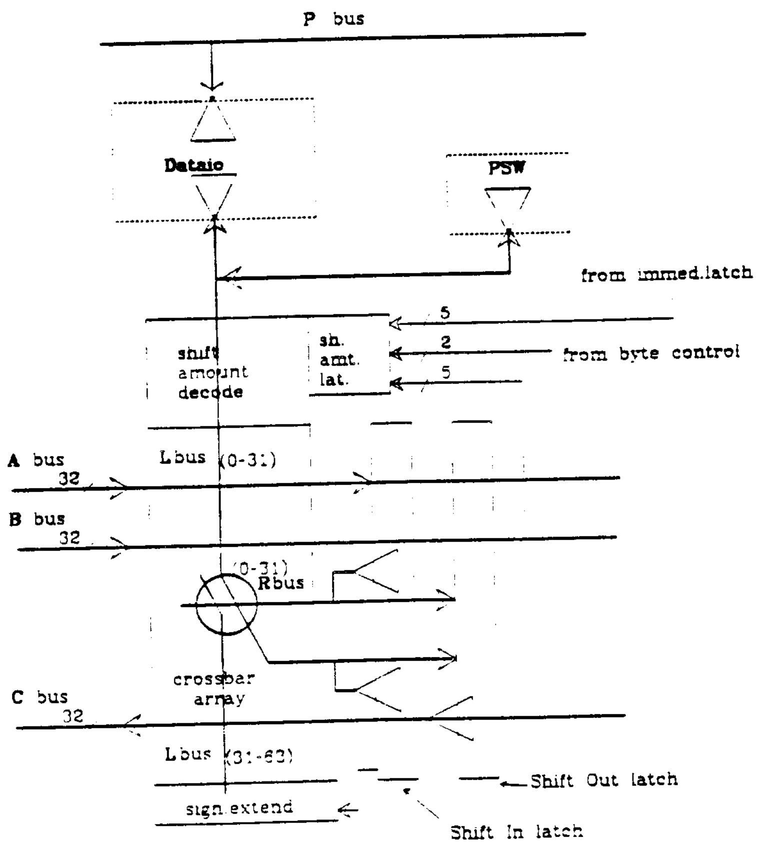


Figure 17: RISC I Shifter Block Diagram

The shifter on RISC I is implemented as a cross bar array with input and output latches. A decoder located above the crossbar is used to control the shift amount. The shifter is connected to the data input-output port and to the PSW. A shift amount latch selects data from either immediate latch, the B bus, or the byte controller.

Although a cross bar shifter could be arranged to do rotations, the implementation used here does not. Instead the wrap around capability is used in conjunction with sign extension circuitry to execute logical and arithmetic shifts. The bidirectional capability of the circuit is used to perform either left to right or right to left shifts.

Cross Bar Description

The cross bar in the RISC I shifter is a 63 by 32 array of pass transistors that connect two internal busses. These are the 32-bit R bus and 63-bit L bus. To shift data to the right: data is written onto the L bus, sign extended from bit 31 to bit 63, while data is read from the R bus. Writing onto the R bus and reading from bits 0-31 of the L bus will shift data to the left. The 2 transistor cross bar cell layout is shown below in figure 18.

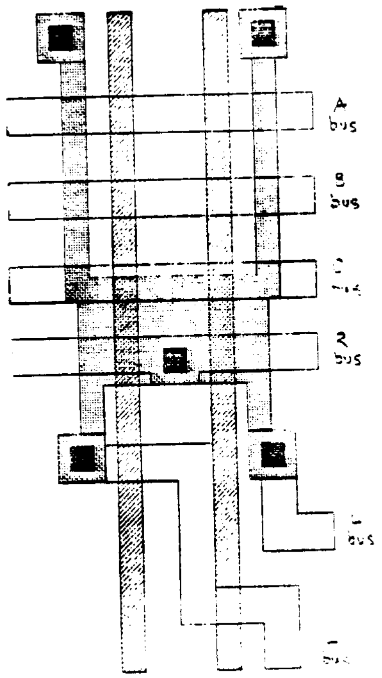


Figure 18: RISC I Cross Bar Cell

The two internal busses are precharged in phi1, and are used in phi2. Inverted data is used in the shifter so that precharging of the array can be used for the zero filling needed by logical shifts.

The R bus passes horizontally through the cross bar array, while the L bus passes diagonally through the array. The least significant 32 bits of each bus are connected to the shifter's input and output latches. The most significant 31 bits of the L bus are driven by the sign extension circuitry, which can extend the sign for arithmetic shifts or extend zeroes (ones on these inverted busses) for logical shifts.

Thirty two shift amount selection lines pass vertically through the cross bar. Each of these poly-silicon lines drives 32 gates of 32 cross bar pass transistors. Thrity two shift amounts, from 0 to 31, are possible. The A, B and C busses are implemented in metal here, and pass through the cross bar circuitry without connecting to it. The dataio port and PSW are connected to the L bus above the cross bar. The get and put PSW instructions actually are implemented as shift instructions where the PSW is either the data source or destination.

Shift amount latch

During a shift instruction the 5-bit shift amount latch receives input from either the immediate latch (for a shift by immediate), or the five least significant bits of the B bus (for a shift by registered variable). During load and store instructions the shift amount latch receives input from the byte controller. These latches use the same layout cell as the register address latches in the register file.

Shift amount decoder and Buffers

The five-bit shift amount number provides the input to a decoder which activates one of the thirty two cross bar control lines connecting the R and L busses. The direction and type of shift is stated by the instruction itself. Non-inverting super buffers (like those in the register file) are used to drive the cross bar control lines.

Shifter Input and Output latches

The 32-bit shifter input latch receives the data to be shifted from the A bus. The output of this latch can present inverted data to either one of the two busses of the cross bar array. The output of the cross bar is received by the 32-bit shifter output latch. It can be read out onto the C bus for storage in the destination register. Both of these latches contain scan-in scan-out circuitry for diagnostic reading and writing.

All control lines for the shifter and the rest of the data path modules are synchronized with the appropriate clocks for proper sequencing.

Data Input-Output Port

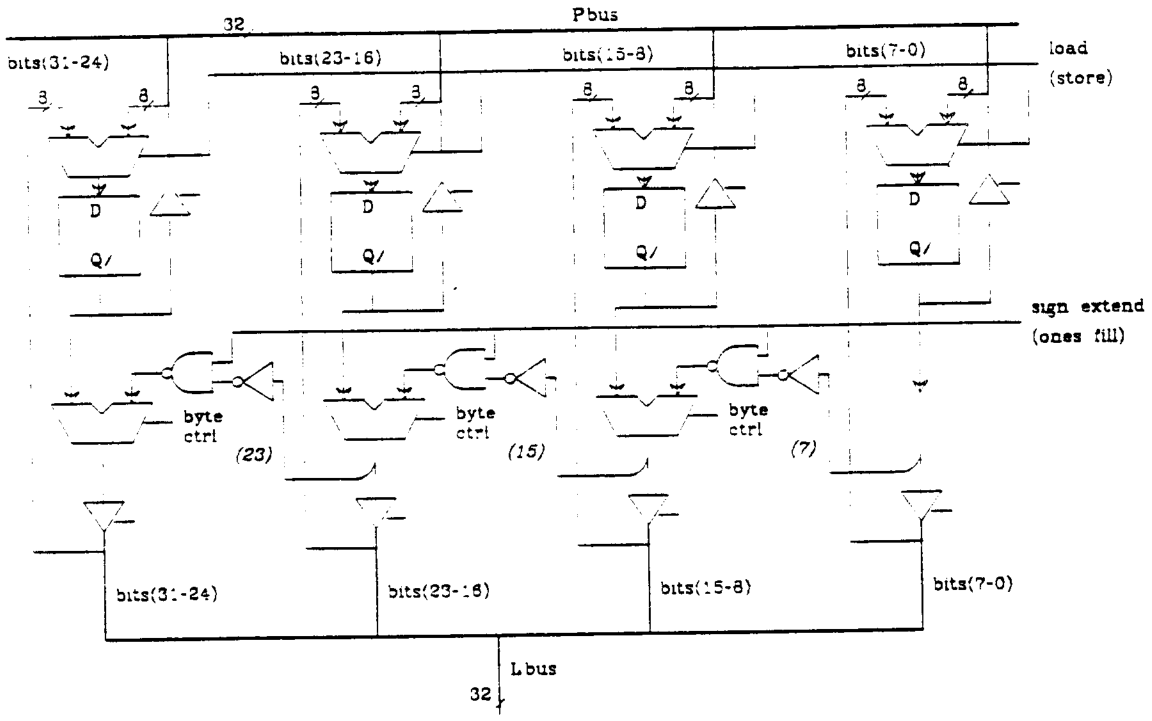


Figure 19: RISC I Data I/O Block Diagram

The data input-output port is shown in figure 19 above. It is used during loads and stores to transfer and sign extend data between the pad bus and the data path. This port is used to pass the byte or bytes of interest and sign extend to the upper bits. The shifter is used to shift the data down to the least signifcant place and sign extend further, if required.

The dataio port is designed as a bidirectional 32-bit latch with an internal byte and half word sign extender. One end of the this latch can read from and write to the L bus in the shifter. The other end of the latch can read from and write to the Pad bus. The data in this latch gets inverted to compensate for the inverted data required by the L bus of the shifter.

