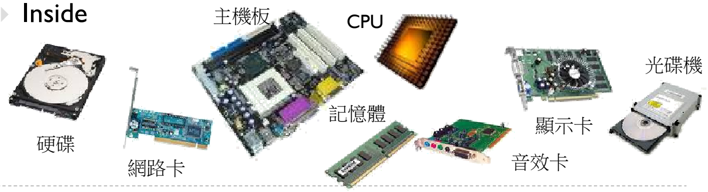
Computer

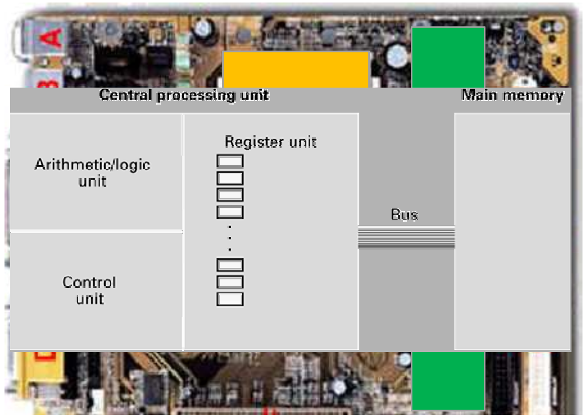
Your desktop

- From outside

- Inside



von Neumann architecture

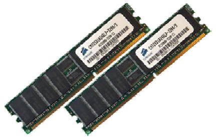


Data storage

- Physical objects that can store bits and retrieve bits can be a storage media.
- Volatile (temporary) memory:
  - DRAM, SRAM, SDRAM
- Non-volatile storage (massive storage)
  - Optical Systems: CD, DVD
  - Magnetic Systems: Hard disk, tape
  - Flash driver: iPod, Cell Phone, USB drivers…

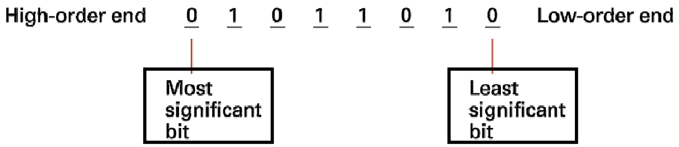
Memory

- Memory is used inside computers for temporary storages.
- They are often called RAMs
  - **R**andom **A**ccess **M**emory: data can be accessed in any order
  - Dynamic RAM (DRAM):
  - Synchronous DRAM (SDRAM)
  - Static RAM (SRAM)



Data storage unit

- To efficiently access data, computers use 8 bits (a byte) as a smallest storage unit.
- Some jargons for a byte
  - **Most significant bit:** at the high-order end
  - **Least significant bit:** at the low-order end



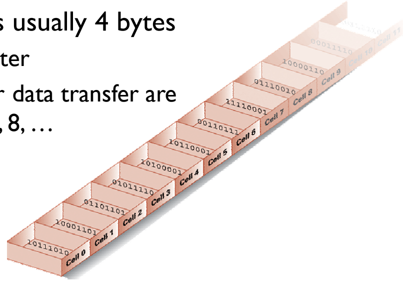
The size of data

- **Kilobyte:** 2^{10} bytes ≈ 10^3 bytes
  - Example: 3 KB ≈ 3 × 10^3 bytes
- **Megabyte:** 2^{20} bytes ≈ 10^6 bytes
  - Example: 3 MB ≈ 3 × 10^6 bytes
- **Gigabyte:** 2^{30} bytes ≈ 10^9 bytes
  - Example: 3 GB ≈ 3 × 10^9 bytes
- **Terabyte:** 2^{40} bytes ≈ 10^{12} bytes
  - Example: 3 TB ≈ 3 × 10^{12} bytes
- **Petabyte:** 2^{50} bytes ≈ 10^{15} bytes
  - Example: 3 PB ≈ 3 × 10^{15} bytes

Exa	10^{18}
Zetta	10^{21}
Yotta	10^{24}
Xona	10^{27}
Weka	10^{30}
Vunda	10^{33}
Uda	10^{36}
Treda	10^{39}
Sorta	10^{42}
Rinta	10^{45}
Quexa	10^{48}
Pepta	10^{51}
Ocha	10^{54}
Nena	10^{57}
Minga	10^{60}
Luma	10^{63}

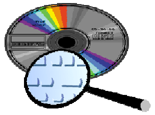
How data are stored in main memory?

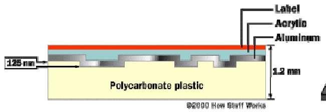
- Each storage unit in memory is numbered by an address so that data can be **stored** and **loaded**.
  - These numbers are assigned consecutively starting at zero.
  - Each cell contains a byte (8 bits)
- The basic transfer unit is usually 4 bytes
  - A **word** for 32bit computer
  - So the valid addresses for data transfer are the multiples of four: 0, 4, 8, …

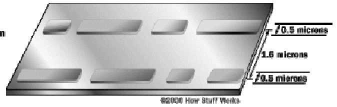


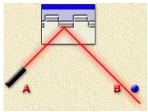
CD/DVD

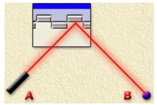
- CD: Compact Disk
- DVD: Digital Video Disk
  - Use bumps to represent 0/1





