Topics in computer architecture

Reduced instruction set computers

P.J. Drongowski
SandSoftwareSound.net

Architecture in the 1970's

- Technology
  - Microprogrammed implementation style
  - Control memory was 10 times faster than primary
  - 8192 bits of ROM occupied space of 8 register bits
- Arguments for a richer instruction set
  - Simplify compilers
    - Register oriented compilers were hard to build
    - Use stack or
    - Memory to memory operations instead
  - Alleviate software crisis
    - Move function to hardware
    - Machine instructions to resemble HLL statements
    - Close "semantic gap"
  - Improve architectural quality
    - Measure "quality" as opposed to execution speed
    - Architectural metrics
      - Program size
      - Number of bits per instruction
      - Bits of data fetched from memory
- Memory efficiency was a dominating concern
  - Slow and expensive core magnetic core memory
  - Belief: Execution speed proportional to program size
  - Code improvement
    - Find long sequence of instructions and
    - Replace with a single instruction

Design principles (1970's)

- Cheap, dense ROM ⇒ Inexpensive additions to ISA
- Microinstructions were faster than ISA instructions
  - ⇒ Move software function to microcode
  - ⇒ Faster, more reliable functions
- Execution speed was proportional to program size
  - ⇒ Smaller programs
  - ⇒ Faster computers
- Register oriented machines were passe
  - ⇒ Stacks or memory-to-memory architectures
  - ⇒ Complex instructions for procedure linkage

Technological changes

- Semiconductor memory
  - Speed would be comparable to CPU
  - Replace core memory as density increased
- Bloated microcode
  - 400,000 bits became typical
  - Errors could not be removed
  - ROM was replaced by RAM (writeable control store)
- Cache memory
  - Small, fast buffer between CPU and primary memory
  - Substantially improved execution speed
- Compilers used only a subset of the ISA
  - Could not always use complex instructions
  - Could use simpler instructions due to better analysis

Writeable control store

- Could not run faster than one uinstruction per clock
- 3 to 4 microcycles per instruction on average
- Migrate application into microcode
- Provide a writeable control store for application microcode
- Problems
  - Microcode is tedious to write and debug
  - Restart on virtual memory fault
  - Limited control store size ˝ time lost optimizing by hand

Examples

Machine	IBM 370/168	VAX-11/780	iAPX-432
Year	1973	1978	1982
Instructions	208	303	222
Control memory	420 Kbit	480 Kbit	64 Kbit
Instruction size	16-48	16-456	6-321
Technology	ECL MSI	TTL MSI	nMOS VLSI
Cache size	64 Kbit	64 Kbit	0
Execution model	reg-mem mem-mem reg-reg	reg-mem mem-mem reg-reg	stack mem-mem

RISC origins

- Instructions should be as fast as microinstructions
- Program or compile to simple operations
- Exploit higher speed of caches and semiconductor memory

Design principles

- Keep function simple
  - Short cycle time
  - Small number of cycles per function
- Execute simple instructions as fast as microinstructions
  - Cache uses same memory technology as WCS
  - Execution speed should be the same
- Make hardware primitives available in machine language
  - Provide same hardware functionality as microengine
  - Use runtime library instead of complex instruction
- Simple decode and pipelined execution
  - More important than program size
  - Simple decode ⇒ fast cycle time
  - Pipelining

    ⇒ Careful partition of function into phases

    ⇒ Each phase is shorter than total instruction time
- Remove work at compile time
  - Keep operands in registers
  - Use register to register instructions
  - Operands are not discarded as in mem-to-mem ISA

RISC traits

- Register to register operations
- LOAD and STORE memory access
  - Simplifies processor design
  - VM fault handling is localized
- Reduced operations
  - Register to register operations take one cycle
  - Hardwired control (microcode unnecessary)
  - Execute multiple cycle instructions in coprocessor
- Reduced addressing modes
  - Two modes: indexed and PC-relative
  - Synthesize other more complicated modes
- Simple instruction formats
  - Instructions do not cross word boundaries
  - Little or no decode time
  - Instructions do not fall across page boundaries
- Delayed (effect) branches
  - Do not take effect until after the *following* instruction
  - Eliminates pipeline "bubbles" due to a flush
  - Compiler handles arrangement of code

Early examples (January 1985)

Machine	IBM 801	RISC I	MIPS
Year	1980	1982	1983
Instructions	120	39	55
CS size	0	0	0
Instruction size	32	32	32
Technology	ECL MSI	nMOS VLSI	nMOS VLSI
Execution model	reg-reg	reg-reg	reg-reg

RISC approaches

- Compiler technology vs. register windows
  - IBM 801 and Stanford MIPS
    - Large general register set
    - Graph coloring algorithm for register allocation
  - Berkeley RISC
    - Register windows
    - Based on observations of program behavior
  - Register windows are bigger and slower
  - Drawbacks of compiler approach
    - Compiler is twice as slow
    - Penalty for register save / restore on procedure call
    - Expand some procedures in-line
  - Frequency of LOAD and STORE
    - 801 - 30 percent (32 registers)
    - MIPS - 35 percent (16 registers)
    - RISC - 15 percent (32 registers per window)

- Memory access
  - Access requires minimum of two cycles
    - One cycle to compute address
    - Second cycle to actual read from memory
  - RISC - Use two cycles and shim the pipe
  - 801 and MIPS - Delayed LOAD
    - Two memory ports - one data, one instruction
    - Data not available until third cycle
    - Second instruction cannot use memory data
    - Data dependency hazard
    - Slot can be filled 90 % of the time

