RISC Architecture
Ch 12

Some History

Instruction Usage Characteristics

Large Register Files

Register Allocation Optimization

RISC vs. CISC

Copyright Teemu Kerola 2001

Original Ideas Behind CISC (Complex Instruction Set Comp.)

- Make it easy target for compiler
  - small semantic gap between HLL source code and machine language representation
  - good at the time when compiler technology big problem
  - make it easier to design new, more complex languages
- Do things in HW, not in SW
  - addressing mode for 2D array reference?

Copyright Teemu Kerola 2001

Occam's Toothbrush

- The simple case is usually the most frequent and the easiest to optimize!
- Do simple, fast things in hardware and be sure the rest can be handled correctly in software

Copyright Teemu Kerola 2001

RISC Approach (2)

- Optimize for execution speed instead of ease of compilation
  - compilers are good, let them do the hard work
  - do most important things very well in HW (e.g., 1-dim array reference) and the rest in SW (e.g., 3-dim. array references)
- What are *most important* things?
  - those that consume most of the time (in current systems?)
  - is this a moving target?

Copyright Teemu Kerola 2001

Amdahl's Law (5)

Speedup due to an enhancement is proportional to the fraction of the time that the enhancement can be used

Floating point instructions improved to run 2X; but only 10% of actual instructions are FP?

No speedup

$$\text{ExTime}_{new} = \text{ExTime}_{old} \times (0.9 * 1.0 + .1 * 0.5)$$

$$= 0.95 \times \text{ExTime}_{old}$$

$$\text{Speedup}_{overall} = \frac{\text{ExTime}_{old}}{\text{ExTime}_{new}} = \frac{1}{0.95} = 1.053$$

<< 2 !!!

Copyright Teemu Kerola 2001

Where is Time Spent? (6)

- Dynamic behaviour
  - execution time behaviour
- Which operations are most common?
- Which types of operands are most common?
- Which addressing modes are most common?
- Which cases are most common?
  - E.g., number of subroutine parameters?
- What is the case with current machines?

Table 12.2

Table 12.3

Table 12.4

Copyright Teemu Kerola 2001

Original Ideas Behind RISC (3)

- Very large set of registers
  - more registers than can be addressed in any single machine instruction?
  - compilers can do good register allocation
- Very simple and small instruction set is faster
  - instruction pipeline is easy to optimise
- Economics
  - Simple to implement

    $\Rightarrow$ quickly to market $\Rightarrow$ beat competition

    $\Rightarrow$ recover development costs $\Rightarrow$ stay in business

Copyright Teemu Kerola 2001

CISC Architecture (5)

- Large and complex instruction sets
  - direct implementation of HLL statements
    - case statement?
    - array or record reference?
- May be targeted to specific high level language
  - may not be so good for others

E.g., i432 and Ada

microJava, JEM?

- Many addressing modes
- Many data types

Vax11/780

char string, float, int, leading separate string, numeric string, packed decimal string, string, trailing numeric string, variable length bit field

Copyright Teemu Kerola 2001

Large Register File

- Overlapping register windows

Fig. 12.1

  - fixed max nr (6?) of subroutine parameters
  - fixed max nr of local variables
  - function return values are directly accessible to calling routine in temporary registers
    - no copying needed
- I.e., when possible, use registers instead of stack for subroutine implementation

Copyright Teemu Kerola 2001

Problems with Large Register Files (2)

- What if run out of register sets? Fig. 12.2
  - save & restore values from memory (stack)
  - hopefully not very common
    - call stacks are usually not very deep!
    - find out from studies what is enough usually
- Global variables
  - store them always in memory?
  - use another, separate register file?

Copyright Teemu Kerola 2001

Register Files vs. Cache (2)

- Would it be better to use the same real estate (chip area) as cache?
  - register files have better locality
  - caches are there anyway
  - caches solve global variable problem naturally
    - no compiler help needed
  - accessing register files is faster
- Third way to use the space for register files: register renaming
  - see next lecture on superscalar architecture

Table 12.5

Fig. 12.3

Copyright Teemu Kerola 2001

Register Allocation (3)

- Goal: Prob(operand in register) = high
- Symbolic register: any quantity that could be in register
- Allocate symbolic regs to real regs
  - if some symbolic regs are not used in same time intervals, then they can be assigned to the same real regs
  - use graph colouring problem to solve reg allocation problem

Copyright Teemu Kerola 2001

Graph Colouring Problem (2)

- Given a graph with connected nodes, assign n colours so that no neighbouring node has the same colour
  - topology
  - NP complete problem (see course on Design and Analysis of Algorithms)
- Application to register allocation
  - node = symbolic register
  - connecting line: simultaneous usage
  - no connecting line: can allocate symbolic registers to same physical register
  - n colors = n registers

Fig. 12.4

Copyright Teemu Kerola 2001

How Many Registers Needed?