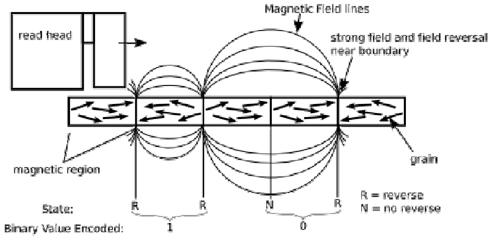




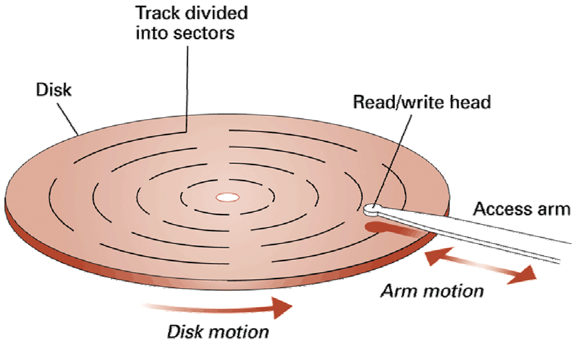


Hard disks (HDD)

- A hard platter holds the magnetic medium
  - Use magnetic field to represent 0/1

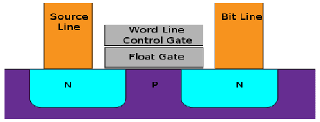


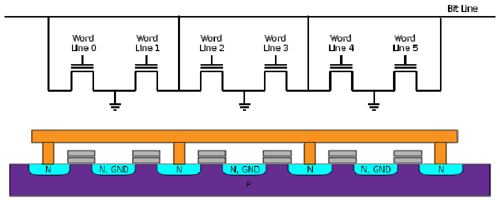
Some terms of hard disk



Flash memory

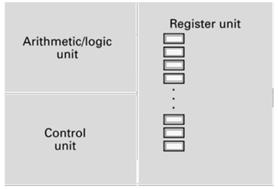
- Use electrical charge to represent 0/1





Inside CPU

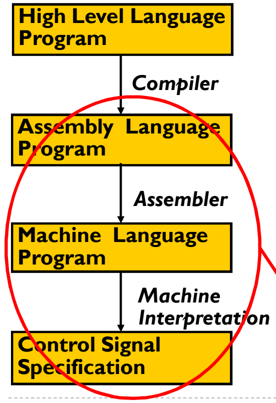
- **ALU** (arithmetic/logic unit)
  - Arithmetic operations: +, -, *, %, …
  - Logic operations: AND, OR, NOT,
- **Control unit**:
  - Control how and when the data and instruction flow.
- **Register**:
  - A temperate (small) storage for data to be processes



Stored program concept

- All desired binary functions can be implemented by gate circuits.
  - But each circuit can only carry out a single function.
- Stored program concept
  - Design a machine that can perform different functions
  - The machine should contain circuits to perform all basic functions
  - Use a 'program' to control the machine to perform different functions
  - Programs are stored for repeated use

Computer programs


High Level Language Program
Compiler
Assembly Language Program
Assembler
Machine Language Program
Machine Interpretation
Control Signal Specification

```
temp = v[k];
v[k] = v[k+1];
v[k+1] = temp;
```

You are learning it in CS1355

```
lw    $15,  0($2)
lw    $16,  4($2)
sw    $16,  0($2)
sw    $15,  4($2)
```

You will learn it in CS2410

```
0000 1001 1100 0110 1010 1111 0101 1000
1010 1111 0101 1000 0000 1001 1100 0110
1100 0110 1010 1111 0101 1000 0000 1001
0101 1000 0000 1001 1100 0110 1010 1111
```

This will be taught in CS4100

We are going to talk about those.

```
ALUOP[0:3] <= InstReg[9:11] & MASK
```

Example: a = b + c

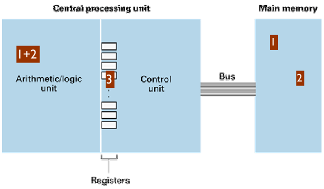
Step 1. Get one of the values to be added from memory and place it in a register.

Step 2. Get the other value to be added from memory and place it in another register.

Step 3. Activate the addition circuitry with the registers used in Steps 1 and 2 as inputs and another register designated to hold the result.

Step 4. Store the result in memory.

Step 5. Stop.

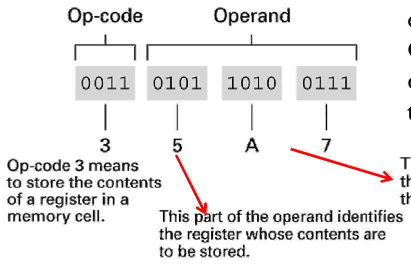


Virtual machine used in the textbook

- ISA: instruction set architecture
  - 16 general-purpose registers (numbered 0 through F)
  - Each register is of one byte (8 bits) long
  - Main memory has 256 memory cells (8 bits)
  - Machine language is of two bytes (16 bits) long
    - op-code field -> leftmost 4 bits (12 instructions)
    - operand field -> the remaining 12 bits
  - Integer is of 1 byte (8 bits) in 2's complement format
  - Floating-point values are stored in the 8-bit format

