CPU Architecture (RISC and CISC)

Contents of Lecture:

- CPU Architecture
- The Semantic Gap
- Main features of CISC
- Main Characteristics of RISC
- RISC vs. CISC

References for This Lecture:

- William Stallings, Computer Organization and Architecture Designing For Performance, 9th Edition, Chapter 15: *Reduced Instruction Set Computers*

CPU Architecture:

- CPU Architecture also known as Instruction Set Architecture (ISA).
  - ISA allows communication between software and hardware.
  - ISA is also a group of commands for CPU in machine Language.
  - Basically ISA tall you how processor is going to process your program and instructions (commands). Instructions is also called opcode or operation code.

  - Example of opcode:
    - add
    - Jump
    - Load
    - store

- There are two type of ISA:
  - CISC
  - RISC

- **CISC:**
  - CISC is stand for **Complex Instruction Set Computing**. It is architecture for **large** number of instructions.
  - **Example of CISC:**
    - The microprocessor of CISC is Intel 80286 which is the 64bit version of 8086 instructions set.
    - Its used on desktop (PC) and laptop.

- **RISC:**
  - RISC is stand for **Reduced Instruction Set Computing**. It is has **lesser** number of instructions.
  - **Example of RISC:**
    - ARM processor
    - Its used on taplet, smart phone and even in game consle.

❖ **Example of CISC and RISC:**

Writing A * B

❖ For CISC Instruction Set there is special command called mull (to complex)

Mull A, B

❖ For RISC Instruction Set its divided into numbers of small instructions (small instructions)

```
LOAD    R1,  A
LOAD    R2,  B
PROD    A,   B
STORE   R3,  A
```

❖ Both RISC and CISC architectures have been developed as an attempt to cover **the semantic gap**, and consequently to reduce the ever growing software costs.

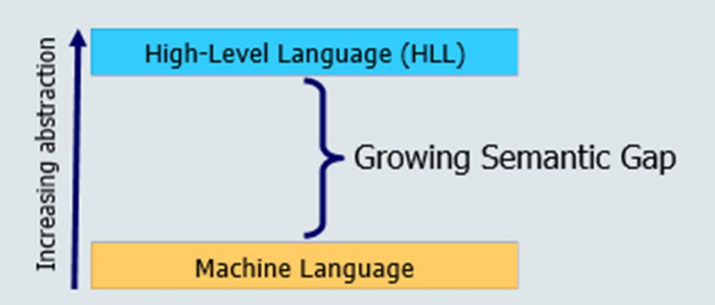
The Semantic Gap:

❖ Semantic gap is the difference between the operations provided in HLLs and those provided in computer architecture.

❖ In order to improve efficiency of software development, powerful high-level programming languages have been developed (e.g., C++, Java).

- ✓ They support higher levels of abstraction.

❖ This evolution has increased the semantic gap between programming languages and machine languages.



Main features of CISC:

❖ A large number of instructions (> 200) and complex instructions and data types.
❖ Many and complex addressing modes.

❖ Direct hardware implementations of high-level language statements.

- ✓ For example, CASE (switch)

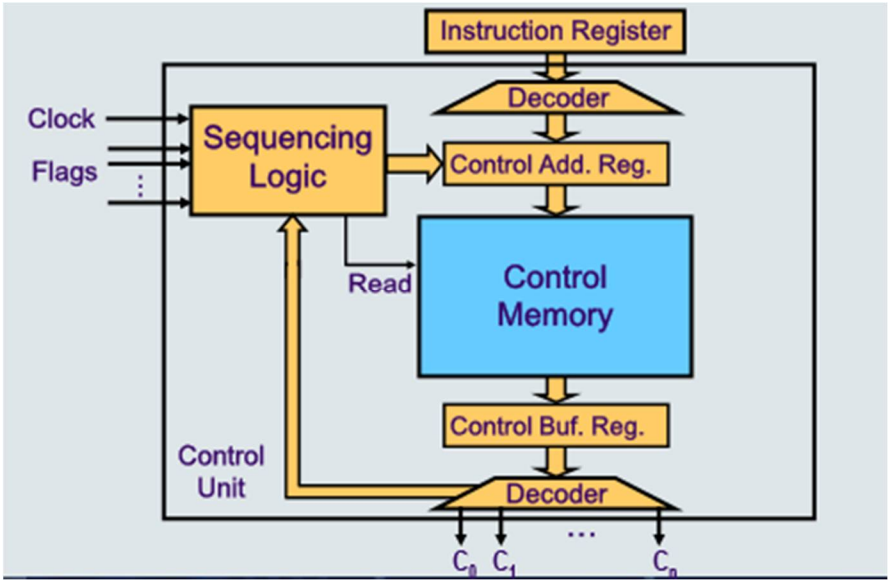
❖ Microprogramming techniques are used so that complicated instructions can be implemented.
❖ Memory bottleneck is a major problem, due to complex addressing modes and multiple memory accesses per instruction.

Microprogrammed Control:

- Microprogramming is a technique used to implement the control unit.

