RISC and CISC

Speed v/s Space

Introduction

- CISCs were the first kind of processors.
- Innovations in RISC architectures are based on a close analysis of a large set of widely used programs.
- One of the main concerns of RISC designers was to maximize the efficiency of pipelining.
- Both RISCs and CISCs try to solve the same problem. CISCs are going the traditional way of implementing more and more complex instructions. RISCs try to simplify the instruction set.

CISC

- In the 1970's, memory was expensive and small in size, so people designed computers that would pack as many action as possible in a single instruction this saved memory space, but added complexity.
- Example: adding 2 numbers which are in memory and storing the result back to memory. Something like, a = a + b;
- CISC – Complex Instruction Set Computing
- RISC – Reduced Instruction Set Computing
  - Less Number of instructions
  - Less Variety of addressing modes
  - Less Size of each instruction
  - More Number of registers
- BCD add instruction is a CISC if it does both add and correct it in one instruction.

CISC features or characteristics

- Microprogrammed control unit
- Large number of instructions (200-500)
- Instructions can do more than 1 thing (that is, an instruction could carry out 2 or more actions)
- Many addressing modes
- Instructions vary in length and format
- This was the typical form of architecture until the mid 1980s, RISC has become more popular since then although most architectures remain CISC

RISC

- Hardwired control unit
- Instruction lengths fixed (usually 32 bits long)
- Instruction set size is limited (80-100 instructions)
- Instructions rely mostly on registers; register to register operations. Memory accessed only on loads and stores
- Few addressing modes
- Easy to pipeline for great speedup in performance
- CPU takes less silicon area to implement and remaining area can be used for providing more number of registers.

RISC

Pure RISC machines have the following features

1. All RISC instruction codes have the same number of bits (Typically 32 bit)

- CISC instructions can vary and have no fixed length.
- Because **RISC** have fixed instruction code, this makes it a lot easer to be pipelined, thus making it faster

2.The RISC instruction set includes only very simple operations that can ideally be executed in a small number of clock cycles.

- These instructions can then be moved through a pipeline quickly and not hold the pipeline up
- for **BCD** add instruction, we should make add and correction two instructions to make it **RISC** machine and have it easily pipelined

RISC

3. **RISC instructions for reading data from memory include only a single operand load instruction and a single operand store instruction.**
   - ❑Because of this **RISC** machines are referred to as "**load and store**" machines
   - ❑Most **CISC** machines have a single instruction to load the operand from memory and then add operand to a register, note that we doing two things with a single instruction. Just like in **BCD** add instruction
   - ❑Two advantages of a simple load and store
     - ❑machine code only takes a few bits
     - ❑machine code can be quickly decoded, which makes pipelines easier to design

RISC

4. **The RISC instruction set is designed so that compilers can efficiently translate high-level language constructs to instruction codes for the machine.**
   - ❑Compiler writers took little advantage of powerful **CISC** instructions because they were very machine specific and they are not very portable this ended up having compilers that acted like they are working for a **RISC** machine even though they were working for **CISC** machine
   - ❑A problem with pure **RISC** is that it takes many small instructions to do anything, which uses a lot of memory; **this excessive use of memory is called as "code bloat"**
   - ❑A pure RISC machine also requires a greater memory bandwidth because we are constantly doing lot of fetch, load and store operations.
   - ❑ microprocessors are usually designed using both **RISC** and **CISC** approach, **(a combination of both), in order to get best of both worlds.**

Arguments for CISC

- A rich instruction set should simplify the compiler design by having instructions which match the HLL statements.
- Many powerful instructions are supported, making the assembly language programmer’s job much easier.
- Generally micro-coded CU, and hence easy for updating and adding new instructions in next generation processors. Easy to have backward compatibility of the instructions (processor).

Comparing RISC and CISC.

CISC

- Large number of instructions- 150 to 300. (more can be counted by considering variety of addressing modes)
- Employs a variety of data types and a large number of addressing modes.
- Variable length instruction formats.
- Instructions can manipulate operands in memory.

RISC

- Fewer number of instructions. Less than 100.
- Supports less variety of addressing modes.
- Fixed length instructions. 32 bits usually. Easy to decode instruction formats.
- Mostly register-register operations. Memory access is by LOAD and STORE instructions.

Comparing RISC and CISC.

CISC

- Number of cycles per instruction (CPI) varies from 1 to 20 depending on the instruction.
- 8 to 32 GPRs and no support is available for parameter passing and function calls.
- Micro-programmed control unit.
- Complexity is in hardware.
- Primary goal is to complete a task in as few lines of assembly as possible.

RISC

- Number of CPI is 1 as it uses pipelining. Pipeline in RISC is optimised because of simple instructions and uniform instruction format.
- 64 to 512 GPRs are available that are helpful for parameter passing in function calls.
- Hardwired control unit.
- Complexity is in software.
- Primary goal is to speedup individual instruction.

The Performance Equation

The following equation is commonly used for expressing a computer's performance ability:

$$\frac{\text{time}}{\text{program}} = \frac{\text{time}}{\text{cycle}} \times \frac{\text{cycles}}{\text{instruction}} \times \frac{\text{instructions}}{\text{program}}$$

1 2

The CISC approach

- minimizes the number of instructions per program (2)
- sacrificing the number of cycles per instruction. (1)

RISC does the opposite

- reduces the cycles per instruction (1)
- sacrificing number of instructions per program (2)