Instruction format

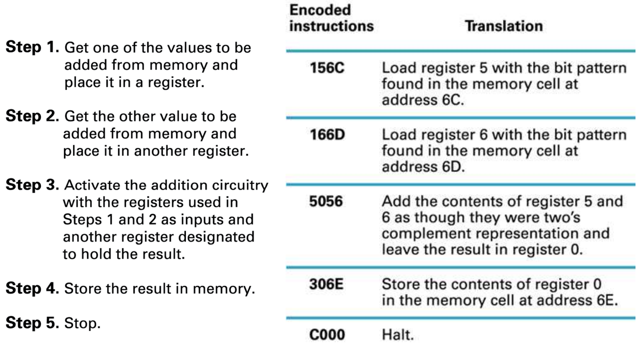
- Store the data in register 5 to memory cell at address A7


Op-code
Operand
0011
0101
1010
0111
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 register whose contents are to be stored.
This part of the operand identifies the address of the memory cell that is to receive data.

Op-code: Specifies which operation to execute

Operand: Gives more detailed information about the operation

Represented by instructions



	Encoded instructions	Translation
Step 1. Get one of the values to be added from memory and place it in a register.	**156C**	Load register 5 with the bit pattern found in the memory cell at address 6C.
Step 2. Get the other value to be added from memory and place it in another register.	**166D**	Load register 6 with the bit pattern found in the memory cell at address 6D.
Step 3. Activate the addition circuitry with the registers used in Steps 1 and 2 as inputs and another register designated to hold the result.	**5056**	Add the contents of register 5 and 6 as though they were two's complement representation and leave the result in register 0.
Step 4. Store the result in memory.	**306E**	Store the contents of register 0 in the memory cell at address 6E.
Step 5. Stop.	**C000**	Halt.

Instruction types

- Data Transfer
  - Copy data between CPU and main memory
  - E.g., LOAD, STORE, device I/O,
- Arithmetic/Logic
  - Use existing data values to compute a new value
  - E.g.,AND, OR, XOR, SHIFT, ROTATE, etc.
- Control
  - Direct the execution of the program
  - E.g., JUMP, BRANCH, JNE (conditional jump),

Instruction types

Encoded instructions	Translation
156C	Load register 5 with the bit pattern found in the memory cell at address 6C.
166D	Load register 6 with the bit pattern found in the memory cell at address 6D.
5056	Add the contents of register 5 and 6 as though they were two's complement representation and leave the result in register 0.
306E	Store the contents of register 0 in the memory cell at address 6E.
C000	Halt.

Data transfer

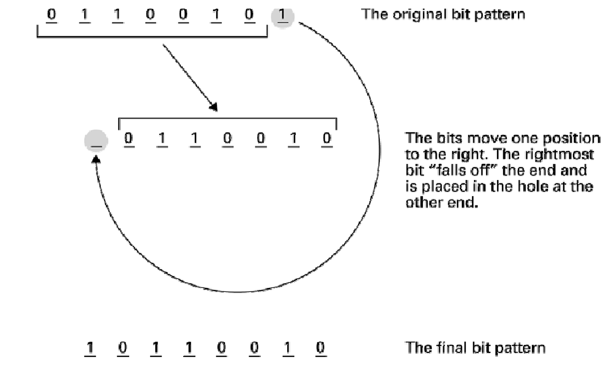
Data transfer

Arithmetic/Logic

Data transfer

Control

Rotate operation



The original bit pattern

The bits move one position to the right. The rightmost bit "falls off" the end and is placed in the hole at the other end.

The final bit pattern

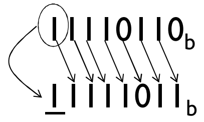
Figure 2.12 Rotating the bit pattern 65 (hexadecimal) one bit to the right

Shift operation

- Circular shift (rotation)
- Logical shift
  - Filling the hole with bit 0
  - Original: 00000101_b ➔ 5_d
  - After 1 left shifting: 00001010_b ➔ 10_d
  - After 2 left shifting: 00010100_b ➔ 20_d
- Arithmetic shift
  - Shifts that leaves the sign bit unchanged

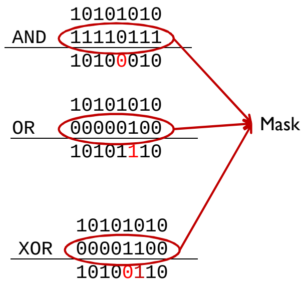
Arithmetic shift

- The two's complement of 00001010_b (10_d) is 11110110_b (-10_d)
- Want to use right shift to perform -10/2=-5,
  - 11110110_b >> 1 = 01111011_b = ?
  - We want the first bit to be 1. (11111011_b=-5)
- Arithmetic shift
  - Copy the first bit



Masking

- AND, OR, XOR can be used for masking
- Example: bit operations on 10101010_b
  - Set the 4th bit to 0
  - Set the 3rd bit to 1
  - Invert the 3rd and the 4th bit



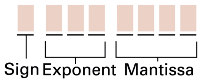
```
    10101010
AND 11110111
    10100010

    10101010
OR  00000100
    10101110

    10101010
XOR 00001100
    10100110
```

Examples of using masks

- Ex1: the floating point,
  - Design masks to retrieve sign, exponent, and mantissa.
  - Design a mask to set sign.