- The basic idea is to implement the control unit as a micro-program execution machine (a computer inside a computer).
  - ✓ The set of micro-operations occurring at one time defines a microinstruction.
  - ✓ A sequence of microinstructions is called a microprogram.

- The execution of a machine instruction becomes the execution of a sequence of micro-instructions.
  - ✓ This is similar to that a C++ statement is implemented by a sequence of machine instructions.



Problems with CISC:

- A large instruction set requires complex and potentially time consuming hardware steps to decode and execute the instructions.

- Complex machine instructions may not match high-level language statements exactly, in which case they may be of little use.
  - ✓ This will be a major problem if the number of languages is getting bigger.

- Instruction sets designed with specialized instructions for several high-level languages will not be efficient when executing program of a given language.
- Complex design tasks.

Main Characteristics of RISC:

- A small number of simple instructions (desirably < 100).
  - ✓ Simple and small decode and execution hardware are required.
  - ✓ A hard-wired controller is needed, rather than using microprogramming.
  - ✓ The CPU takes less silicon area to implement, and runs also faster.

- Execution of one instruction per clock cycle.
- The instruction pipeline performs more efficiently due to simple instructions and similar execution patterns.

- Complex operations are executed as a sequence of simple instructions.
  - ✓ In the case of CISC they are executed as one single or a few complex instructions.

- An illustrative example with the following assumption:
  - ✓ A program with 80% of executed instructions being simple and 20% complex.
  - ✓ CISC: simple instructions take 4 cycles, complex instructions take 8 cycles; cycle time is 100 ns.
  - ✓ RISC: simple instructions are executed in one cycle; complex operations are implemented as a sequence of instructions (14 instructions on average); cycle time is 75 ns.

How much time takes a program of 1 000 000 instructions?

- CISC: $(10^6*0.80*4 + 10^6*0.20*8)*10^{-7} = 0.48$ s
- RISC: $(10^6*0.80*1 + 10^6*0.20*14)*0.75*10^{-7} = 0.27$ s

- Load-and-store architecture
  - ✓ Only LOAD and STORE instructions reference data in memory.
  - ✓ All other instructions operate only with registers (are register-to-register instructions).

- Only a few simple addressing modes are used.
  - ✓ Ex. register, direct, register indirect, displacement.

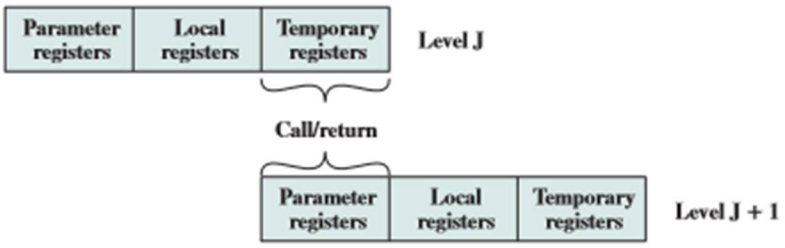
- Instructions are of fixed length and uniform format.
  - ✓ Loading and decoding of instructions are simple and fast.
  - ✓ It is not needed to wait until the length of an instruction is known in order to start decoding it.
  - ✓ Decoding is simplified because the opcode and address fields are located in the same position for all instructions.

- A large number of registers is available.
  - ✓ Variables and intermediate results can be stored in registers and do not require repeated loads and stores from/to memory.
  - ✓ All local variables of procedures and the passed parameters can be stored in registers.

- The large number of registers is due to that the reduced complexity of the processor leaves silicon space on the chip to implement them.
  - ✓ This is usually not the case with CISC machines.

Register Windows:

- A large number of registers is usually very useful.
- However, if contents of all registers must be saved at every procedure call, more registers mean longer delay.
- A solution to this problem is to divide the register file into a set of fixed-size windows.
  - ✓ Each window is assigned to a procedure.
  - ✓ Windows for adjacent procedures are overlapped to allow parameter passing.



Main Advantages of RISC:

- Best support is given by optimizing most used and most time consuming architecture aspects.
  - ✓ Frequently executed instructions.
  - ✓ Memory reference.
  - ✓ Procedure call/return.
  - ✓ Pipeline design.
- Less design complexity, reducing design cost, and reducing the time between designing and marketing.

RISC vs. CISC:

- CISC has more **instrution sets** than RISC so instrution cycle in CISC is more complex than RISC.
- CISC is in hardware and RISC is in software.
- **Execution time:**
  - ✓ CISC has multiple number of cycle per instrution
  - ✓ RISC has 1 cycle per instrution
- CISC CPUs are more larger than RISC CPUs.
  - ✓ RISC CPUs can but in a single chip which make RISC light and portable.
- **Speed:**
  - ✓ CISC CPU take little while to process the instructions
  - ✓ RISC can only perform simple task quicker than CISC.
- Most recent processors are not typical RISC or CISC, but combine advantages of both approaches.
  - ✓ Called EPIC (Explicity Parallal Instrution Computing).