ALU

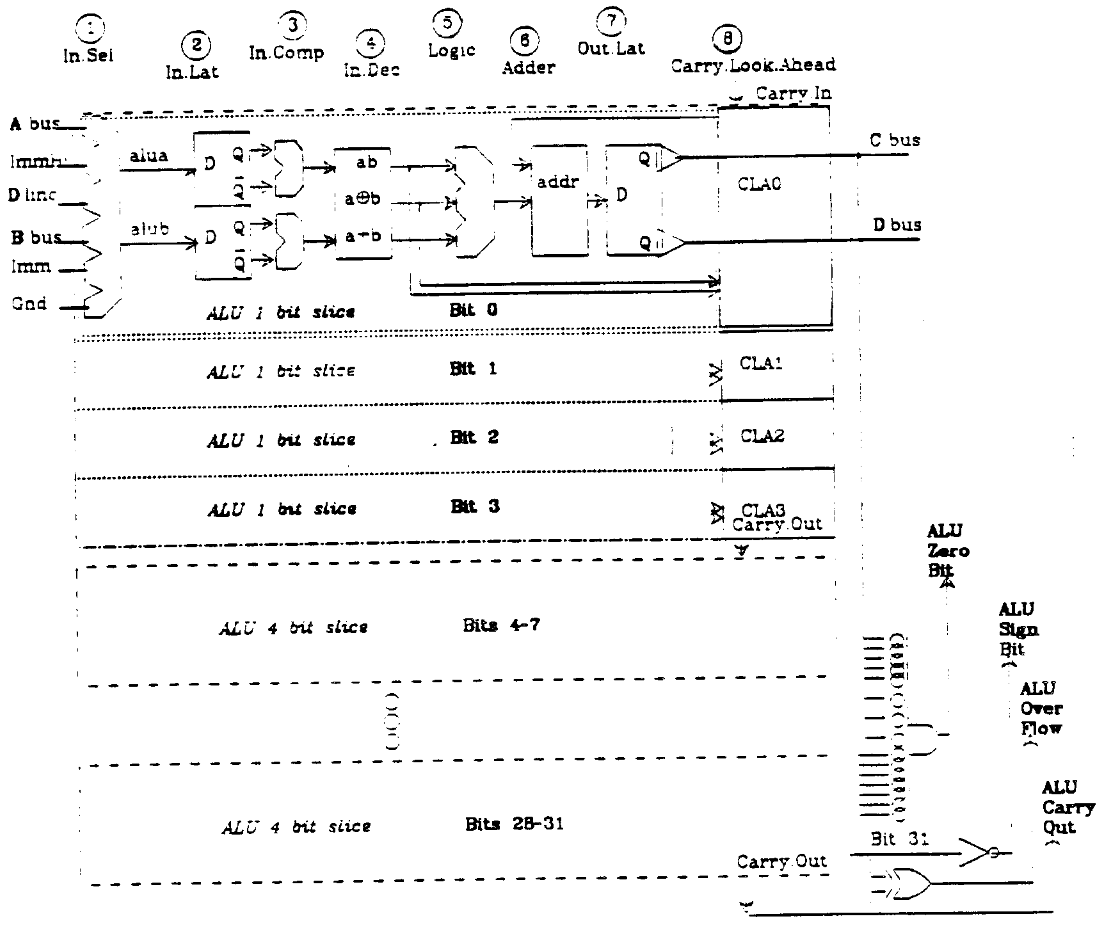


Figure 20: RISC I ALU Block Diagram

The block diagram of the RISC I arithmetic and logical unit is shown above in figure 20. It is based on a one bit slice, with a carry look ahead unit designed as a four bit slice. The ALU can perform 4 arithmetic and 4 logical operations. Eight separate circuits make up the ALU. These are the input mux, input latch, complementer, decoder, logic selector, adder, output latch, and carry look ahead unit.

Input Multiplexor.

The first circuit is the input multiplexor. It is used to select the ALU-A and ALU-B operands from 5 possible operands. The input selection is done with 7 pass transistors and 5 control lines.

The 4 ways of Loading the ALU Operands	
ALU-A	ALU-B
Source1 (from A bus)	Source2 (from B bus)
Source1 (from A bus)	13-bit immediate
PC (from D bus)	13-bit immediate
Zero (ground)	19-bit high immediate

The 8 RISC I ALU Operations	
Logical Operations	Arithmetic Operations
pass B	A plus B
A AND B	A plus B plus Carry in
A OR B	A minus B
A XOR B	A minus B minus Carry in

Input Latches

Two input latches hold the operands valid during clock phase 2 while the ALU computes its result. These latches contain scan-in scan-out circuitry.

Complementor and Decoder

The next two stages, the complementor and decoder compute the three logical functions of the two input operands and their inverses.

Logic Selector and Adder

The logic selector passes one of the three results of the previous block to the next stage, the adder.

The adder can either pass data directly to the output register or add with carry in. Addition is performed by computing the XNOR function of the output of the logic selector with the carry in for each bit ((A XNOR B) XNOR Cin).

Output latch and Carry Look ahead unit

The output register is used to hold the result until clock phase Phi3 when it can be read onto the C or D bus. This latch contains scan in scan out circuitry.

The carry look ahead unit receives propogate and generate signals directly from the logic decoder. This is a 4 bit circuit based on the logic implemented by the 74182 TTL chip. The control of the least significant carry in bit is handled by three logic gates located at the top of the ALU.

The ALU conditions: CNZV

The four usual conditions are generated by the RISC I ALU. The zero condition bit (Z) is computed by a 32 input negative input AND (NOR) gate connected to the C bus. The sign, or negative bit (N) is driven by bit 31 of the C bus. The carry bit (C) comes directly out of the last carry look ahead unit. The arithmetic overflow bit is generated by an XOR gate connected to the carry bit and sign bit.

Immediate Port

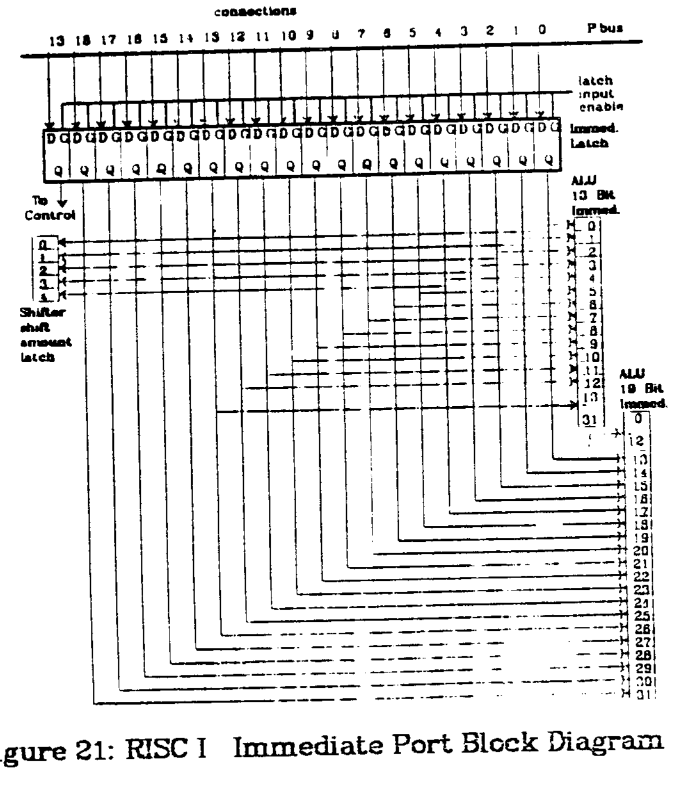


Figure 21: RISC I Immediate Port Block Diagram

The Immediate Port consists of the 19-bit immediate latch which can drive the shift amout latch and two separate inputs into the ALU. One input to the ALU connects the least significant 13 bits to the least significant 13 bits of the ALU, while connecting the 13th bit of the latch to the higher 19 bits. This connection passes the 13-bit RISC I sign extended immediates into the ALU. The 14th bit of the immediate register is the immediate flag for RISC I instructions and it us used by the control circuitry to decide whether to gate the 13-bit immediate or the second source register into the ALU. The other input of the ALU driven by the immediate latch has all 19 bits of the latch connected to the most significant 19 bits of that input, with the least significant 13 inputs grounded. This facilitates the 19-bit load-high-immediate RISC I instruction.

Program Counter

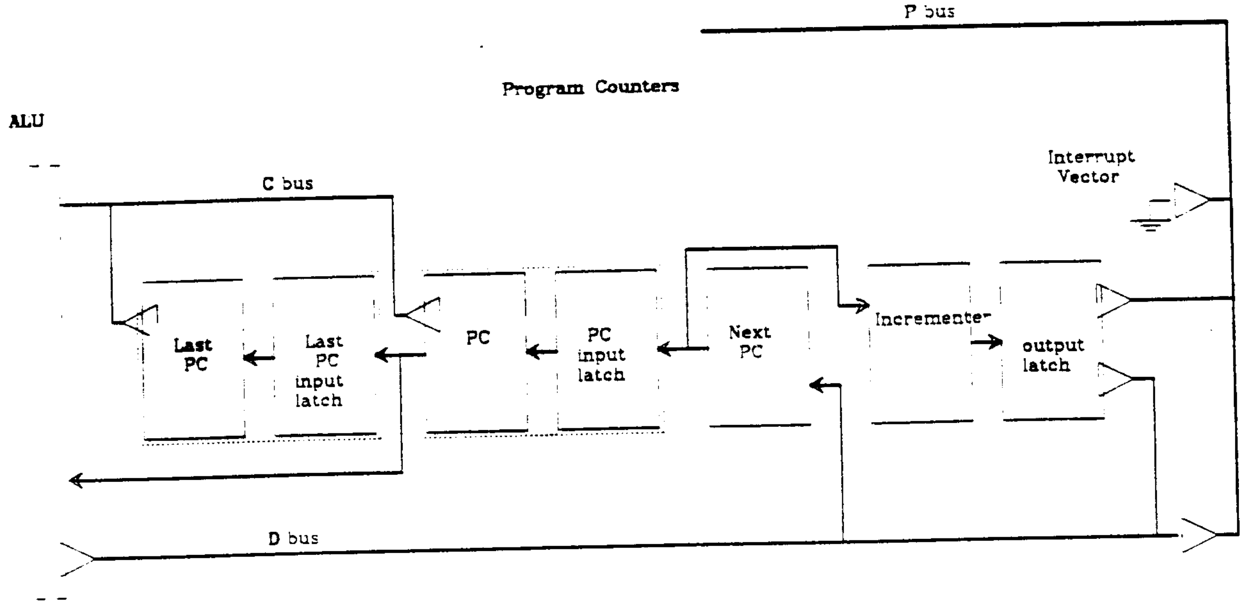


Figure 22: RISC I Program Counter Block Diagram

The program counter in RISC I is shown above in figure 22. It is implemented as a set of three program counter latches. Three instruction addresses (the PC, next PC, and last PC) are required for the operation of RISC I. The PC holds the address of the current instruction. The next PC holds the address of the instruction being fetched, and the last PC holds the address of the previous instruction so that it can be saved away after a call or an interrupt.