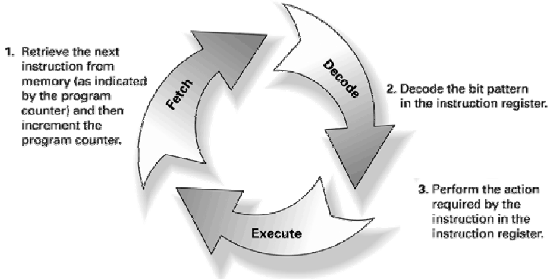


- Ex2: The ASCII code
  - Design a mask to convert capital letters to small letters or vice versa

A	1000001	a	1100001
B	1000010	b	1100010
C	1000011	c	1100011
D	1000100	d	1100100
E	1000101	E	1100101

Program execution cycle

- Program execution has 3 stages: Fetch, Decode, Execute.



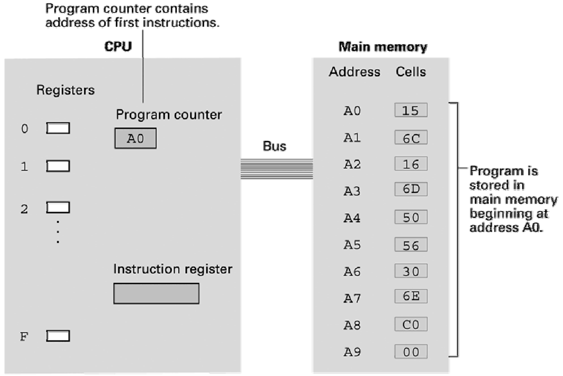
1. Retrieve the next instruction from memory (as indicated by the program counter) and then increment the program counter.

2. Decode the bit pattern in the instruction register.

3. Perform the action required by the instruction in the instruction register.

- Suppose the execution of each stage takes 1 clock cycle.
  - A 2.0GHz CPU has 2×10^{12} clock cycles per second.

How to "run" a program?

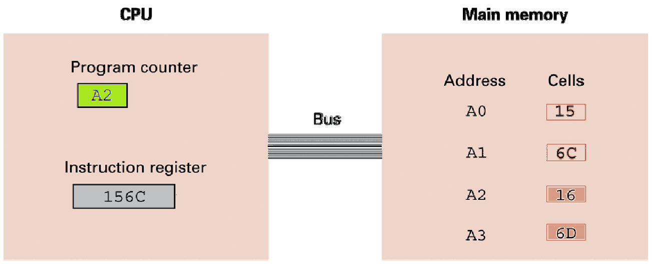


Program counter contains address of first instructions.

Program is stored in main memory beginning at address A0.

Figure 2.10 The program from Figure 2.7 stored in main memory

Instruction fetch

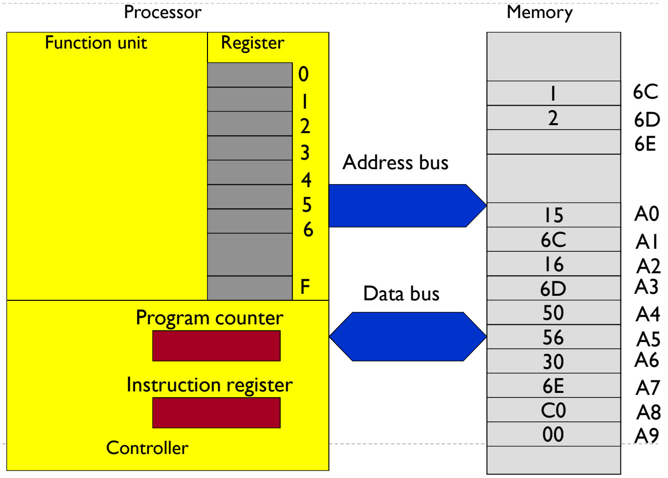


Address	Cells
A0	15
A1	6C
A2	16
A3	6D

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

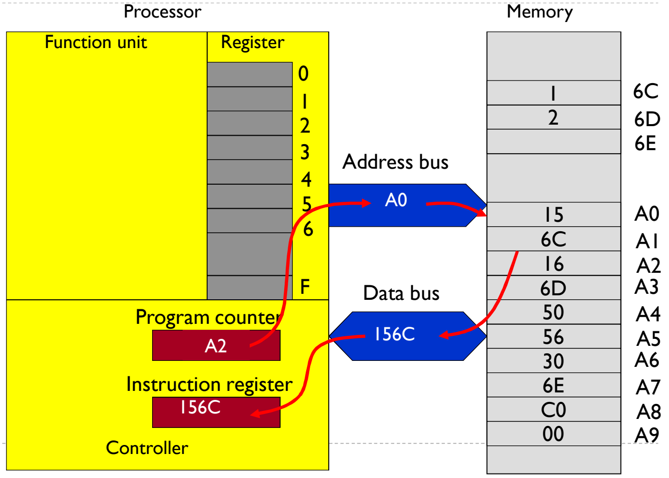
Figure 2.11 Performing the fetch step of the machine cycle

