Designing Computers

• All computers more or less based on the same basic design, the Von Neumann Architecture!

The Von Neumann Architecture

• Model for designing and building computers, based on the following three characteristics:
  1) The computer consists of four main sub-systems:
     • Memory
     • ALU (Arithmetic/Logic Unit)
     • Control Unit
     • Input/Output System (I/O)
  2) Program is stored in memory during execution.
  3) Program instructions are executed sequentially.
The Von Neumann Architecture

- Memory: Store data and program.
- Processor (CPU): Execute program, do arithmetic/logic operations requested by program.
- Control Unit: Communicate with "outside world", e.g., screen, keyboard, storage devices.
- ALU: Computed with MAR, MDR.
- Input-Output: Communicate with "outside world", e.g., screen, keyboard, storage devices.

Structure of the Memory Subsystem

- Fetch(address):
  - Load address into MAR.
  - Decode the address in MAR.
  - Copy the content of memory cell with specified address into MDR.
- Store(address, value):
  - Load the address into MAR.
  - Load the value into MDR.
  - Decode the address in MAR.
  - Copy the content of MDR into memory cell with the specified address.

Implementation of the Memory Subsystem

- MAR:
- 4-to-16 Decoder circuit:
- Memory Address:
  - 0000 (8)
  - 0001 (1)
  - 0010 (2)
  - ...
**CACHE - Modern addition**

- High-speed memory, integrated on the CPU
  - Ca. 10 times faster than RAM
  - Relatively small (128-256K)
- Stores data most recently used
  - Principle of Locality
- When CPU needs data:
  - First looks in the cache, only if not there, then fetch from RAM.
  - If cache full, new data overwrites older entries in cache.

**I/O Subsystem: Hard-Drives**

- Uses magnetic surfaces to store the data.
  - Each surface has many circular tracks.
  - Each track consists of many sectors.

The surfaces rotate at a high speed
Typically ~7000 rev/min
The read/write arm moves:
  back and forth to locate a track

**Hard-Drive**

- Uses magnetic surfaces to store the data.
  - Each surface has many circular tracks.
  - Each track consists of many sectors.

The surfaces rotate at a high speed
Typically ~7000 rev/min
The read/write arm moves:
  back and forth to locate a track
Disk Access Time

- The time it takes to read/write data to a disk, consists of:
  - Seek time
    - The time it takes to position the read/write head over correct track (depends on arm movement speed).
  - Latency
    - The time waiting for the beginning of the desired sector to get under the read/write head (depends on rotation speed).
  - Transfer time
    - The time needed for the sector to pass under the read/write head (depends on rotation speed).
  - Disk Access Time = Seek time + Latency + Transfer time

- Measure worst, best, and average case. (Example: p. 189)

Structure of the ALU

- Registers:
  - Very fast local memory cells, that store operands of operations and intermediate results.
  - CCR (condition code register), a special purpose register that stores the result of <, = , > operations.
- ALU circuitry:
  - Contains an array of circuits to do mathematical/logic operations.
- Bus:
  - Data path interconnecting the registers to the ALU circuitry.

Implementation of the ALU

- Every circuit produces a result but only the desired one is selected.
Structure of the Control Unit

- **PC** (Program Counter):
  - stores the address of next instruction to fetch
- **IR** (Instruction Register):
  - stores the instruction fetched from memory
- **Instruction Decoder**:
  - Decodes instruction and activates necessary circuitry

**Machine Language Instructions**

- A machine language instruction consists of:
  - Operation code, telling which operation to perform
  - Address field(s), telling the memory addresses of the values on which the operation works.
- Example: ADD X, Y (Add content of memory locations X and Y, and store back in memory location Y).
- Assume: opcode for ADD is 9, and addresses X=99, Y=100

<table>
<thead>
<tr>
<th>Opcode (8 bits)</th>
<th>Address 1 (16 bits)</th>
<th>Address 2 (16 bits)</th>
</tr>
</thead>
<tbody>
<tr>
<td>000001001</td>
<td>0000000001100011</td>
<td>0000000001100100</td>
</tr>
</tbody>
</table>

**Implementation of the Control Unit**
How does this all work together?

Program Execution:
- PC is set to the address where the first program instruction is stored in memory.
- Repeat until HALT instruction or fatal error
  - Fetch instruction
  - Decode instruction
  - Execute instruction
  - End of loop

Program Execution (cont.)
- Fetch phase
  - PC -> MAR (put address in PC into MAR)
  - Fetch signal (signal memory to fetch value into MDR)
  - MDR -> IR (move value to Instruction Register)
  - PC + 1 -> PC (increase address in program counter)
- Decode Phase
  - IR -> Instruction decoder (decode instruction in IR)
  - Instruction decoder will then generate the signals to activate the circuitry to carry out the instruction
Program Execution (cont.)

- Execute Phase
  - Differs from one instruction to the next.
- Example:
  - LOAD X (load value in addr. X into register)
    - IR_address -> MAR
    - Fetch signal
    - MDR -> R
  - ADD X
    - left as an exercise

Instruction Set for Our Von Neumann Machine

<table>
<thead>
<tr>
<th>Opcode</th>
<th>Operation</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>LOAD X</td>
<td>CON(X) -&gt; R</td>
</tr>
<tr>
<td>0001</td>
<td>STORE X</td>
<td>R -&gt; CON(X)</td>
</tr>
<tr>
<td>0010</td>
<td>CLEAR X</td>
<td>0 -&gt; CON(X)</td>
</tr>
<tr>
<td>0011</td>
<td>ADD X</td>
<td>R + CON(X) -&gt; R</td>
</tr>
<tr>
<td>0100</td>
<td>INCREMENT X</td>
<td>CON(X) + 1 -&gt; CON(X)</td>
</tr>
<tr>
<td>0101</td>
<td>DECREMENT X</td>
<td>CON(X) - 1 -&gt; CON(X)</td>
</tr>
<tr>
<td>0111</td>
<td>COMPARE X</td>
<td>CON(X) - 1 -&gt; CON(X)</td>
</tr>
<tr>
<td>0101</td>
<td>JUMPGT X</td>
<td>Get next instruction from memory loc. X</td>
</tr>
<tr>
<td>1000</td>
<td>JUMP X</td>
<td>Get next instruction from memory location X</td>
</tr>
<tr>
<td>1001</td>
<td>JUMPPx X</td>
<td>Get next instruction from memory loc. X</td>
</tr>
<tr>
<td>1101</td>
<td>IN X</td>
<td>Input an integer value and store in X</td>
</tr>
<tr>
<td>1110</td>
<td>OUT X</td>
<td>Output, in decimal notation, content of mem. loc. X</td>
</tr>
<tr>
<td>1111</td>
<td>HALT</td>
<td>Stop program execution</td>
</tr>
</tbody>
</table>