Recap – Computer Architecture

• Computer Organization

- The von Neumann architecture
- Same storage device for both instructions and data
- The von Neumann Bottleneck
Recap – Computer Architecture

- Device Controllers
  - Memory mapped I/O
  - Direct Memory Access (DMA)

- Instruction Set
  - Data transfer operations
  - Arithmetic / logic operations
  - Control flow instructions

The architecture of the machine described in Appendix C

CPU

- Arithmetic / Logic Unit
  - Registers
    - 0
    - 1
    - 2
    - ...
    - F

- Control Unit
  - Program counter
  - Instruction register

Main memory

- Address Cells
  - 00
  - 01
  - 02
  - 03
  - ...
  - FF

Outside world

Diagram showing device controllers connected to the CPU, memory, video controller, DVD controller, USB controller, and network controller.
Program Execution
“The machine cycle”

1. **Fetch** next instruction from memory (as indicated by the program counter) then increment the program counter.

2. **Decode** the bit pattern in the instruction register.

3. **Execute** the action requested by the instruction in the instruction register.

The composition of an instruction for the machine in Appendix C

4-digit hexadecimal form: 3 5 A 7

16-bit pattern: 0011 0101 1010 0111

- **Op-code:** 0011
- **Operand:** 0101 1010 0111
1. At the beginning of the fetch step the instruction starting at address A0 is retrieved from memory and placed in the instruction register.
Performing the fetch step of the machine cycle II

2. Then the program counter is incremented so that it points to the next instruction.

Decoding the instruction 35A7

Op-code 3 means to store the contents of a register in a memory cell.

This part of the operand identifies the register whose contents are to be stored.

This part of the operand identifies the address of the memory cell that is to receive data.
Mnemonics

- It is hard to remember many numbers
- Use words associated with the numbers

```
store R5, [0xA7] <=> 35A7
```

<table>
<thead>
<tr>
<th>db</th>
<th>org</th>
<th>immediate load</th>
<th>direct load</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>load reg, addr</td>
<td>load reg, [addr]</td>
</tr>
<tr>
<td>indirect load</td>
<td>direct store</td>
<td>indirect store</td>
<td>move</td>
</tr>
<tr>
<td>integer addition</td>
<td>floating point addition</td>
<td>bitwise OR</td>
<td>bitwise AND</td>
</tr>
<tr>
<td>addi reg, reg, reg</td>
<td>addf reg, reg, reg</td>
<td>or reg, reg, reg</td>
<td>and reg, reg, reg</td>
</tr>
<tr>
<td>bitwise XOR</td>
<td>rotate right</td>
<td>jmp</td>
<td>jmpLE</td>
</tr>
<tr>
<td>xor reg, reg, reg</td>
<td>ror reg, num</td>
<td>jmp addr</td>
<td>jmpLE reg&lt;=R0,addr</td>
</tr>
<tr>
<td>jmpEQ</td>
<td>halt</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Assembly Language I

- **Immediate load**
  
  \[
  \text{load} \quad \text{reg}, \text{number} \\
  \text{load} \quad \text{reg}, \text{label}
  \]

- **Direct load**
  
  \[
  \text{load} \quad \text{reg}, [\text{adr}]
  \]

- **Indirect load**
  
  \[
  \text{load} \quad \text{reg1}, [\text{reg2}]
  \]

- **Direct store**
  
  \[
  \text{store} \quad \text{reg}, [\text{adr}]
  \]

- **Indirect store**
  
  \[
  \text{store} \quad \text{reg1}, [\text{reg2}]
  \]

- **Unconditional jump**
  
  \[
  \text{jmp} \quad \text{adr}
  \]

- **Origin**
  
  \[
  \text{org} \quad \text{adr}
  \]

- **Data byte**
  
  \[
  \text{db} \quad \text{dataitem}
  \]

Program that switches the contents in memory location 0x20 and 0x10

\[
\begin{align*}
\text{jmp} & \quad \text{Start} \\
\text{org} & \quad 0x30; \\
\text{Start:} \\
\text{load} & \quad \text{R0}, 0x10; \\
\text{load} & \quad \text{R1}, [\text{R0}]; \\
\text{load} & \quad \text{R2}, [\text{new\_number}]; \\
\text{Store} & \quad \text{R1}, [\text{new\_number}]; \\
\text{Store} & \quad \text{R2}, [\text{R0}]; \\
\text{halt}; \\
\text{org} & \quad 0x20; \\
\text{new\_number} : & \quad \text{db} \ 10d \\
\text{org} & \quad 0x10; \\
\text{old\_number} : & \quad \text{db} \ 25d;
\end{align*}
\]
CQ I

1. Both Contain 0
2. 0xfe contains 0, 0xff contains 04
3. 0xfe contains 0, 0xff contains 05
4. I don’t know

Assembly II

- **bitwise or**
  
  or reg1, reg2, reg3

- **bitwise and**
  
  and reg1, reg2, reg3

- **bitwise exclusive or**
  
  xor reg1, reg2, reg3
Program to demonstrate the basic bit-wise constructs

```plaintext
load R1, 00100110b;
load R2, 11111111b;
load R0, 00000000b;
and R3, R1, R2;
and R4, R1, R0;
or R5, R1, R2;
or R6, R1, R0;
xor R7, R1, R2;
halt;
```

CQ II

1. 1001
2. 0000
3. 0110
4. I don’t know