RISC I: A Reduced Instruction Set VLSI Computer

David A. Patterson and Carlo H. Sequin

Introduction

It is a general trend today, exemplified by many machines, to increase the complexity of processor architectures.
However, the number of devices available on a single chip is still limited, leading to latency and loss in power in data transfers across chip boundaries.

The aim of the RISC project is to "explore alternatives to the general trend toward[s] architectural complexity." The hypothesis is that reducing the instruction set can enable a VLSI architecture to use scarce resources more effectively than CISC.

It is expectd that this approach will reduce design time, design errors and execution time.

For simplicity, the following constraints were placed on the proposed implementation:

- Execute one instruction per cycle (microcode control is made unnecessary).
- Uniform instruction size (implementation is simplified).
- Only load and store instructions access memory (design is simplified).
- Support HLLs.

32-bit addresses, 8-, 16- and 32-bit data, and several 32-bit registers are supported.

Support for HLLs

As long as the system hides lower levels from the programmer, whether the HLL is implemented by the hardware or the software should be irrelevant. Most pieces of the RISC HLL system are in the software.

The following were determined statistically through analysis of C and Pascal programs:

- integer constants (of which over 80% were scalars) occur as frequently as entries in arrays or structures (of which over 90% were global).
- procedure call/return is the most time-consuming operation overall in typical HLL programs.

Basic Architecture of RISC I

There are four types of instructions: ALU, memory access, branch and miscellaneous. Register 0 always contains 0.

Load and store instructions use two CPU cycles (as an alternative to extending the length of a cycle). There is only one addressing mode (index + displacement) but register 0 allows us to synthesise absolute and register indirect addressing.

Branch instructions include call, return and conditional and unconditional jump.

Register Windows

It is important to optimise the call operation, since many instructions in CISC are subroutines in RISC, in addition to ordinary high-level function calls.

The allocation of local scalars in registers lets us avoid register saving and restoring, using multiple banks of registers (each of which is allocated to a new procedure when it is called). Returning only alters a pointer, indicating the old set.

RISC I's scheme involves setting aside 10 registers (r0 to r9) as global registers, and using a bank of 22 remaining registers as a store of local variables for a single procedure. Banks are overlapped; the top 5 registers (HIGH; r26 to r31) are parameters passed from the caller function, the middle 12 (LOCAL; r16 to r25) are for local scalar storage, and the bottom 5 (LOW; r11 to r16) are parameters passed to the callee function. The LOW registers of the caller become the HIGH registers of the callee, which is accomplished by overlapping the hardware.

When there are no free register banks, a trap to a software routine adjusts a register overflow stack in memory and a corresponding stack pointer.

A related issue is that of pointers – local variables stored in registers would not have addresses. Putting these variables in memory would slow down variable access. Rather, a portion of the address space is reserved for the registers and a single comparison allows one to determine whether an address points to a register or to memory.

Delayed Jump

The processing of RISC I instructions is pipelined, but sophisticated lookahead techniques for branch instructions are too hardware-heavy for a single chip. Thus, jumps are redefined to take effect only after the following instruction.

Either a NOP is inserted (to push the incorrect branch forward so it does not overlap with the jump's execution) or an optimiser can be used that rearranges the instructions to take place without NOP.

Evaluation

Register Windows

This scheme "reduces the cost of using procedures significantly". It also reduces off-chip memory accesses – less than 20% of the instructions were loads/stores and more than 50% were register-to-register.

Delayed Jump

Initially, 18% of the instructions are NOPs. However, a simple optimiser can reduce this to 8%. 90% of unconditional NOPs were removed, but only 20% of the conditional ones. If the target instruction modifies only temporary resources (like condition codes), the optimiser can be improved to replace the NOP with it.

Overall Performance

On average, RISC I uses $\frac{2}{3}$ more instructions than VAX and $\frac{2}{5}$ more than PDP-11.

Dynamically, RISC I substantially reduces data accesses, but the number of instruction words accessed increased.

Execution time (judged using simulators) was judged as 400 ns per cycle; there is thus a chance for it to be faster than VAX 11/780.

Memory Interface

The memory interface is a main performance bottleneck. Under the assumption that two CPU cycles are needed to access data memory, performance degraded 10%; however, increasing instruction access time would reduce performance considerably more.

An on-chip cache would improve performance considerably, but it has to be larger than the register file. Also, translation/decoding might increase the CPU cycle time.

The number of instruction fetches has increasing and is a "major speed-limiting factor". Thus an instruction cache would be very beneficial. Since it does not have to be written to, its controller can be simpler as well.

Summary

- This approach appears promising based on the few programs that were studied.
- Complicated addressing schemes are not a vital contributor to high throughput.
- The register window scheme makes significant contributions towards performance.
- Most of the complexity found in modern machines has been eliminated, while retaining code density to some degree.
- The functionality of RISC I has not been compromised; it can emulate more complex computers.
- The design time of a single-chip computer is, in addition, much lesser than that of traditional architectures.