Two of these values (the PC and last PC) are loaded at the same time they are read out, so they are implemented as master-slave latches. This group of PC's consists of six 30-bit latches, one 30-bit incrementer, four 32-bit bus drivers, and a hard wired interrupt vector. The C and D busses pass through the PC and connect to it. The PC latch contains scan-in scan-out circuitry.

The circuitry was designed as 7 different one-bit cells. Like the rest of the data path, the least significant bit is on the top and the most significant bit is on the bottom. The 30 bits of the PC's are connected to bits 2-31 of the data path. The two least significant bits are connected to read out zeroes. This can be done because each RISC I instruction is 32 bits wide and is to be aligned across the four byte wide memory, so that the least significant two bits of each instruction's byte address are always 00.

The program counter performs the following tasks to maintain the proper flow of instruction addresses. The last PC latch is loaded from its input latch every Phi1n. During each Phi2n the next PC is passed to the PC input latch. The next PC is loaded from the D bus during each Phi3n and Phi3i. The data on the D bus comes from the incrementer for normal instruction flow; and the ALU for jumps. The incrementer operates from Phi3 to Phi1 and is valid by Phi2. It uses a ripple carry across all 30 bits. Immediately after call instructions and interrupts the last PC can be saved in the register file with the "get last pc" instruction. Return instructions restore the saved PC's via the D bus.

After the data path was designed it was simulated to check for design errors. The simulator "Slang" was written to do this and to simulate the operations that the control circuitry would perform. [Van Dyke 82b] The following 5 diagrams in figures 23 and 24 show the 30 control signals (plus clocks) which must be provided to the RISC I data path for it to work correctly.

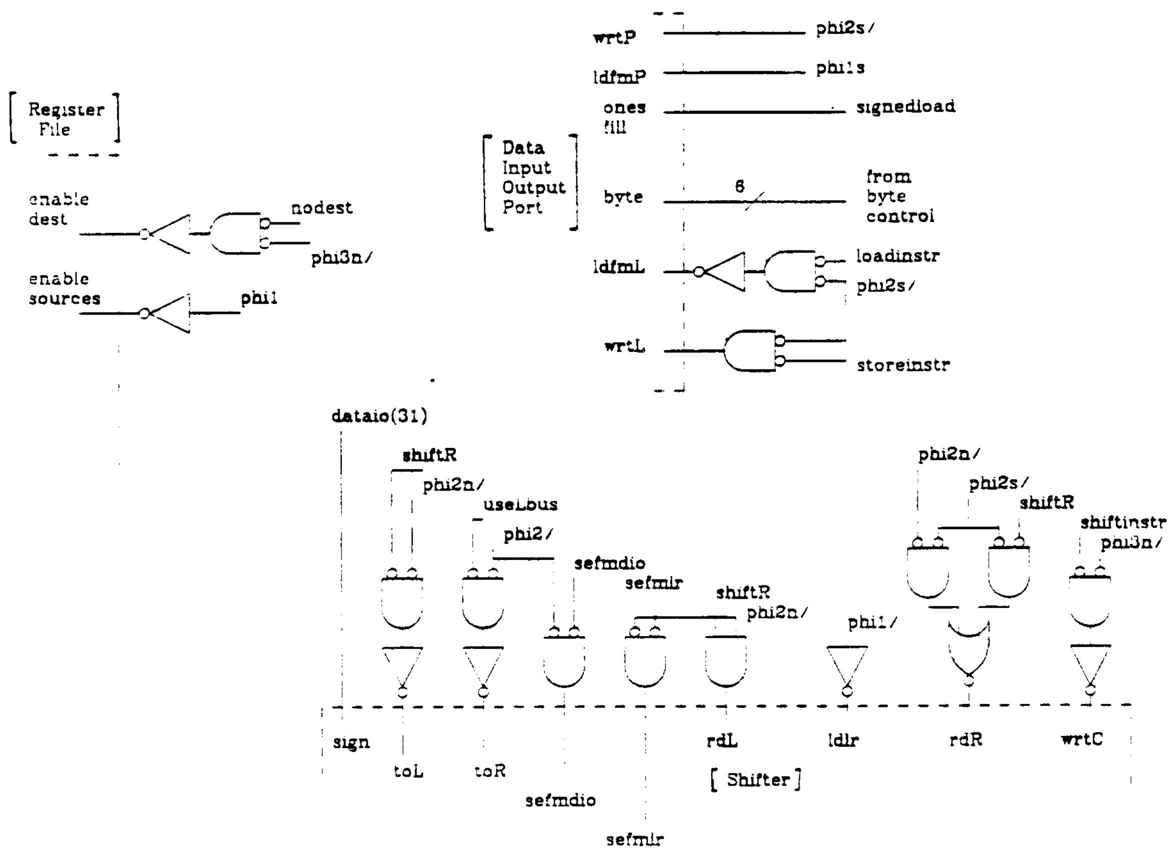


Figure 23: RISC I Regfile, Data I/O, Shifter Control Lines

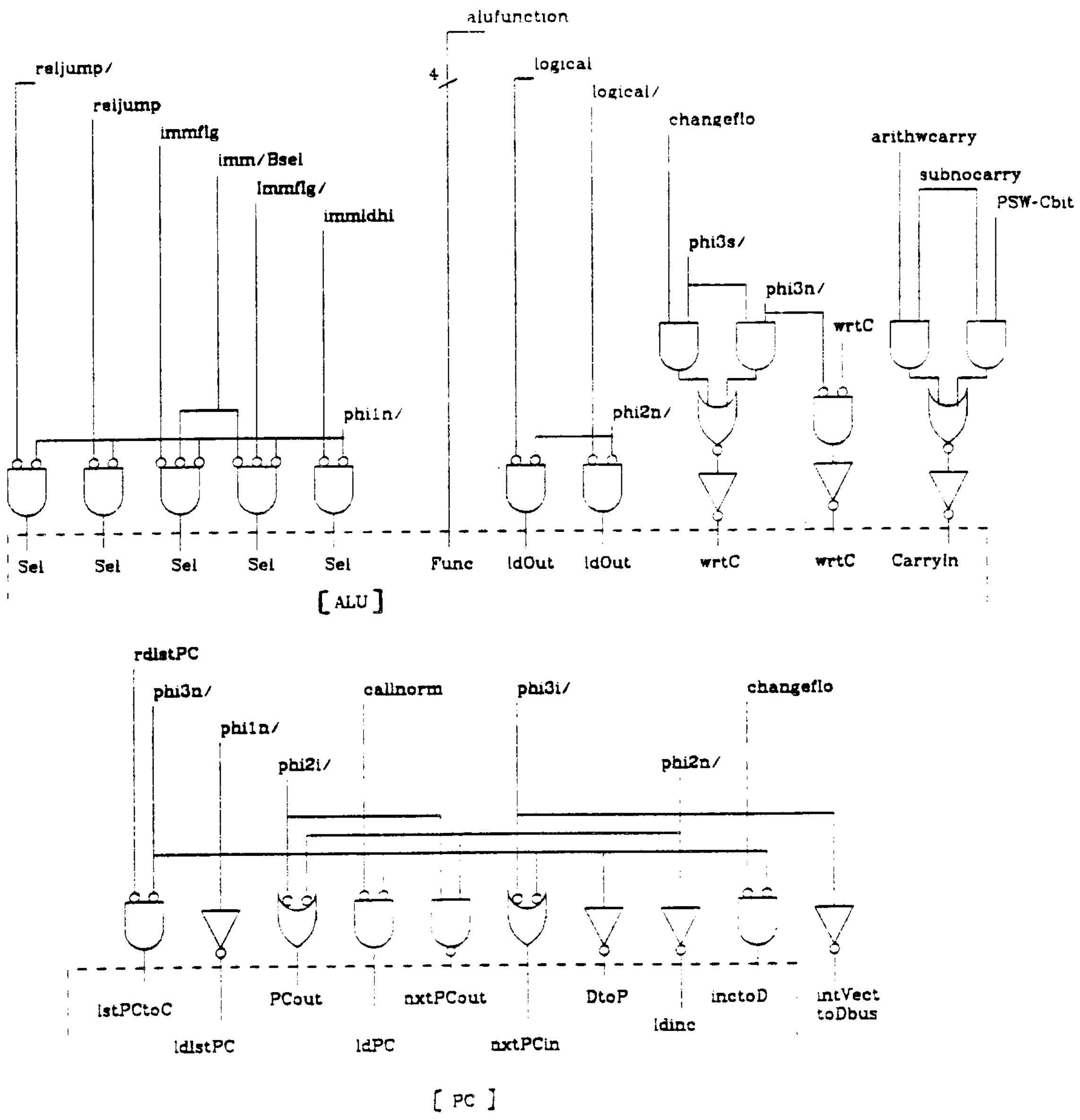


Figure 24: RISC I ALU, PC Control Lines

Our simulations of the data path uncovered design errors in each module which were easily corrected, once found. Slang provided the capability for interactive, high level functional simulations. After roughing out the control functions, the detailed logic simulator ESIM was run internally by Slang. This provided the designers with a way to zoom in on the problems and ignore the massive amounts of uninteresting data that were actually being processed. This was especially useful because at any one clock phase only about 100 nodes were actually doing anything interesting out of the more than 10,000 nodes simulated in the data path.