- Usually 32 enough
  - more ⇒ longer register address in instruction
  - more ⇒ no real gain in performance
- Less than 16?
  - Register allocation becomes difficult
  - not enough registers
    ⇒ store more symbolic registers in memory
    ⇒ slower execution

Copyright Teemu Kerola 2001

RISC Architecture (4)

- Complete one (or more!) instruction per cycle
  - read reg operands, do ALU, store reg result
  - all instructions are simple instructions
- Register to register operations
  - load-store architecture
- Simple addressing modes
  - easy to compute effective address
- Simple instruction formats
  - easy to load and parse instructions
  - fixed length

Copyright Teemu Kerola 2001

RISC vs. CISC

- Fixed instruction length (32 bits)
- Very few addressing modes
- No indirect addressing
- Load-store architecture
  - only load/store instructions access memory
- At most one operand in memory
- Aligned data
- At least 32 addressable registers
- At least 16 FP registers

Table 12.8

Copyright Teemu Kerola 2001

RISC & CISC United? (5)

- Pentium II, CISC architecture
- Each complex CISC instruction translated during execution (in CPU) into multiple fixed length 118 bit micro-operations (uop)
  - 1-4 uops/IA-32 (32 bit Intel Architecture) instruction
- Lower level implementation is RISC, working with RISC micro-ops
- Best of both worlds?
- Could CPU area/time be better spent without this translation?
  - Who wants to try? Transmeta Corporation?
  - Why? Why not?

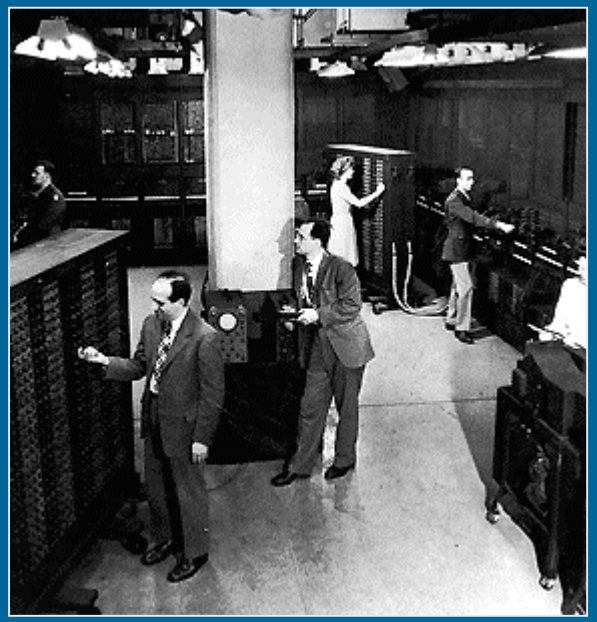
Copyright Teemu Kerola 2001

RISC & CISC United? (3)

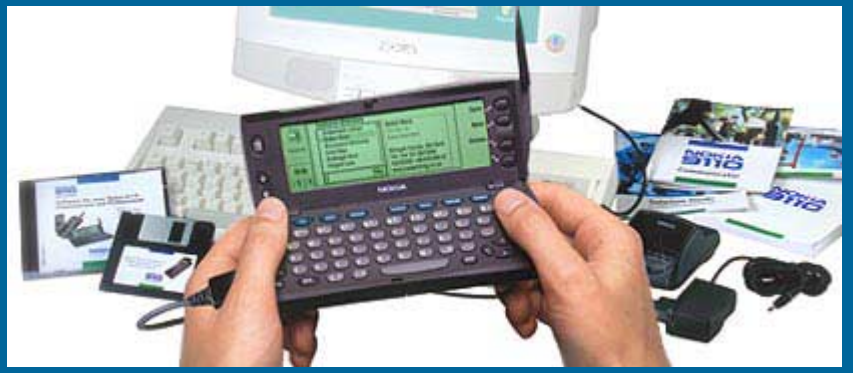
- Crusoe (by Transmeta) – emulate CISC
  - CISC architecture (IA-32, IA-64, Java?) visible to outside
- Each complex CISC instruction translated just before execution (in separate JIT translation with possibly optimized code generation) into multiple fixed length simple micro-operations
  - translation in SW, not in HW like with Pentium
- Lower level implementation is RISC, working with RISC micro-ops
  - VLIW (very long instruction word, 128 bits)
    - 4 uops/instruction (I.e., 4 *atoms/molecule*)

Copyright Teemu Kerola 2001

-- End of Chapter 12: History and RISC --



50 years



50 years

???

Copyright Teemu Kerola 2001