Introduction to Computers and Programming

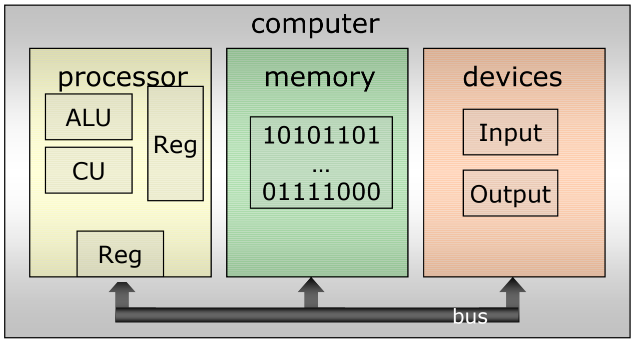
Prof. I. K. Lundqvist

Reading: B pp. 505-507, Machine language handout

Lecture 7
Sept 16 2003

Recap – Computer Architecture

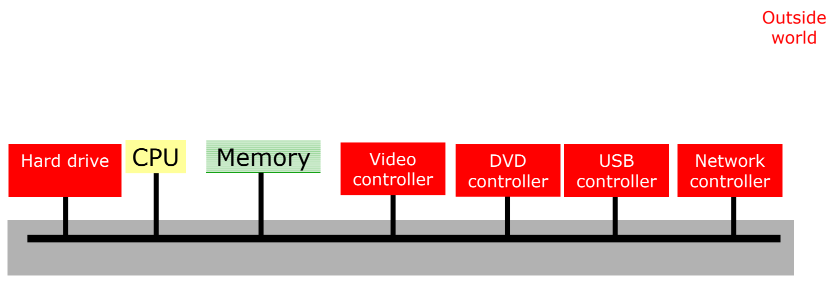
- Computer Organization



  - The von Neumann architecture
  - Same storage device for both instructions and data
  - The von Neumann Bottleneck

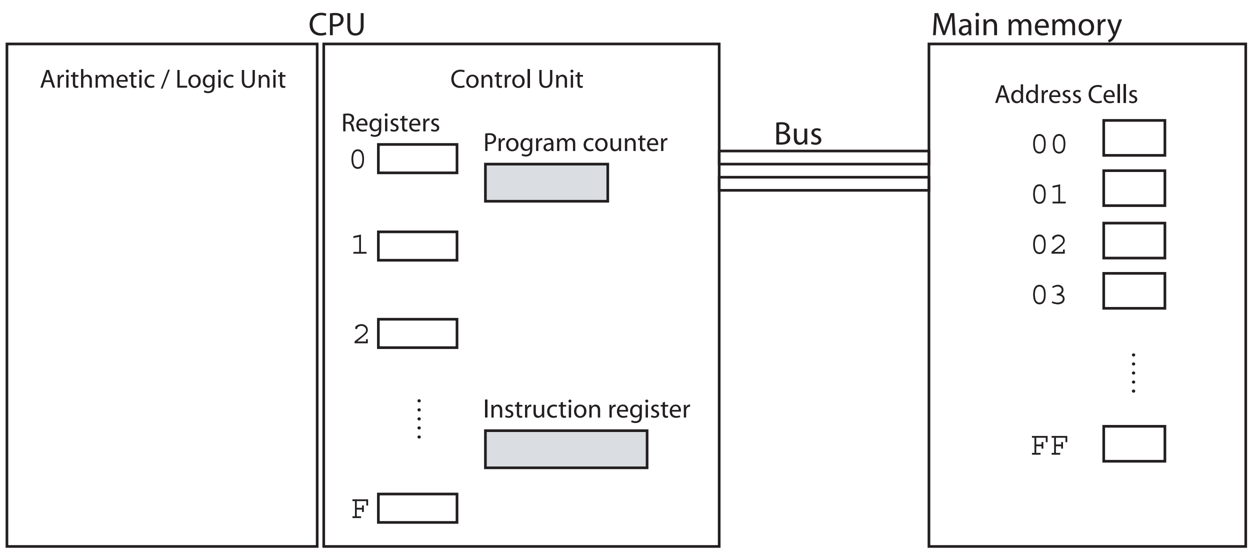
Recap – Computer Architecture

- Device Controllers



  - Memory mapped I/O
  - Direct Memory Access (DMA)
- Instruction Set
  - Data transfer operations
  - Arithmetic / logic operations
  - Control flow instructions