Processor architecture



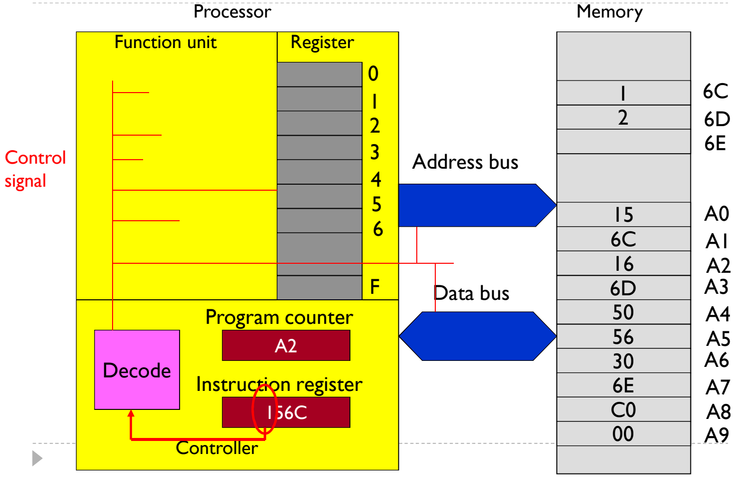
1	6C
2	6D
	6E
15	A0
6C	A1
16	A2
6D	A3
50	A4
56	A5
30	A6
6E	A7
C0	A8
00	A9

Fetch instruction 1



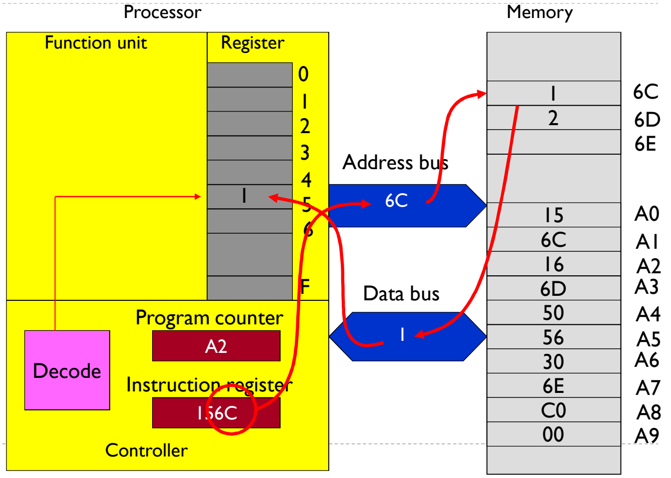
1	6C
2	6D
	6E
15	A0
6C	A1
16	A2
6D	A3
50	A4
56	A5
30	A6
6E	A7
C0	A8
00	A9

Decode instruction 1



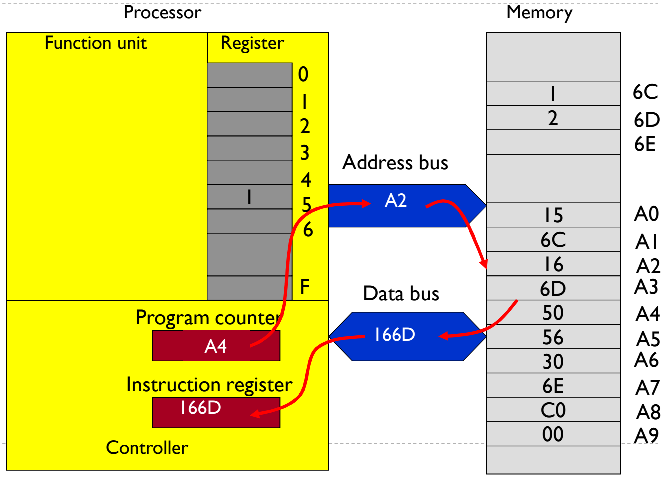
1	6C
2	6D
	6E
15	A0
6C	A1
16	A2
6D	A3
50	A4
56	A5
30	A6
6E	A7
C0	A8
00	A9

Execute instruction 1



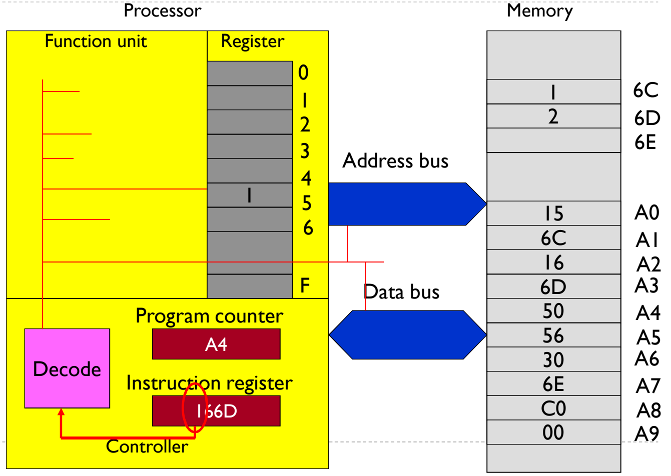
1	6C
2	6D
	6E
15	A0
6C	A1
16	A2
6D	A3
50	A4
56	A5
30	A6
6E	A7
C0	A8
00	A9