- Pipelines
  - 801 - Four stage pipeline
  - RISC - Three stage pipeline
  - 801 and RISC - Value forwarding
  - MIPS
    - Microprocessor without Interlocked Pipelined Stages
    - Compiler removes resource conflicts

Hidden RISC

- VAX architecture study
  - VLSI VAX - nine custom chips
  - Observation
    - 20 % of instructions take
    - 60 % of the microcode, but are
    - 0.2 % of all instructions executed
  - MicroVAX 32
    - Subset of the VAX ISA
    - Complex instructions in software
    - One chip plus optional FP chip
  - VLSI VAX was only 20 % faster
  - 20 % can be gained by simpler compiler

- IBM 360 model 44
  - Subset ISA in hardware
  - Complex instructions in software
  - Better cost/performance than neighbors in family

Source: "Reduced instruction set computers,"
David A. Patterson, CACM, January 1985.

RISC (CPI) goals

- Minimize cycles per instruction (CPI)
  - Simple instructions
  - Large, low miss rate caches
  - Load / store architecture
  - Pipelining
  - Minimize loss for incorrectly predicted branch
    - Delayed branch
    - 20 % of instructions are control transfers
    - 10 % are conditional control transfers

- Minimize number of instructions executed
  - Large general register set
  - Reduce occurrence of loads and stores
  - Windows to pass procedure arguments / results
  - Interprocedural register allocation (IRA)
  - Do not modify condition codes on every instruction
  - Compiler can more easily rearrange code
  - SPARC executes 20 % more instructions than 68000

- Minimize clock period
  - Depends on design of cache and pipeline
  - Critical circuit delay ˜ cache access path
  - Return cache value in one clock period
  - Simple formats speed decoding and dependency checks

Measures

- P: Large compute-bound program
- CPI: Cycles per instruction
- I_P : Number of instructions executed by P
  - Depends on benchmark program
  - Efficiency (quality) of the instruction set
  - Quality of the compiler
  - Number and organization of registers
- C_P : Average number of CPI executed by program P
  - Depends on benchmark program and compiler
  - Microarchitecture
  - Size and speed of cache/memory system
    - Sensitive to cache miss rate
    - More misses means more lost memory wait cycles
  - Goal: Execute most frequent instructions in the least number of cycles
- T: Time per cycle (reciprocal of clock frequency F)
  - Depends on chip technology
  - Projected cost
  - Development time and risk

- Time to execute P = $I_P \times C_P \times$
- $MIPS_P$ rate = $\frac{1}{C_P \times}$
- Time to execute P = $\frac{I_P}{MIPS_P}$

Characterization of programs

- Berkeley characterization study (Patterson & Sequin, 1982)
- Four Pascal programs
  - Pascal compiler
  - Macro expansion phase of DA system
  - Pascal prettyprinter
  - File comparison program
- Four C language programs
  - Portable C compiler (VAX)
  - VLSI mask layout program
  - Text formatter
  - Sorting program

Dynamic frequency of operands	Pascal & C	Remarks
Integer constants	20 ± 7 %	
Scalars	55 ± 11%	> 80 % refer to local variables
Arrays/structures	25 ± 14 %	> 90 % refer to global variables

Dynamic frequency of statement types	Pascal	C
Assignment	45 ± 8 %	38 ± 15 %
If	29 ± 8 %	43 ± 17 %
Call/return	15 ± 1 %	12 ± 5 %
With	5 ± 5 %	3 ± 1 %
Loop	5 ± 0 %	3 ± 4 %
Case	1 ± 1 %	< 1 ± 1 %

Characterization (2)

- Observations
  - Loops were counted once
  - Statements within loop counted once per execution
  - Table below indicates amount of execution time
  - Call/return includes save/restore, parameter overhead
  - For loop statement, count includes all instructions executed during each iteration

Weighted dynamic frequency of statement types	Machine instructions		Memory references	
	Pascal	C	Pascal	C
Call/return	31 ± 3 %	33 ± 14 %	44 ± 4 %	45 ± 19 %
Loop	42 ± 3 %	32 ± 6 %	33 ± 2 %	26 ± 5 %
Assignment	13 ± 2 %	13 ± 5 %	14 ± 2 %	15 ± 6 %
If	11 ± 3 %	21 ± 8 %	7 ± 2 %	13 ± 5 %

- More observations
  - 80% of all scalar references were to local variables
  - 90% of array/structure references were to globals
  - Call/return are the most time-consuming statements
  - "RISC architectures for VLSI," Katevenis, 1985
    - Programs are organized into procedures
    - Calls are frequent and costly in time
    - Procedures have few arguments and local variables
    - Locals are usually scalars and heavily used
    - Nesting depth fluctuates within narrow ranges
  - "Empirical ..." Lunde, CACM, March 1977
    - 10 regs sufficient 90% of time for 41 programs studied
    - 10 regs sufficient 98% for 36 of the 41 programs
    - Size, complexity, efficiency did not imply many regs
  - "Implications ..." Tanenbaum, CACM, March 1978
    - Assignments with 1 RH side term: 75%S, 64%D
    - Assignments with 2 RH side terms: 15%S, 20%D
    - 98%D of procedures had less than 6 arguments
    - 92%S of procedures has less than 6 scalar variables