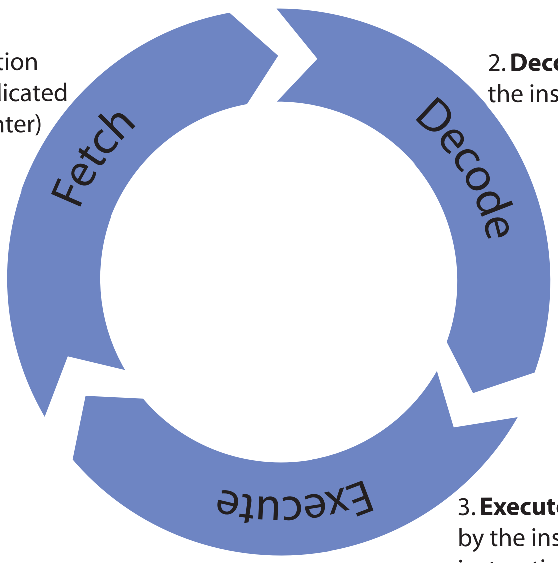
The architecture of the machine described in Appendix C



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.



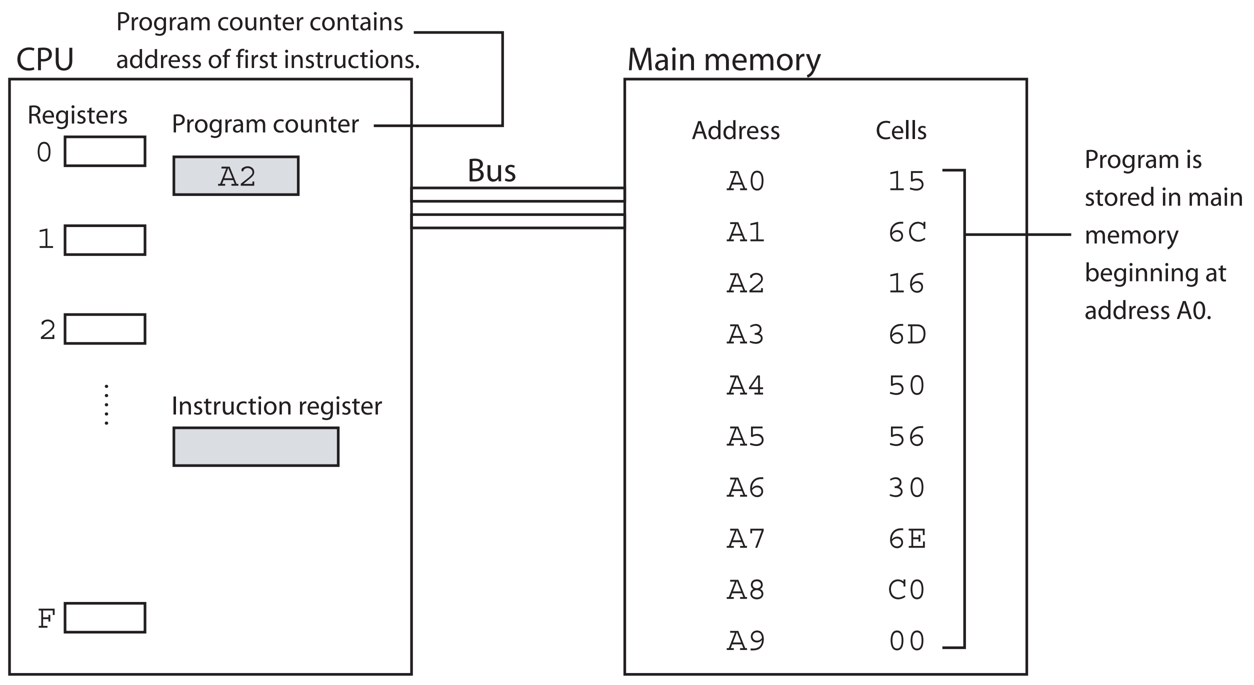
3. **Execute** the action requested by the instruction in the instruction register.