The ideas for the control were checked out in simulation at a high level before they were implemented in transistors and interconnections. Programmable logic arrays (PLA's) were used almost everywhere to perform the logic functions required by the controller. These PLA's were layed out automatically from logic equations by a set of programs built around the program "makepla". [Landman 82]

The RISC I controller was designed as 3 circuits. Together they contain approximately 1500 transistors. The primary circuit is the opcode controller. The window controller and byte controller are the other two circuits.

Opcode Controller

The opcode controller consists of 2 PLA's and some latches. It latches in the 7-bit operation code (opcode) of the next instruction and generates 31 signals in its pipeline latch which go to control the data path and the other two sections of the control circuiry. It is shown in figure 25 below.

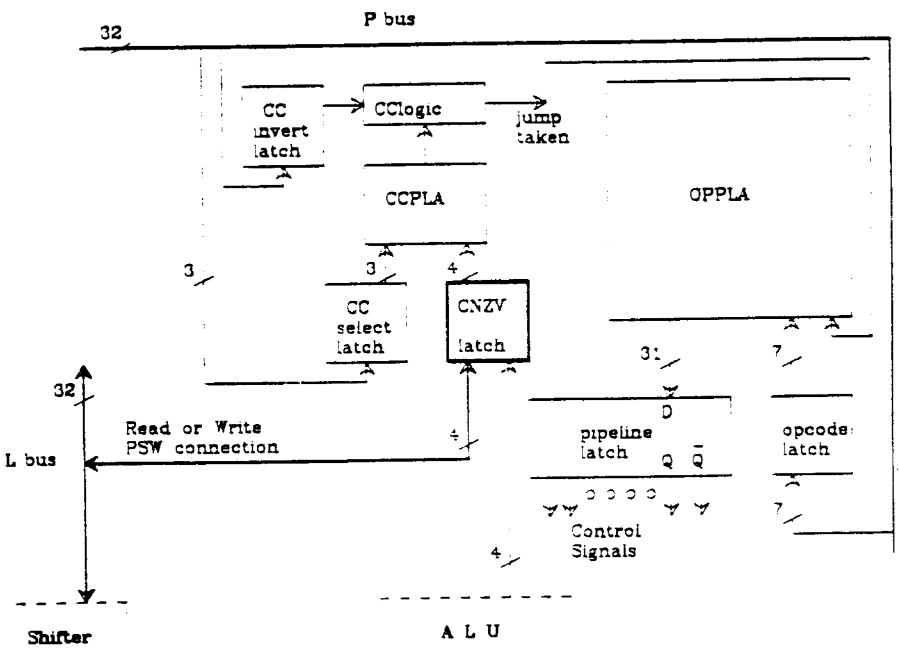


Figure 25: RISC I Opcode Controller Block Diagram

The basic organization is as follows. The opcode of an incoming instruction is latched into the instruction input latch from the P bus during clock phase

Phi2n. This latch drives the opcode decoding (OPPLA); the largest PLA on the chip.

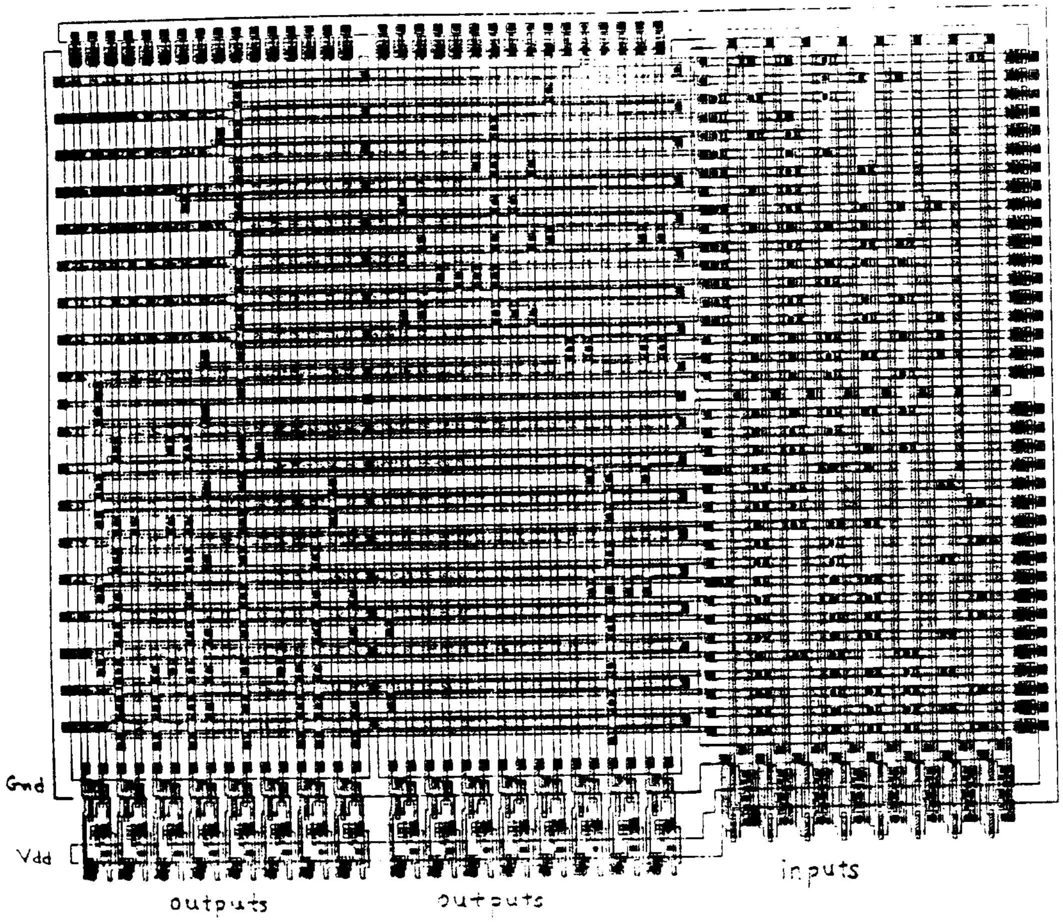


Figure 26: RISC I Opcode PLA Layout Plot

The opcode decoding PLA is shown above in figure 26. This PLA has 8 input terms, 7 from the instruction latch, and 1 from the condition code logic which decides if a jump is to occur. It contains 36 product terms, and drives 31 outputs.

The outputs of the OPPLA are latched into the pipeline latch either on Phi3n or Phi1n depending on when they are used. The signals that are needed for Phi1n or Phi2n of the execution cycle are latched in Phi3n of the decode cycle. Signals needed for Phi3n of the execution cycle are latched in Phi1n of the decode cycle.

Most of the outputs of the pipeline latch are synchronized with the 8 internal RISC I clock phases at the upper edge of the data path before driving the data path control lines. A few signals like the ALU function control lines do not need to be clocked.

The 31 bit control word latched into the pipeline latch is unique to each of the 39 RISC I instructions. The load and store instructions use the same control word for both the "normal" and "secondary" clock cycles of their execution.

The ALU condition bits can be latched into the CNZV latch following an ALU operation. During a conditional jump instruction the CNZV bits are tested by the condition code PLA to determine if the correct condition is valid. If the condition is valid the CC-logic circuit drives the jump taken bit into the OPPLA and the jump is taken.

Window Controller

The other two sections of the control circuitry are the window controller and the byte controller. The window controller responds to call and return instructions by changing the register file window number and checking for under and overflow. It is shown below in figure 27.

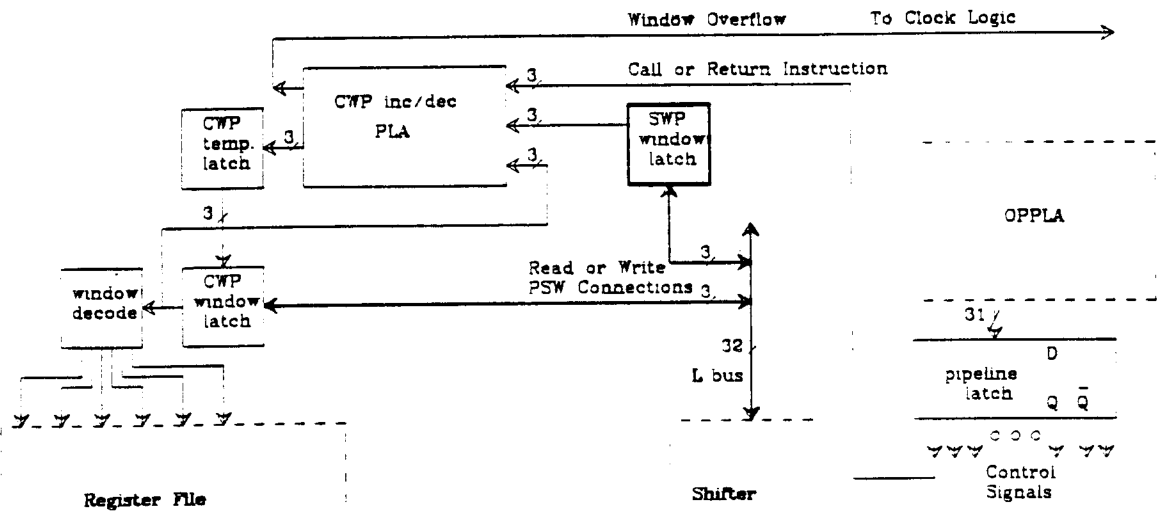


Figure 27: RISC I Window Controller Block Diagram

Two 3-bit pointers are used to manage the window file in RISC I. There is the "saved window pointer" (SWP) which always points to a window saved on chip to be used for interrupts and traps. This number is written by the processor as a part of the PSW and is not changed during calls and returns.

