A Minimal TTL Processor for Architecture Exploration

Bradford J. Rodriguez
McMaster University

Keywords: CPU, processors, architecture, education

Computer architecture is presently taught "hands-on" only when adequate VLSI design tools are available. The PISC is a processor constructed from discrete TTL logic, which illustrates the operation of both hardwired and microcoded CPUs. An efficient stack machine is easy to implement, and simple hardware modifications demonstrate interrupts, memory segmentation, microsequencers, parallelism, and pipelining. A standalone PISC board should be an economical and effective tool for teaching processor design.

Introduction

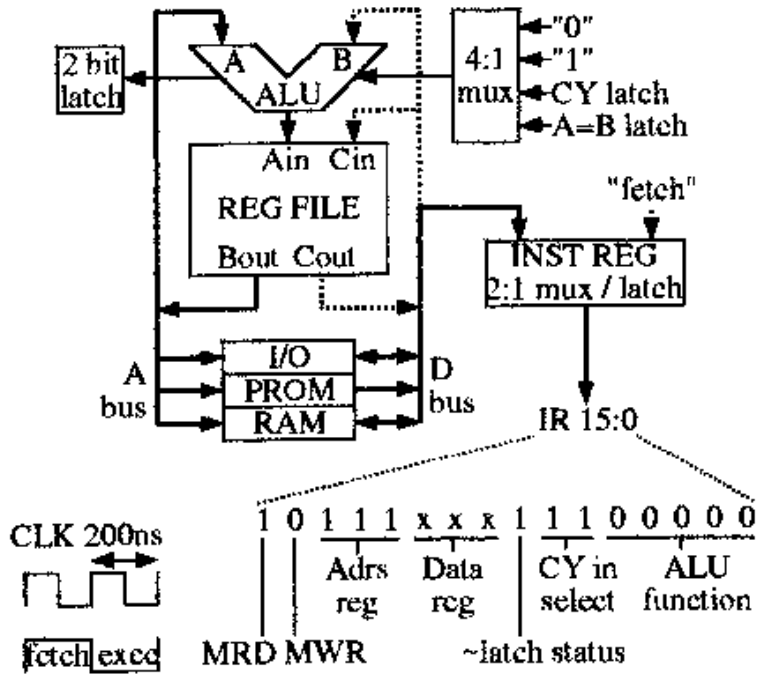
The study of computer architecture is often an abstract, paper exercise. Students cannot probe the inner workings of a single-chip microprocessor, and few discrete-logic machines are open to student inspection. Only universities that have VLSI design tools can give hands-on experience in processor design and implementation. Less fortunate institutions can only offer their students "book learning."

The Pathetic Instruction Set Computer is a model processor constructed entirely of discrete logic, illustrating the principles of both hardwired and microprogrammed CPUs. Requiring only 22 standard TTL chips (excluding memory), it is well within the ability of a student to construct and understand. Its writeable microprogram store uses inexpensive EPROM and RAM. Being fully static, it can be run at slow clock speeds or manually single-stepped for observation. Simple extensions demonstrate interrupts, split instruction and data spaces, microsequencers, parallelism, and pipelining.

The Basic Processor

The PISC-1a processor (Fig. 1) is designed to achieve a maximum of functionality with a minimum of logic. It is remarkable for having only 16 internal control signals, and thus a horizontal (unencoded) microinstruction only 16 bits long!

Fig. 1 The Basic PISC-1a



The ALU comprises four 74181s, which can perform the arithmetic operations of add, subtract, increment, and decrement, plus all logical operations, on 16-bit numbers. A programmable status latch and a 4-way multiplexer for the carry input complete the ALU logic. Eight 74172s provide eight 16-bit registers in a three-port register file. This file may simultaneously write one register ("A"), read a second ("B"), and read or write a third ("C"). In a single clock cycle, the following occurs:

a) one register is output to the Address bus and the ALU's A input;
b1) another register may be output to the Data bus and the ALU's B input; or
b2) data from memory may be input to another register;
c) an ALU function is applied to A (and perhaps B) and the result is stored in the first (address) register.

There is no dedicated microsequencer; its functions are performed by the ALU and register file. Every microinstruction has two phases: fetch and execute. During the fetch phase (illustrated in Fig. 1) a hardwired "pseudo-instruction" is executed:

a) output R7 (the program counter) to the Address bus and the ALU's A input;
b) read data from memory;
c) apply the function A+1 and store the result back in R7 (at the trailing edge of the clock).