Fetch instruction 2



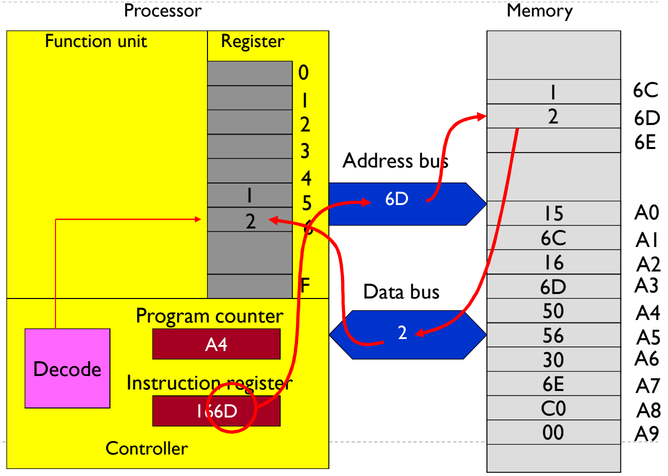
1	6C
2	6D
	6E
15	A0
6C	A1
16	A2
6D	A3
50	A4
56	A5
30	A6
6E	A7
C0	A8
00	A9

Decode instruction 2



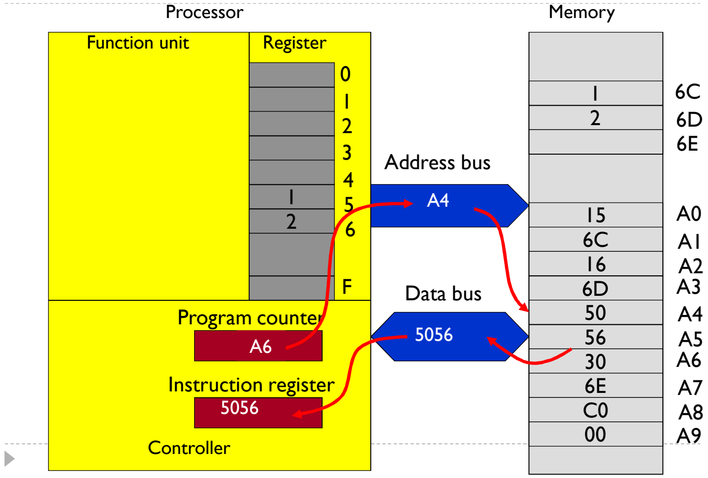
1	6C
2	6D
	6E
15	A0
6C	A1
16	A2
6D	A3
50	A4
56	A5
30	A6
6E	A7
C0	A8
00	A9