The other pointer is the "current window pointer" (CWP) which points to the window of registers that is currently being used. This number is incremented on returns and decremented on call instructions. A PLA is used to increment and decrement the CWP and to check for equality of the CWP and SWP. Incrementing and decrementing wrap around between 0 and 5 so that the six windows can be used effectively. If equality is detected then a window overflow trap is issued to the RISC I interrupt circuitry. When the processor recieves this trap it jumps to a special routine to take care of the situation.

Byte Controller

The byte controller responds to load and store instructions by signalling to the data input-output port to shift and sign extend its data. It is shown in figure 28 below.

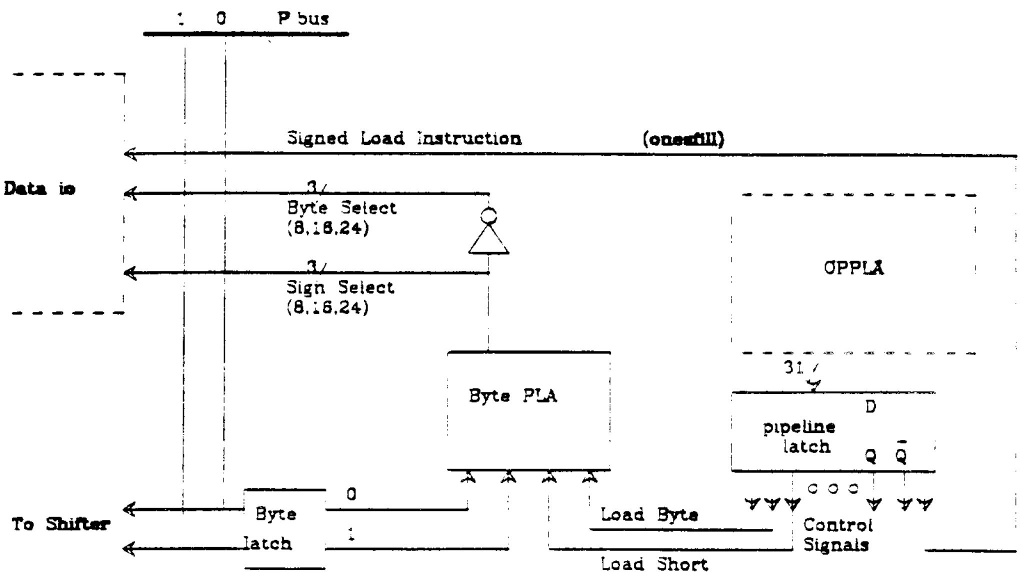


Figure 28: RISC I Byte Controller Block Diagram

The 2 least significant bits of the effective address are latched in to the byte latch. These are combined with two control bits from the pipeline latch, and these 4 bits together drive the byte PLA. The three outputs of this PLA and their inverses drive the byte selection circuitry of the dataio port.

The Processor Status Word

The processor status word, or PSW is part data path, part control unit. The way the bits in the PSW affect the control have been discussed, but it can also be written from and read into the general purpose registers with the put and get PSW instructions. The format for the RISC I PSW is shown below.

Processor Status Word Format			
XXX <31-10>	CWP <9-7>	SWP <6-4>	VZNC <3-0>

The last four bits (CNZV) come from the alu. They indicate the status of result of the alu operation. The next 6 bits are the 2 window pointers.

Read Write Wait Pads

The final circuits of RISC I to be described are the read-write-wait circuity and pads which are used to control the external memory. An 8 transistor circuit detects load and store instructions and signals the external memory if a write is required.

The sequencing of RISC I requires that data be valid from memory during phi1 of a load instruction. This only allows 1 phase for the external memory to operate in. All other instructions allow the memory to delay until phi2. To remedy this the "wait" pad is asserted during a load instruction. It can be used to request that the external clock circuitry wait one phase before issuing the next phi1.

Two other pads are used during store instructions to indicate whether a byte, short word, or long word is to be written. They are driven from bits 27 and 28 of the current instruction held by the opcode input latch.

RISC I Control in Operation

To better understand the operations of the internal circuits of RISC I, during its design, we created a table that shows the major actions that occur on the chip for each clock phase as it processes each of its instructions. This table is in Appendix A.

CHAPTER 5 Overview and Conclusions from RISC I

The chip level architecture and layout considerations of the RISC I microprocessor have been presented. The structured and regular design of the data path, clock and control circuits have been discussed. We now discuss some of the positive and negative aspects of the RISC I architecture and VLSI implementation.

One advantage of the RISC architecture is that the implementation time can be quite short. Fewer circuits need to be layed out and simulated than for a more complicated architecture. The performance of the RISC I processor has been tested. Even at 500 nano seconds per phase, it has been shown to be comparable with other microprocessors. [Foderero 82]

The control circuitry of RISC I was simple to design and understand. Most computers and microprocessors use a microcoded control circuit that executes an instruction by decoding a series of micro instructions requiring micro jumps and branches. The RISC I controller executes instructions by directly decoding the instruction and combining the bits of the decoded control word with the on-chip clock phases. RISC I executes its instructions quickly, providing the compiler with a small set of fast, orthogonal instructions.

Approximately 100 cells were used in the design of RISC I. No library of cells was available at the time RISC I was designed so circuits were designed and placed as needed by the various designers of each module. The ALU, shifter, and PC use three different latch designs, and three different bus driver circuits were used where only one of each was needed. Using a cell library design approach could allow the the number of circuits and the design time to be reduced, while increasing the flexibility and performance of the final chip.

CAD tools were essential throughout the design of RISC I. The layout editor, plotter, and simulators were the most essential of the tools. The functional simulator SLANG was used to be simulate, design and document the control circuitry. The layout extractor and rules checker were also heavily relied upon.

Two CAD tools that would have been very useful in the design of RISC I are a timing simulator (or analyzer) and a schematic editor for documentation.

Using programs to generate PLA's from logic equations was very worthwhile. The time required to design and modify them was reduced from days to minutes. They were simulated before being included mainly to check the validity of the logic equations rather than the layout.

Performance analysis of RISC I was carried out manually. We first extracted some of the known critical paths and simulated them on SPICE. Later this method was abandoned and instead only rough hand calculations of capacitance charging were analyzed which consumed much less of the designer's time. A timing analyzer such as described by Ousterhout [Ousterhout 83] would have been very useful. As it was some critical paths were missed and the chips run at about one half of the speed hoped for.

Primary emphasis was placed on acheiving functional correctness at first silicon. Accomplishing this required full scale software simulations with some detailed hand analysis. Some errors, like forgetting to refresh a latch, would not normally be caught by a simulator and needed to be checked manually.

Testing the chips proved to be a complicated problem. The scan-in scan-out circuitry was found to be incompatible with the capabilities of the testing hardware. After many frustrating hours this circuitry was ignored, and full scale CPU operations were attempted on the chips with good results. A few chips actually ran all the original diagnostic programs and new programs were written. In running these new programs the only known design bug (so far) was found. The shifter was unable to set the sign bit in the PSW as had originally been desired. Luckily the compiler could be changed to take the problem into account: using the ALU to set the sign bit if so needed. Perhaps a fault coverage simulator (another CAD tool) could have helped us write more comprehensive diagnostics programs to catch such errors.

Some Ideas for Future RISC's

Many good ideas were utilized in the design of the RISC I chip, but there are many ways of improving it. Designing a chip with increased functionality is one way for improvement; designing for faster throughput is another. Due to yield limitations of large chips like RISC I, designing for a smaller die size with the same functionality and same through put would be a different path for improvement (RISC I measures 10mm X 7.8mm). A second generation RISC has recently been designed at Berkeley [Sherburne 82]. The RISC II design incorporates ideas that could lead to a faster and smaller chip. The following list includes some ideas which could improve the design of RISC I in one or more ways.

[1] Use a six transistor register cell as done in the RISC II design.

[2] Fully overlapping the registers in the windows allows the compiler to make maximum use the number of registers in the regfile.

[3] Designing the control circuitry as a rectangular module would require a little more area, but its layout would be much simpler. Figure 29 shows a block diagram for a control module.

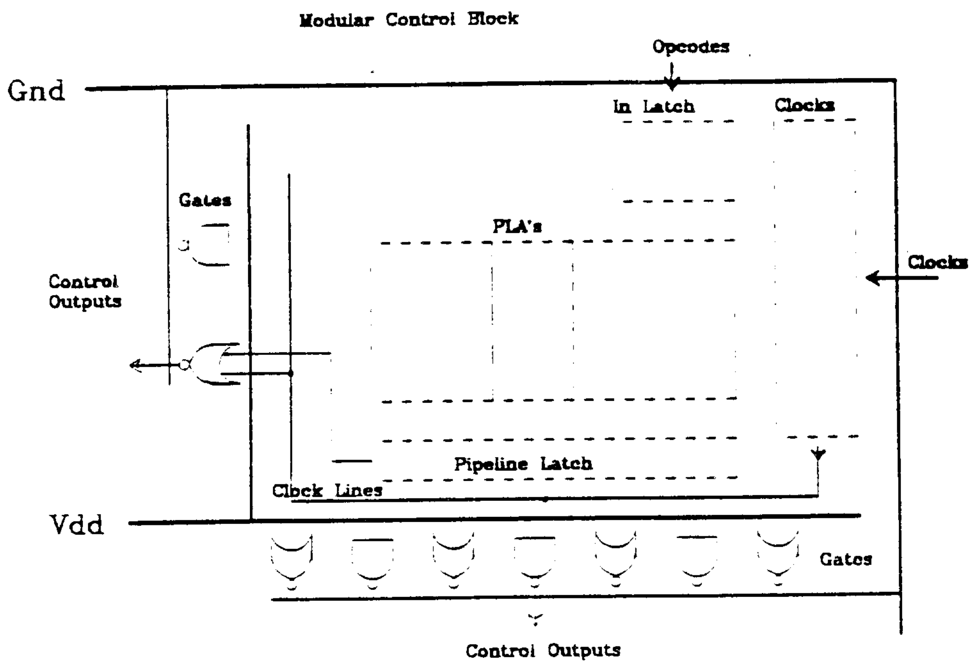


Figure 29: Proposed Modular Control Block for a RISC

The clock circuitry lines and gates could be included in the control module minimizing clock skews and simplifying clock routing.