Dedicated logic causes the memory data to be stored in the Instruction Register (IR) rather than in the register file. During the execute phase, this instruction is performed and the fetch instruction is reloaded into the IR. Thus every microinstruction requires two clock cycles. The control logic for this (not shown) involves only two flip-flops and a NAND gate.

Microprogram store and main program store are one and the same. Indeed, the PISC has characteristics of both a hardwired CPU and a microcoded CPU.

PISC as a Hardwired CPU

The PISC may be viewed as a conventional CPU with a hardwired control unit, a register-register architecture [3], and a badly encoded instruction set vaguely reminiscent of the PDP-11.

ALU operations are one- or two-operand, and include register-register move, add, add with carry, subtract, subtract with borrow, increment, decrement, left shift, and all logical functions. There is no multiply, divide, or right shift.

Memory operations are load and store, and have three addressing modes: register indirect, register indirect with postincrement, or register indirect with postdecrement. Postincrement addressing on R7 (the program counter) yields immediate addressing.

Control instructions are ALU operations on the program counter. Register indirect jump (absolute or relative) and conditional skip can be done in a single instruction. Other jumps and branches, and subroutine call/return, must be explicitly coded.

PISC as a Microprogrammed CPU

The PISC instruction word actuates physical control signals, and one can view the PISC as a microprogrammed CPU. The basic PISC implements a conventional machine inefficiently, since it lacks the logic to separate and decode the fields of a macroinstruction. But the PISC excels as a zero-operand architecture, i.e., a stack machine.

The fastest implementation uses threaded code [1] and a 16-bit macroinstruction. This reduces the microinterpreter to one microinstruction:

```
MRD PC,IP,A+1 ; mem(IP)->PC, IP+1->IP
```

One register is dedicated as the macro Instruction Pointer, and one or two others as stack pointers. Some flexibility can be gained by adding one microinstruction to the interpreter to use indirect threaded code [2].

Logically, the microprogram store should be separate from the macroprogram memory. But a unified macro- and micro-program store allows the programmer to write microcode and add new macroinstructions -- a valuable educational tool. Such an "extensible instruction set" in a stack machine is evocative of the programming language Forth, and this concept has been anticipated by several dedicated Forth processors [4].

Glaring Deficiencies

Many weaknesses of the PISC become evident after a short period of use, including:

a) no conditional branch microinstruction -- an important need [6];
b) no provision for literal values in the microinstruction;
c) no ALU logic for multiply, divide, and right shift;
d) no logic for decoding of macroinstructions;
e) no provision for interrupts;
f) sparse coding of the ALU function select; and
g) two clocks required per microinstruction.

It can be argued that the PISC is a valuable educational tool because these faults, and several potential solutions, are painfully obvious. Some faults cannot be rectified without substantial added logic, or an expansion of the microinstruction word. But several improvements are trivial.

ALU Operation Decoding

Seven bits of the 16-bit microinstruction (including the carry input select) select the ALU function. Fewer than 32 of the 128 codes are actually useful, however. A "nanocode" memory [3], or combinational logic, could reduce the ALU field(s) from seven to five bits.

Alternatively, unused function codes can be decoded to generate auxiliary control signals. For example, the carry input select (IR6:5) is a "don't-care" for logic operations (IR4=1). Thus 48 of the 64 logic function codes can be used for other purposes. A single 74138 can

provide eight supplementary control signals, such as interrupt enable and disable. Additional logic can prevent the ALU output from being written to a register, if desired.

Conditional ALU Operations

A simple conditional microinstruction modifies the ALU function according to the carry status. Two modifications are useful:

a) if carry set, change ALU operation from "A" to "B" (conditional jump)
b) if carry set, change ALU operation from "A" to "A+B" (conditional branch)

Unused ALU functions can be decoded for these operations, and suitable logic added to alter the ALU function select inputs. This does, however, add delay to the critical timing path of the CPU.

Interrupts

Two kinds of interrupts can be added easily to the PISC. Microinterrupts can be implemented by having the interrupt set a flip-flop, changing the program counter in the fetch pseudo- instruction from R7 to R6. R6 must be dedicated as an interrupt service register, and an additional control signal must be decoded to reset the flip-flop. (The RCA 1802 used a similar interrupt scheme.)