The composition of an instruction for the machine in Appendix C

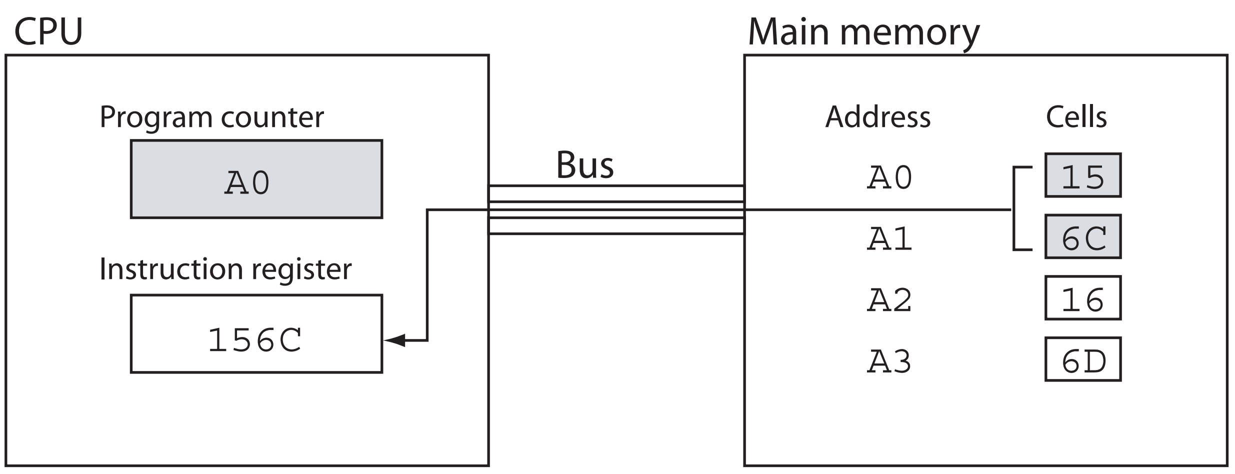
4-digit hexadecimal form	3	5	A	7
16-bit pattern	0011	0101	1010	0111

$$\underbrace{0011}_{\text{Op-code}}\ \underbrace{0101\ 1010\ 0111}_{\text{Operand}}$$