[4] The PSW could also be included in the control module with a bus connecting it to the data path, independent of the shifter.

[5] A ROM could be used as a smaller and faster instruction decoder.

[6] The ALU could be designed with a smaller and faster carry look ahead circuit.

[7] The PC could be designed with a multiplexer so as not to require a bus to select the next address.

[8] The bus connecting to the on chip pad drivers could be designed as 2 busses that would be faster than the one tri-state bus used on RISC I. This is done on RISC II.

[9] The immediates could be loaded into the data path through the dataio port removing the need for the separate immediate latch on RISC I. This also is done on RISC II.

[10] The data and address busses could use separate pins of a large chip carrier package to minimize the external support circuitry required in the design of a system using the chip.

[11] Designing the sequencing around a 2 or 4 phase clock could allow the external clock circuit to be simpler. This could also provide better load and store timing.

[12] Some hand shaking logic could be added to the on chip clocks to allow it to more easily access data across slow external memory busses.

[13] Better on chip diagnostic circuitry would be a very good idea. A special parallel access to the controller and data path via the I/O pads would be useful.

[14] In order to make the circuits more useful to other designers working on the same or later projects, fully characterizing and documenting them is very important. This will be more work initially, but should pay off with less work and hassle soon after.

Acknowledgements

This project was made possible by the help of many people, both directly and indirectly. Professors David Patterson and Carlo Sequin originated the idea and provided guidance and a frame work for the project. I thank the students of the original 292R class who developed the architecture that became RISC I, and especially Manolis Katevenis who did the initial work towards implementing that architecture in silicon.

I would like to thank David Patterson, John Foderero, Korbin Van Dyke, Dan Fitzpatrick, Zvi Peshkess, and Howard Landman for their parts in the design of RISC I. I would also like to thank Robert Sherburne and Manolis Katevenis for their ideas and their help.

For the CAD tools which made the design of RISC I a possiblity, I thank John Ousterhout, John Foderero, Dan Fitzpatrick, Robert Cmelik, Ken Keller, Howard Landman, and Mike Arnold. I would like to thank the people at MOSIS and Xerox PARC, for fabricating the circuits.

I especially thank Dr. Patterson and Dr. Ousterhout for their reading and advice on this paper. This work was funded by ARPA, whose support is gratefully acknowledged.

References

[Baker 80]

Baker, C.M., "Artwork Analysis Tools for VLSI Circuirts," TR-239, Lab. for Comp. Science, M.I.T. (May 1980).

[Corcoran 80]

Corcoran, G.T., "Using ISPS to Specify the RISC I Architecture and Implementation," Master's Report, U.C.Berkeley (Dec. 1980)

[Fitzpatrick 81]

Fitzpatrick, D.T., Foderero, J.K., Katevenis, M.G.H., Landman, H.A., Patterson, D.A., Peek, J.B., Peshkess, Z., Sequin, C.H., Sherburne, R.W., and Van Dyke, K.S., "VLSI Implementation of a Reduced Instruction Set Computer," *Proc. CMU Conf. on VLSI Systems and Computations*, Pittsburgh PA, pp. 327-336 (Oct. 1981).

[Foderero 81]

Foderero, J.K. and Van Dyke, K.S., "SLANG Slinger's Cyclopedia," Internal Report, U.C.Berkeley (Dec. 1981).

[Foderero 82]

Foderero, J.K., Van Dyke, K.S., and Patterson, D.A., "Running RISC's," *VLSI Design* III(5) pp. 27-32 (Sept. 1982).

[Halbert 80]

Halbert, D. and Kessler, P. *Windows of Overlapping Registers*, CS292R Final Reports (June 1980).

[Katevenis 80]

Katevenis, M.G.H., "A Proposal for the LSI Implementation of the RISC I CPU," Internal Report, U.C.Berkeley (Sept. 1981).

[Landman 82]

Landman, H.A., "Automatic Layout of Optimized PLA Structures," Master's Report, U.C.Berkeley (1982).

[Nagel 75]

Nagel, L.W., "SPICE2: A Computer Program to Simulate Semiconductor Circuits," ERL Memo ERL-M520 U.C.Berkeley (May 1975).

[Ousterhout 82]

Ousterhout, J., "Caesar: An Interactive Editor for VLSI Circuits," *VLSI Design* II(4) pp. 34-38 (Nov. 1981).

[Ousterhout 83]

Ousterhout, J., "Crystal: A Timing Analyzer for nMOS VLSI Circuits," *Proc. of the 3rd Caltech Conf. on VLSI* (Mar. 1983).

[Patterson 80]

Patterson D.A., Sequin C.H., "The Case for the Reduced Instruction Set Computer," *Computer Architecture News* 8(6) pp. 25-33 (Oct. 1980).

[Patterson 81]

Patterson D.A., Sequin C.H., "RISC I: A Reduced Instruction Set VLSI Computer," *Proc. 8th Intnl. Symp. on Computer Arch.*, Minneapolis MINN, pp. 443-457 (May 1981).

[Patterson 82]

Patterson D.A., Sequin C.H., "A VLSI RISC," *Computer 15* (9) pp. 8-21 (Sept. 1982).

[Sequin 82]

Sequin C.H., Patterson D.A., "Design and Implementation of RISC I," U.C.Berkeley, Report No. UCB/CSD 82/106 (Oct. 1982).

[Sherburne 82]

Sherburne, D.A., Katevenis, M.G.H., Patterson D.A., Sequin C.H., "Datapath Design for RISC," *Proc. Conf. on Adv. Research in VLSI*, M.I.T., Cambridge, MA, pp. 53-62 (Jan. 1982).

[Tamir 81]

Tamir Y., "Simulation and Performance of the RISC Architecture," ERL Memo M81/17 U.C.Berkeley (Mar. 1981).

[Terman 82]

Terman, C., "Simulation Tools for VLSI Design," Thesis, Lab. for Comp. Science, M.I.T. (1982).

[Van Dyke 82a]

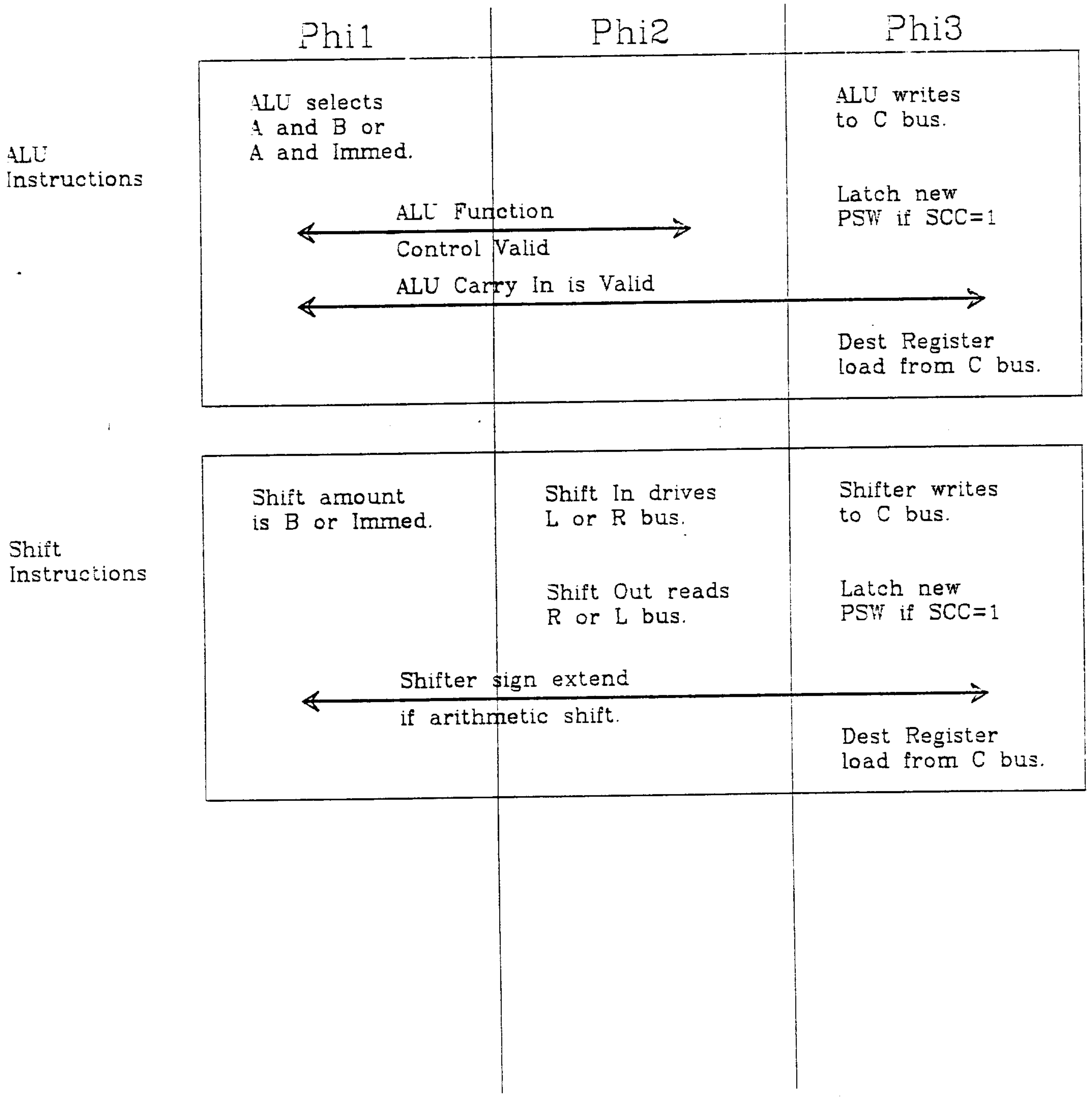
Van Dyke, K.S., *The Reduced Instruction Set Computer Single Board Computer Specification*, CS252A/B Final Report (Spring 1982).