Execute instruction 2

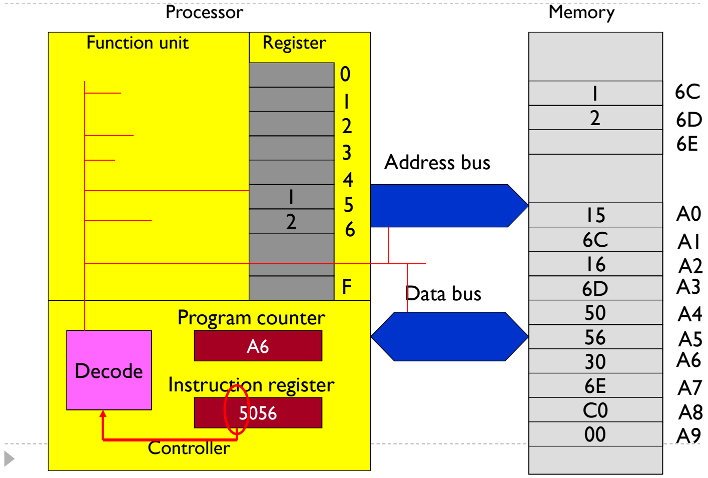


1	6C
2	6D
	6E
15	A0
6C	A1
16	A2
6D	A3
50	A4
56	A5
30	A6
6E	A7
C0	A8
00	A9

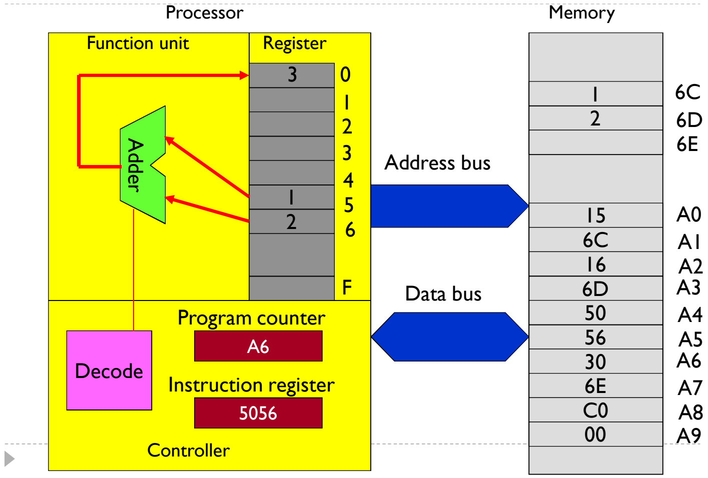
Fetch instruction 3



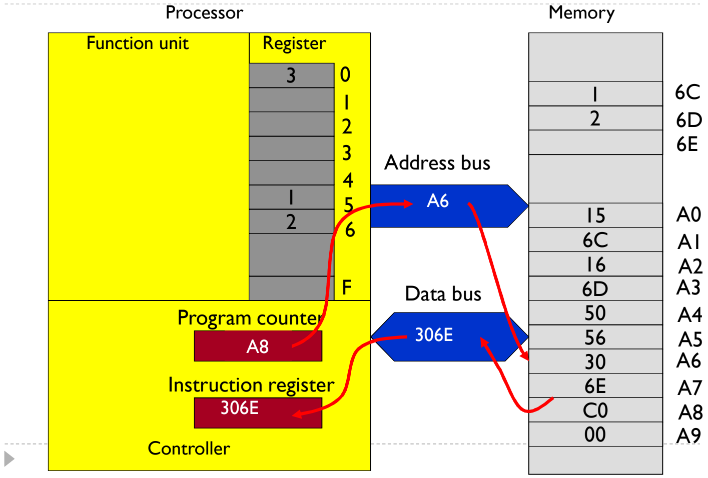
Decode instruction 3



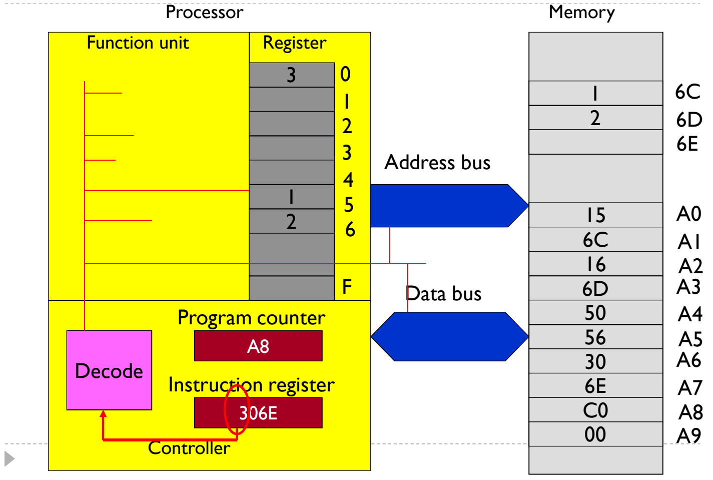
Execute instruction 3



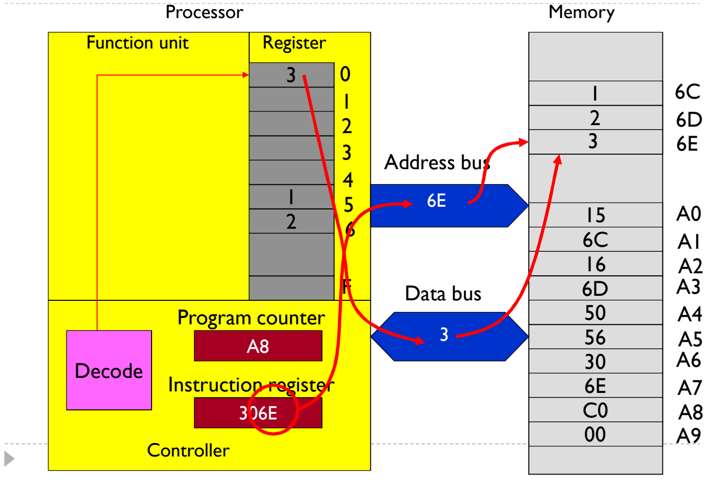
Fetch instruction 4



Decode instruction 4



Execute instruction 4



Exercise

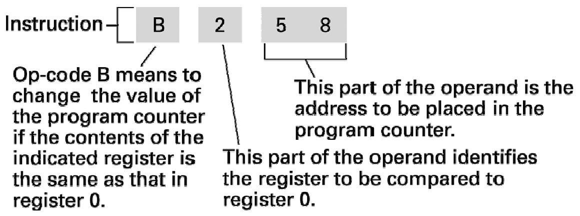
Suppose PC=B0

1. What is in register 3 after the first instruction?
2. What is the memory cell B8 when the program halts?

	0	1	2	3	4	5	6	7	8
B	13	B8	A3	02	33	B8	C0	00	0F

Jump

- JUMP to instruction at address 58_H if the content of register 2 is the same as that of register 0



Op-code B means to change the value of the program counter if the contents of the indicated register is the same as that in register 0.

This part of the operand is the address to be placed in the program counter.

This part of the operand identifies the register to be compared to register 0.

Exercise (17)

Suppose PC=00. When the machine halts, what bit pattern will be in

1. register 1
2. register 0
3. program counter

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	20	03	21	01	40	12	51	12	B1	0C	B0	06	C0	00		
1																

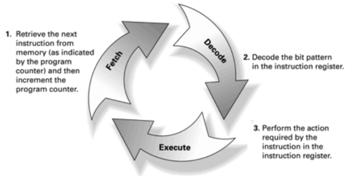
Exercise (28)

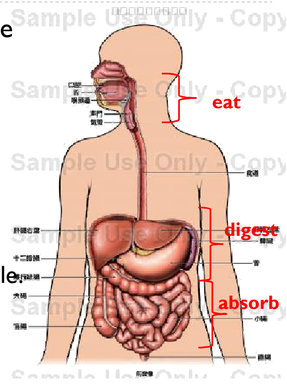
Suppose the program counter = 30_H when the program is started. What task will the program perform?