Macrointerrupts can be recognized by the microinterpreter by adding a conditional skip and an interrupt service routine to the microinterpreter. The interrupt input is connected to the carry input multiplexer, replacing the little-used A=B input. The interpreter is slowed by only one microinstruction.

In either case, registers and the status latch must be saved and restored by microcode. Logic to disable the interrupt inputs is desirable.

Multiple Memory Spaces

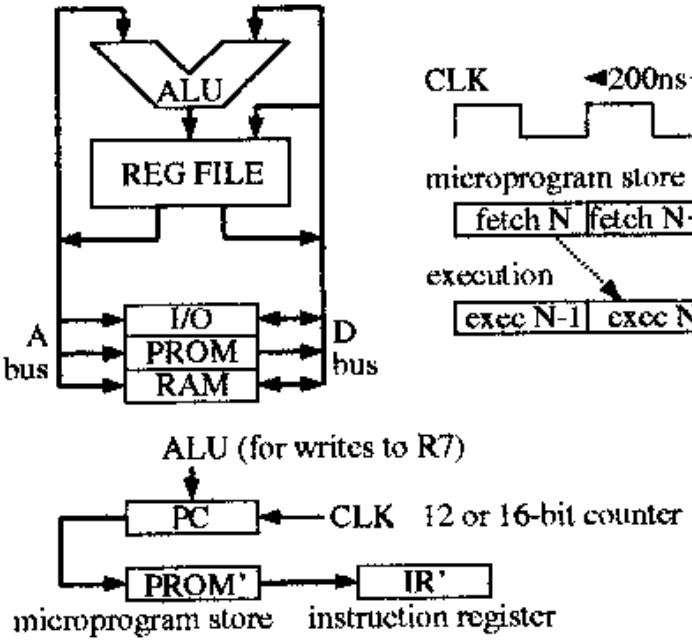
Many processors, such as the PDP-11, enforce a division of instruction and data space. Since the bits that select the PISC address register are available, and R7 is the program counter, a three-input NAND gate can produce an "instruction space" enable signal. This signal is correctly asserted for immediate addressing mode.

Further segmentation of the memory space depends upon the register assignment for the macro machine. One 2-to-4 decoder can identify microcode, macrocode, stack, and data spaces in the threaded stack machine. (This is suggestive of the 80x86.)

Parallelism

If the microcode memory space is separate from main memory, the fetch and execute phases can occur in parallel, and each microinstruction can execute in one clock cycle (Fig. 2). A 12-bit (optionally 16-bit) counter serves as a rudimentary microsequencer. This liberates the ALU from its sequencer role, and eliminates the need for the fetch pseudo-instruction. Each clock cycle fetches a microinstruction from the microprogram ROM.

Fig. 2 Parallel Fetch and Execute



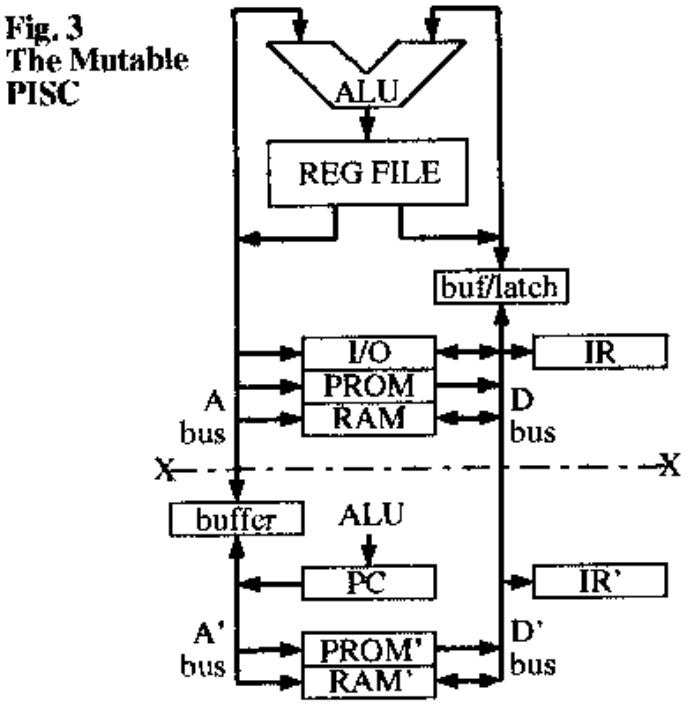
Absolute jumps are still possible by loading the counter from the ALU or Data bus. But ALU operations on the program counter are no longer permitted, eliminating the relative branch, conditional skip, and immediate addressing capabilities. Microinterrupts are also lost; macrointerrupts become impractical.