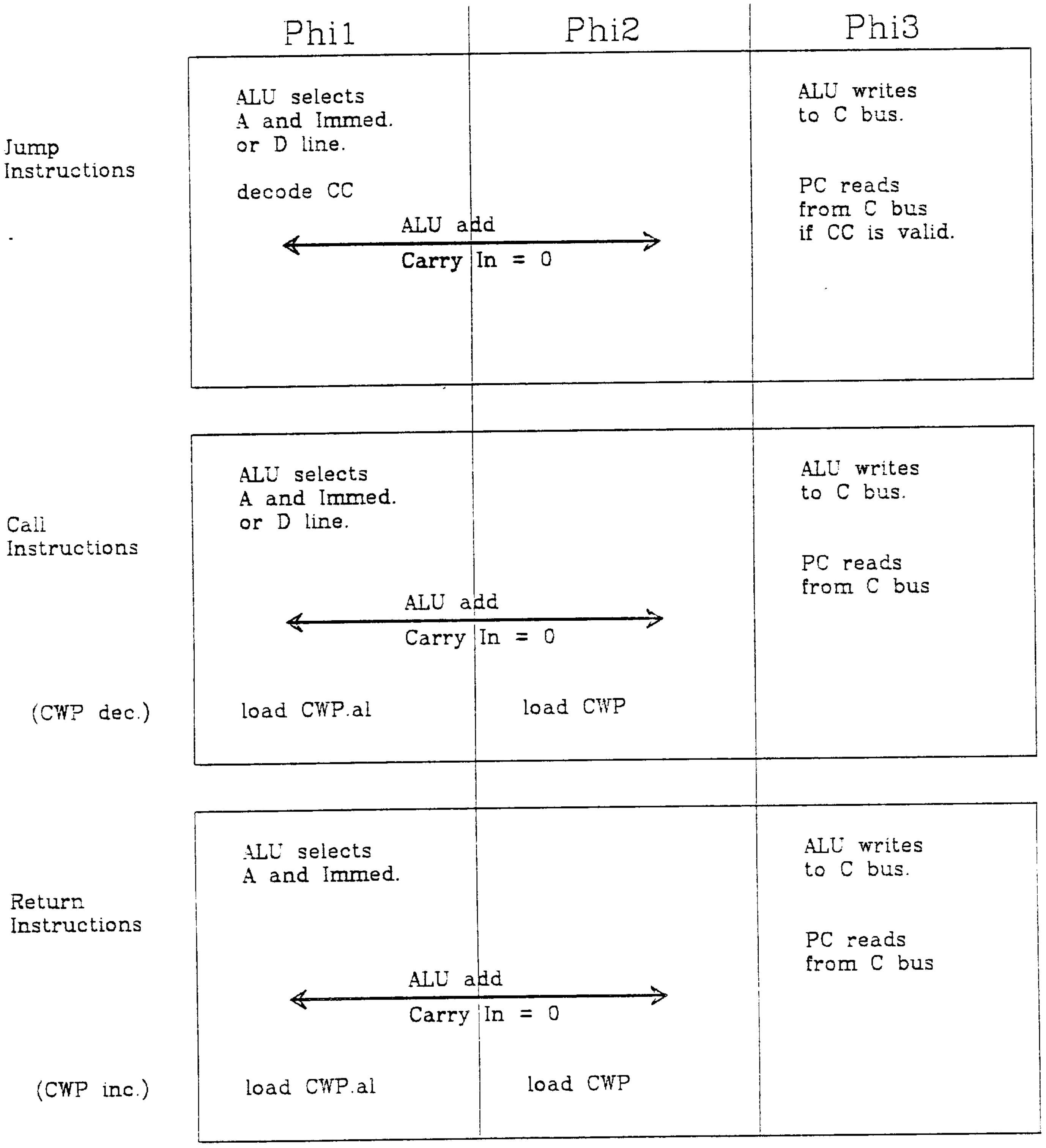
[Van Dyke 82b]

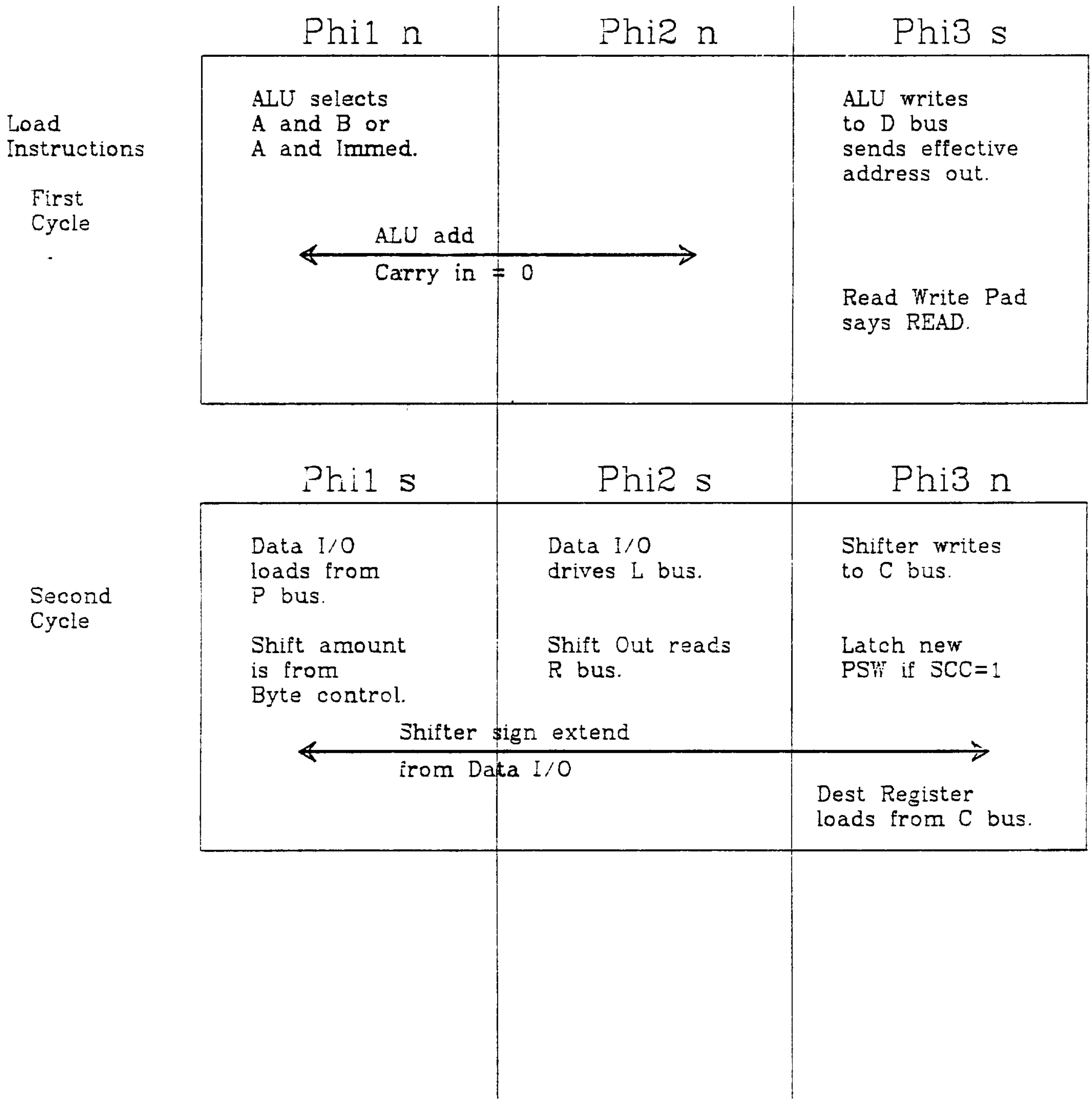
Van Dyke, K.S., "SLANG: A Logic Simulation Language," Master's Report, U.C.Berkeley (June 1982).

Appedix A RISC I On Chip Activity

	Phi1	Phi2	Phi3
Basic Cycle all Instructions		Precharge D bus.	Dbus gets PC or ALU. Dbus drives D line. Read Write Pad says READ.
	← Instruction Returned from Mem. →	Load all Instruction latches from P bus.	Precharge A and B busses Decode Instruction
	Precharge L and R busses. Read A and B registers Load Shifter input latch from A bus.	Precharge C bus.	