Stored Program

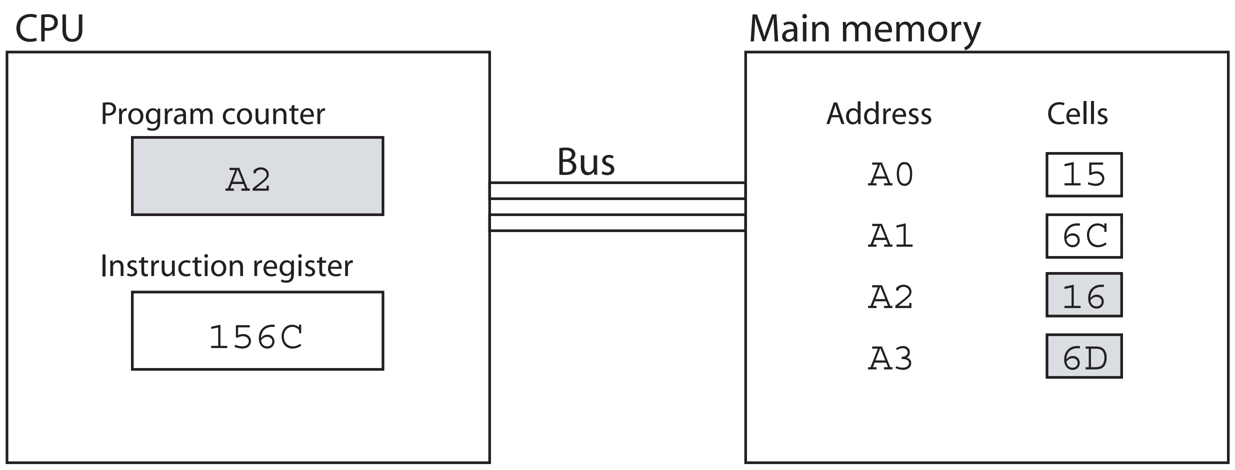


Performing the fetch step of the machine cycle I



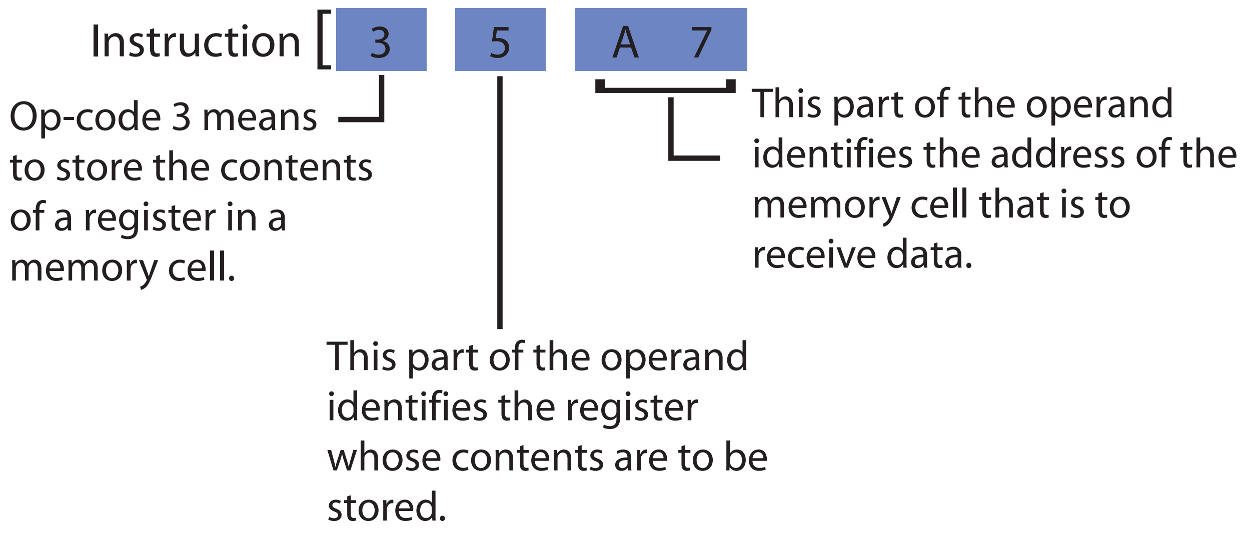
1. At the beginning of the fetch step the instruction starting at address A0 is retrieved from memory and placed in the instruction register.

Performing the fetch step of the machine cycle II



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

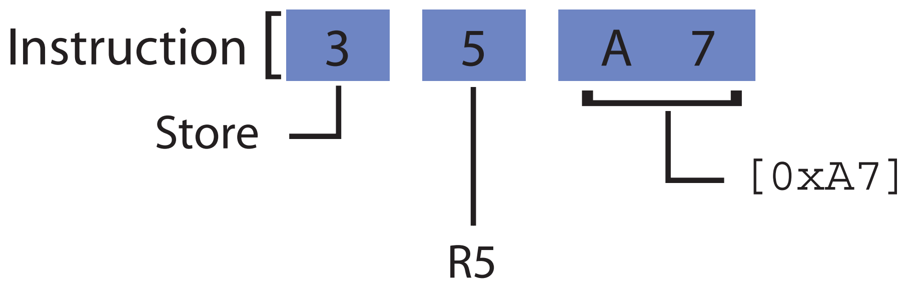
Decoding the instruction 35A7



Instruction
3
5
A 7
Op-code 3 means to store the contents of a register in a memory cell.
This part of the operand identifies the address of the memory cell that is to receive data.
This part of the operand identifies the register whose contents are to be stored.

Mnemonics

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



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

db	org	immediate load load reg, addr	direct load load reg, [addr]
indirect load load reg, [reg]	direct store store reg, [addr]	indirect store store reg, [reg]	move move reg1, reg2
integer addition addi reg, reg, reg	floating point addition addf reg, reg, reg	bitwise OR or reg, reg, reg	bitwise AND and reg, reg, reg
bitwise XOR xor reg, reg, reg	rotate right ror reg, num	jmp jmp addr	jmpLE jmpLE reg<=R0,addr
jmpEQ jmpEQ reg=R0,adr	halt		

Assembly Language I

- ***immediate load***
  ```
  load    reg,number
  load    reg,label
  ```
- ***direct load***
  ```
  load    reg,[adr]
  ```
- ***indirect load***
  ```
  load   reg1,[reg2]
  ```
- ***direct store***
  ```
  store   reg,[adr]
  ```
- ***indirect store***
  ```
  store reg1,[reg2]
  ```
- ***unconditional jump***
  ```
  jmp     adr
  ```
- ***origin***
  ```
  org     adr
  ```
- ***data byte***
  ```
  db      dataitem
  ```

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

```
jmp Start
org 0x30;
Start:
load  R0, 0x10;
load R1, [R0];
load R2, [new_number];
Store R1,[new_number];
Store R2, [R0];
halt;
org 0x20;
new_number : db 10d

org 0x10;
old_number : db 25d;
```

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

```
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