Since each microinstruction is executed one clock cycle after it is fetched (Fig. 2), a jump instruction will experience a one-clock-cycle delay, and the instruction following a jump will always be executed. This is the "delayed branch" seen in most microprogrammed machines and some pipelined RISC machines.

With this configuration, a main memory access need not complete in one clock cycle. The limiting factors in clock speed are the ALU path and the microinstruction ROM. Preliminary studies indicate that the clock speed of the PISC can be doubled, with two clock cycles required for a main memory reference.

Load/Store (RISC-like) Operation

A further modification (Fig. 3) re-unifies the microcode and macrocode memories. During ALU operations the address buffer is disabled, and fetch and execution remain simultaneous. But during a memory load or store, the output of the PC is tri-stated and the address buffer enabled, allowing the ALU to address the main memory (PROM' and RAM'). The count input of the PC is also disabled, and the IR is loaded with a "dummy" instruction (NOP). After the memory access is complete, the dummy instruction executes and the PC fetches the next instruction.



In this configuration the PISC can be viewed as a load/store architecture [3]. Like a RISC machine, all instructions execute in one clock cycle, except for memory references which require two.

If the address buffer is bidirectional, the PC can be routed to the ALU. This restores the relative branch, conditional skip, and immediate addressing capabilities.

The "Mutable PISC"

It would be unreasonable to hand students "a bag of parts" and a wirewrap tool, and expect them to construct the PISC as a class exercise. More attractive to the student, and perhaps less attractive to the experimenter, would be a printed-circuit-board of the basic PISC and its simpler enhancements. This is envisioned as the PISC-2 (Fig. 3).

The PISC-2 will use the Am29705 register file instead of the obsolescent and scarce 74172. This will require the addition of a 16-bit Data latch, and a few more control signals. Rather than expand the microinstruction word, a minimal function decoder will be added.

By judicious insertion and removal of components, the PISC-2 can be configured as a PISC-1 (two-clock instructions), a separate microprogram machine, or a load/store machine. Program development in the "unified memory" configurations could be via keypad and display, or an RS-232 serial port, using an on-board monitor program.

Conclusions

An educational drawback of the PISC is its abysmal implementation of "conventional" (1- or 2-operand) macromachines. This is a consequence of the PISC's original "mission": a stack processor using a minimum of standard TTL logic (2100 gates). But the PISC-1 vividly teaches that power does not necessarily imply complexity. Compare the speed of a 5 MHz PISC-1a to a 5 MHz 8086, when executing eForth primitives (given as number of 200 nsec clock cycles):

primitive	PISC	8086
NEXT	4	23
EXECUTE	4	19

DROP	6	29
EXIT	6	39
BRANCH	8	36
DUP	8	46
@	10	52
LIT	10	46
R>	10	50
R@	10	44
>R	10	51
!	10	53
ENTER	10	49
OVER	12	50
AND	12	53
0<	14	47
SWAP	12	61
?BRANCH	16	51/56
UM+	18	69

A processor design remarkably similar to the PISC, the QS2, has been successfully used in a VLSI design course [5]. The PISC has the advantage of being built from standard TTL; thus, it is within the reach of even the most impoverished institutions and students.

References

1. Bell, James R., "Threaded Code," Communications of the ACM, Vol. 16 No. 6 (June 1973), pp. 370-372.

2. Dewar, Robert B. K., "Indirect Threaded Code," Communications of the ACM, Vol. 18 No. 6 (June 1975), pp. 330-331.

3. Hennessy, John L. and Patterson, David A., Computer Architecture: A Quantitative Approach, Morgan Kaufmann Publishers, San Mateo, CA (1990).

4. Koopman, Philip J., Stack Computers: the new wave, Ellis Horwood Ltd., Chichester, England (1989).

5. Rible, John, "QS2: RISCing it all," Proceedings of the 1991 FORML Conference, Forth Interest Group, Oakland, CA (1991), pp. 156-159.

6. Stallings, William, Computer Organization and Architecture, Macmillan Publishing Co., New York (1987).