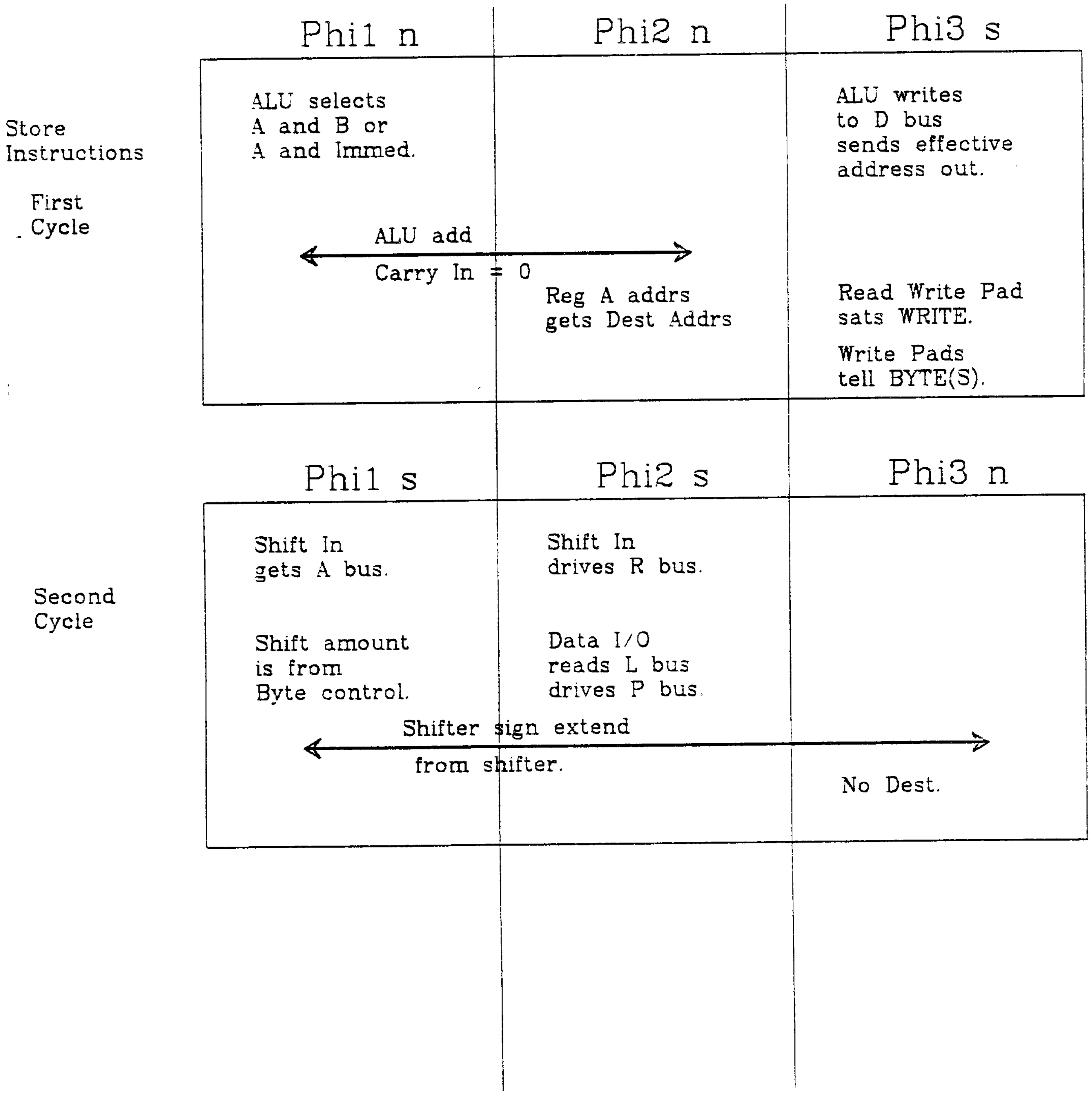

Phi1
Phi2
Phi3
ALU Instructions
ALU selects A and B or A and Immed.
ALU writes to C bus.
ALU Function Control Valid
Latch new PSW if SCC=1
ALU Carry In is Valid
Dest Register load from C bus.
Shift Instructions
Shift amount is B or Immed.
Shift In drives L or R bus.
Shifter writes to C bus.
Shift Out reads R or L bus.
Latch new PSW if SCC=1
Shifter sign extend if arithmetic shift.
Dest Register load from C bus.

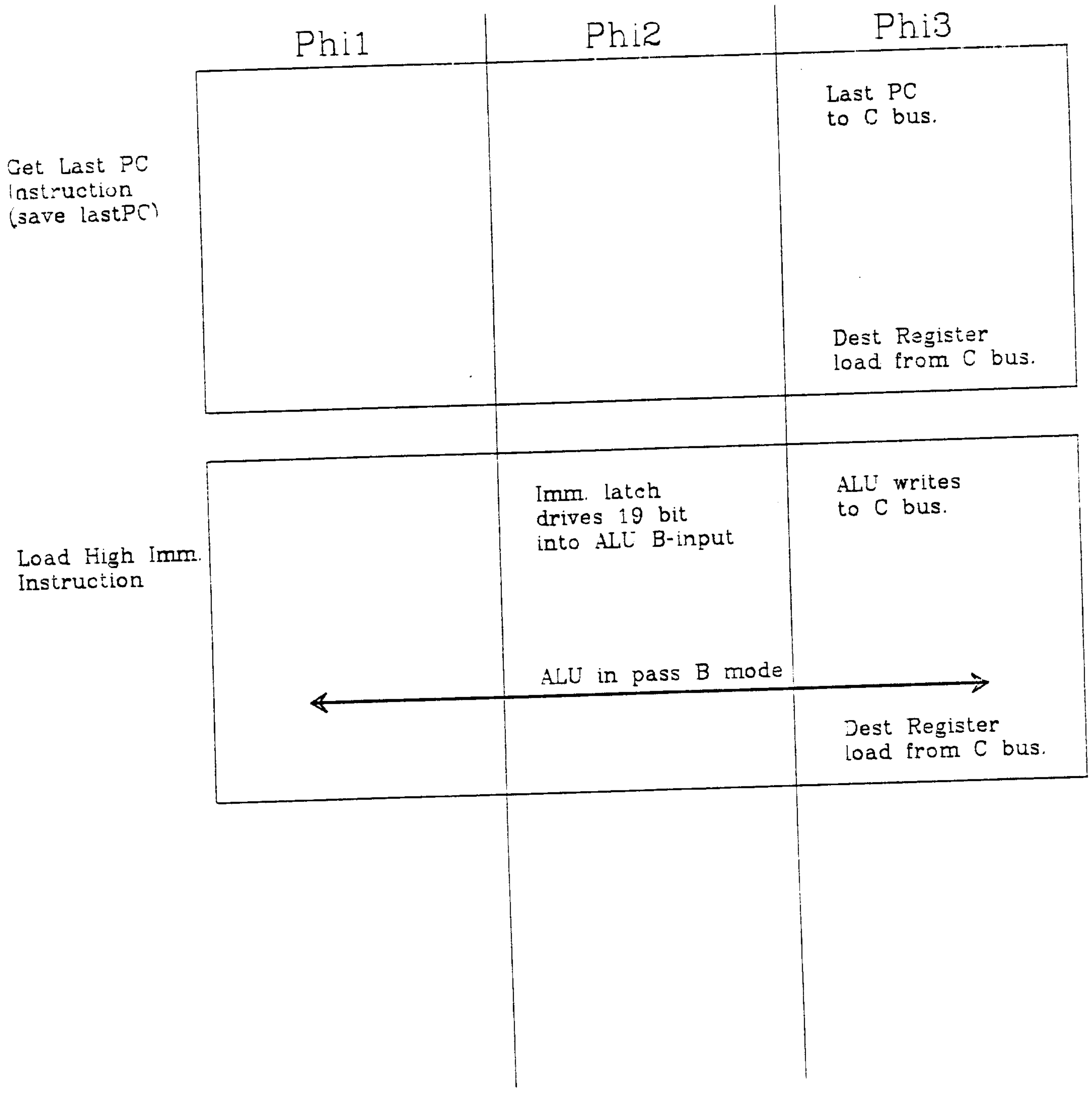

Phi1
Phi2
Phi3
Jump Instructions
ALU selects A and Immed. or D line.
decode CC
ALU add
Carry In = 0
ALU writes to C bus.
PC reads from C bus if CC is valid.
Call Instructions
(CWP dec.)
ALU selects A and Immed. or D line.
ALU add
Carry In = 0
load CWP.al
load CWP
ALU writes to C bus.
PC reads from C bus
Return Instructions
(CWP inc.)
ALU selects A and Immed.
ALU add
Carry In = 0
load CWP.al
load CWP
ALU writes to C bus.
PC reads from C bus



	Phi1 n	Phi2 n	Phi3 s
Load Instructions First Cycle	ALU selects A and B or A and Immed. ALU add ⟷ Carry in = 0		ALU writes to D bus sends effective address out. Read Write Pad says READ.

	Phi1 s	Phi2 s	Phi3 n
Second Cycle	Data I/O loads from P bus. Shift amount is from Byte control. Shifter sign extend ⟷ from Data I/O	Data I/O drives L bus. Shift Out reads R bus.	Shifter writes to C bus. Latch new PSW if SCC=1 Dest Register loads from C bus.


Store Instructions
First Cycle
Phi1 n
Phi2 n
Phi3 s
ALU selects A and B or A and Immed.
ALU add
Carry In = 0
Reg A addrs gets Dest Addrs
ALU writes to D bus sends effective address out.
Read Write Pad sats WRITE.
Write Pads tell BYTE(S).
Second Cycle
Phi1 s
Phi2 s
Phi3 n
Shift In gets A bus.
Shift amount is from Byte control.
Shift In drives R bus.
Data I/O reads L bus drives P bus.
Shifter sign extend
from shifter.
No Dest.


Phi1
Phi2
Phi3
Get Last PC Instruction (save lastPC)
Last PC to C bus.
Dest Register load from C bus.
Load High Imm. Instruction
Imm. latch drives 19 bit into ALU B-input
ALU writes to C bus.
ALU in pass B mode
Dest Register load from C bus.