# RISC Architecture
# Ch 12

Some History

⟹ Instruction Usage
Characteristics

Large Register Files

Register Allocation
Optimization

RISC vs. CISC

# Original Ideas Behind CISC
# (Complex Instruction Set Comp.)

- Make it easy target for compiler
  - small <u>semantic gap</u> 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?

# 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

# RISC Approach (2)

- Optimize for execution <u>speed</u> instead of ease of compilation
  - compilers are good, let them do the hard work
  - do most important things very well in HW (machine instruction), rest in SW (subroutines)
- What are *most important* things?
  - Those that consume most of the time (in current systems)

# 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

$$\text{ExTime}_{new} = \text{ExTime}_{old} \times (0.9 * 1.0 + .1 * 0.5)$$
$$= 0.95 \times \text{ExTime}_{old}$$

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

$$\ll 2 \ !!!$$

10/10/2000                                    Copyright Teemu Kerola 2000                                    22

# Where is Time Spent? (5)

- Dynamic behaviour
  - execution time behaviour                           Table 12.2
- Which operations are most common?
- Which types of operands are most common?            Table 12.3
- Which addressing modes are most common?
- Which cases are most common?    Table 12.4
  - E.g., number of subroutine parameters?

10/10/2000                                    Copyright Teemu Kerola 2000                                    23

# Ideas Behind RISC (3)

- Very large set of registers
  - bigger than can be addressed in machine instruction?
  - compilers can do good register allocation
- Very simple and small instruction set is faster
  - easy to optimize instruction pipeline
- Economics
  - Simple to implement
    $\Rightarrow$ quickly to market
    $\Rightarrow$ beat competition
    $\Rightarrow$ recover development costs

10/10/2000 Copyright Teemu Kerola 2000 24

# 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
- 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

10/10/2000 Copyright Teemu Kerola 2000 25

# Large Register File

- Overlapping register windows $\boxed{\text{Fig. 12.1}}$
  - 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

# Problems with
# Large Register Files (2)

- What if run out of register sets? $\boxed{\text{Fig. 12.2}}$
  - save & restore values from memory
  - 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?

# Register Files vs. Cache

- Would it be better to use the same real estate (chip area) as cache?  Table 12.5
  - register files have better locality
  - caches are there anyway
  - caches solve global variable problem naturally
    - no compiler help needed
  - accessing register files is faster  Fig. 12.3

10/10/2000                    Copyright Teemu Kerola 2000                    28

# Register Allocation (3)

- Goal: Prob(operand in register) = high
- Symbolic register: any <u>quantity</u> 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 coloring problem to solve reg allocation problem

10/10/2000                    Copyright Teemu Kerola 2000                    29

# Graph Coloring Problem (2)

- Given a graph with connected nodes, assign n colors so that no neighboring node has the same color
  - topology
  - NP complete problem
- Application to register allocation
  - node = symbolic register
  - connecting line = simultaneous usage
  - n colors = n registers

Fig. 12.4

10/10/2000                      Copyright Teemu Kerola 2000                      30

# How Many Registers Needed?

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

10/10/2000                      Copyright Teemu Kerola 2000                      31

# RISC Architecture (4)

- <u>Complete</u> one instruction per cycle
  - read reg operands, do ALU, store reg result
  - <u>all</u> 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

10/10/2000 Copyright Teemu Kerola 2000 32

# RISC vs. CISC (8)

- 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

10/10/2000 Copyright Teemu Kerola 2000 33

# RISC & CISC United? (4)

- Pentium II, CISC architecture
- Each complex CISC instruction translated during execution (in CPU) into multiple fixed length simple micro-operations
- Lower level implementation is RISC, working with RISC micro-ops
- Could CPU area/time be better spent without this translation?
    - Who wants to try? Transmeta Corporation?
    - Why? Why not?

# RISC & CISC United? (3)

- Crusoe (by Transmeta)
    - CISC architecture (= Intel) visible to outside
- Each complex CISC instruction translated during execution (in separate <u>translation</u> with optimized code generation) into multiple fixed length simple micro-operations
- Lower level implementation is RISC, working with RISC micro-ops

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

50 years

???

50 years