Chapter 2:
Computer Systems Organization

2.1 Processors

Organization, Instruction Cycle, RISC vs. CISC, Design Principles, Parallelism

2.2 Primary Memory

Addressing, Endianness, Parity, Cache, Packaging

2.3 Secondary Memory

Hierarchies, Disk Architectures

2.4 Input / Output

Busses, Character Codes

CPU: Central Processing Unit

- Control Unit

  Controls the operation of all system components via control signals in order to fetch, interpret and execute program instructions.
- Arithmetic Logic Unit – ALU

  Performs arithmetic and logic operations on its inputs, e.g. ADD, AND
- Registers

  Store (singular) data or addresses.
  - Special Purpose: **PC**, **IR** (others) have a dedicated function
  - General Purpose: Under program control
- Bus

  Collection of parallel wires for transmitting information (data, addresses, control signals) between computer system components.
  - System Bus: Connects main components- CPU, Memory and I/O Devices
  - CPU Busses: Connect CPU components- ALU, Registers and Cache

Machine-Level Instructions

- Register-Memory
  - Data Move: Transfer Data between Memory Words and Processor Registers, LOAD, STORE

  The **ALU** inputs are sourced from registers and **ALU** output is stored into a register. But programs typically maintain their data in variables in memory.

- Register-Register
  - Operate: Perform Arithmetic and Logic, e.g. ADD, AND

  Source and destination operands are general purpose registers.

  - Control: Alter Execution Sequence, e.g. BRANCH, CALL

  These instructions overwrite special purpose register **PC** with an address constructed from information in **PC** and **IR** or general purpose registers.

The Instruction Cycle

- Instruction Fetch Phase

  Fetch next instruction from memory into **IR**

  Update **PC** to locate the next instruction

- Decode Phase

  Determine the types of the operation and its operands

- Evaluate Address Phase *(Register-Memory only)*

  Evaluate the effective address of a memory operand

- Execute Phase *(Register-Register only)*

  Source the **ALU** inputs from **CPU** registers

  Perform the **ALU** operation identified at the Decode Phase

  Write **ALU** output to a **CPU** register

- Data Transfer Phase *(Register-Memory only)*

  Read the data from memory into a **CPU** register (LOAD)

  Write the data from a **CPU** register into a memory word (STORE)

An Interpreter (part 1)

```
public abstract class Interpreter
{
   private boolean RUN; //Run Flag

   //Special Purpose Registers (not all showm)
   protected int PC;     //Program Counter
   protected int IR;     //Instruction Register

   protected int[] GPR; //General Purpose Registers
   protected int[] RAM; //Random Access Memory

   //Constructor
   public Interpreter(int n_GPR, int n_RAM)
   {
      this.GPR = new int[n_GPR];
      this.RAM = new int[n_RAM];
      while ( true ) {
         this.loadProgram();
         this.interpret();
      }
   }
```

An Interpreter (part 2)

```
   private void interpret()
   {
      while ( this.RUN ) {
         fetchInstruction();  //Update IR, PC
         decodeInstruction(); //Interpret IR
         evaluateAddress();   //Evaluate EA
         execute();         //ALU operation
         dataTransfer();    //GPR <--> RAM[EA]
      }
   }

   protected abstract void loadProgram();
   protected abstract void fetchInstruction();
   protected abstract void decodeInstruction();
   protected abstract void evaluateAddress();
   protected abstract void execute();
   protected abstract void dataTransfer();
}
```

Hardware Instructions

- Hardware and Software are logically equivalent
  - Instructions can be implemented in hardware or software.
  - Software instructions are interpreted by translation into lower level instructions. At some lower level, there must be hardware instructions.
  - A language may be hybrid – part hardware, part software.
- Advantages of Hardware Instructions
  - Speed – direct execution is faster than interpretation
- Disadvantages of Hardware Instructions
  - Cost – *complex instructions require lots of hardware*
  - Complexity – *complex instructions are hard to implement*
  - Inflexibility – *very difficult to modify or re-implement*

Software Instructions

- Hardware and Software are logically equivalent
  - Instructions can be implemented in hardware or software.
  - Software instructions are interpreted by translation.
- Advantages
  - Economy – complex instructions can be implemented via simple low-level instructions; no expensive hardware.
  - Extendibility – easy to add new instructions.
  - Structured Design – new instructions can be tested and "*debugged*" even after deployment.
  - Scalability – facilitates "*downward compatibility*" (360).
- Disadvantages
  - Speed – interpretation is slower than direct execution.

Microinstructions

- A machine language instruction is performed as a series of data path operations; each transfers data between two registers.
  Example: LOAD REG, X_address

  MAR ← PC
  PC ← PC + 1
  MDR ← memory[MAR]
  IR ← MDR
  MAR ← X_address
  MDR ← memory[MAR]
  REG ← MDR

- When coupled with control signals each data path operation is described as a microinstruction, an atomic data path operation.
- The job of the Control Unit is to sequence the microinstructions

LC-3 Microinstructions

	GateMARMUX	MARMUX Select	GatePC	LD.PC	PCMUX_1	PCMUX_2	LD.IR		LD.MAR	MEM.EN	MEM.RW	MEM.MR	LD.MDR	GateMDR				
addr	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
addr	0	0	1	1	1	0	0	0	1	0	0	0	0	0	0	0	0	0
addr	0	0	0	0	0	0	0	0	0	1	1	0	1	0	0	0	0	0
addr	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0
addr	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.

Microprogramming

- Microprogramming is the technique of implementing the functions of the control unit in software.
- Microprogramming is a software alternative to hardwiring the control unit.
- A microprogram comprises sequences of microinstructions for each machine language instruction.
- The microprogram is stored into a ROM (control store). It functions as an interpreter – for each machine language instruction, it issues the appropriate microinstructions.
- Microprogramming was the predominant implementation of the 1960's and 1970's – IBM System 360, VAX 11's.

CISC
Complex Instruction Set Computer

- The desire for more "powerful" computers led to more complex instructions, e.g.
  - Integer multiplication and division
  - Floating-point arithmetic instructions
  - Function call and return
  - Indexing and indirect addressing
  - Interrupt handling
- Increased demand for "lower-cost" computers was met by implementing complex instructions via multiprogramming
- Availability of fast control stores facilitated cost vs. speed trade-off in designing compatible families of computers.

RISC
Reduced Instruction Set Computer

- Emergence of VLSI technology reduced the dependence on multiprogramming.
- Development of parallelism strategies reduced the reliance on complex instructions for speed.
- Increased speed of new technologies led to processor designs intended to execute more simple instructions faster.

RISC Design Principles

- Hardwire the most frequently used instructions.
- Utilize parallelism to maximize instruction completion rate:
  - Instruction-Level, e.g. pipelining
  - Processor-Level, e.g. multiprocessors
- Design instructions formats to be simple and regular:
  - fixed length
  - minimum operand fields
  - few formats
- Minimize memory references
  - operate and control instructions should be register-register
  - only data move instructions should reference memory
- Provide a large number of general purpose registers

Instruction-Level Parallelism

- Figure 2.4: A Five-Stage Pipeline
- Figure 2.5: Dual Pipelines
- Figure 2.6: Multiple Functional Units

Processor-Level Parallelism

- Figure 2.7: SIMD (Single Instruction Multiple Data)
- Figure 2.8: Multiprocessors