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	12	34	56	78	9A	BC	DE	F0								
1																
2																
3	20	03	21	01	22	00	23	01	14	00	34	10	52	21	53	31
4	32	39	33	3B	B2	48	B0	38	C0	00						

Pipeline

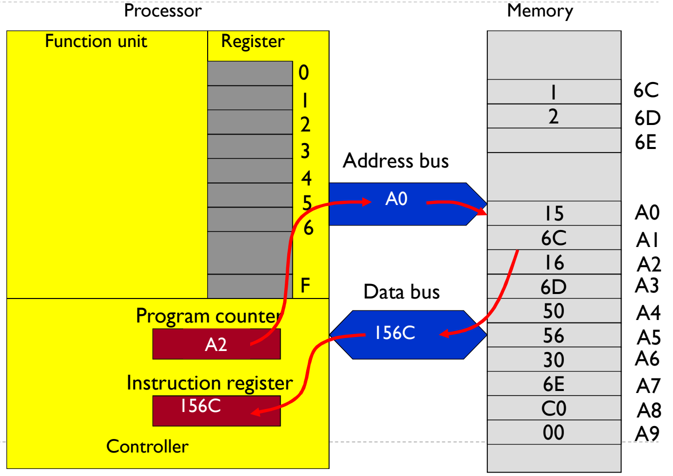
- Three stages: fetch, decode, execute


1. Retrieve the next instruction from memory (as indicated by the program counter) and then increment the program counter.
Fetch
Decode
2. Decode the bit pattern in the instruction register.
Execute
3. Perform the action required by the instruction in the instruction register.

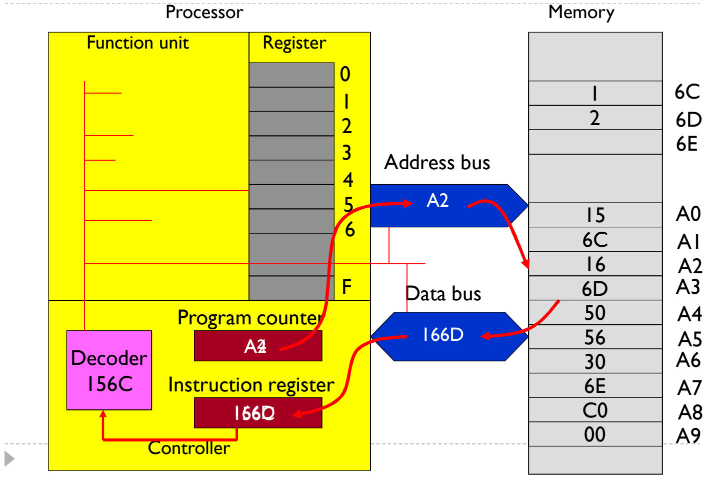


- Suppose each stage takes 1 clock cycle.
- How many clock cycles are needed to execute 1 instruction?
- How many clock cycles are needed to execute 50 instructions?

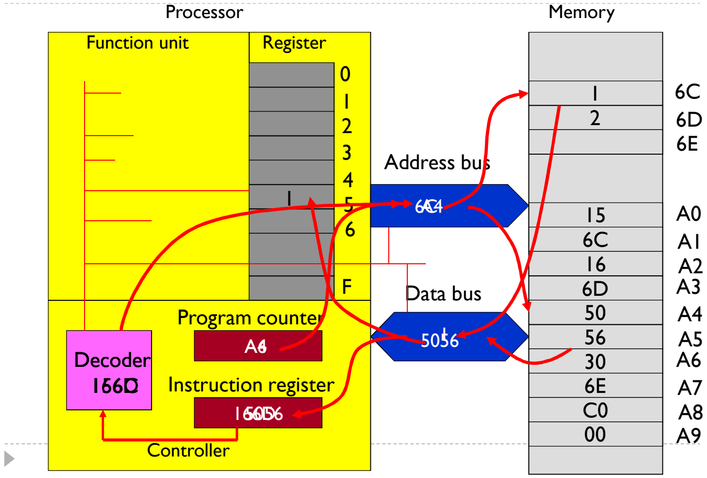
Fetch instruction 1



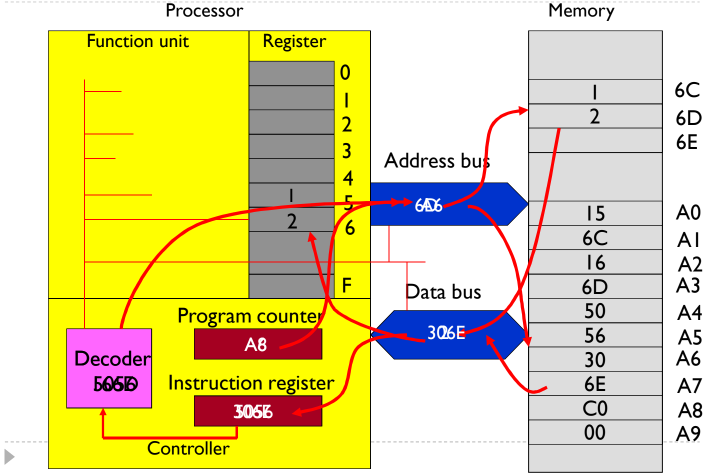
Decode Inst1+Fetch Inst 2



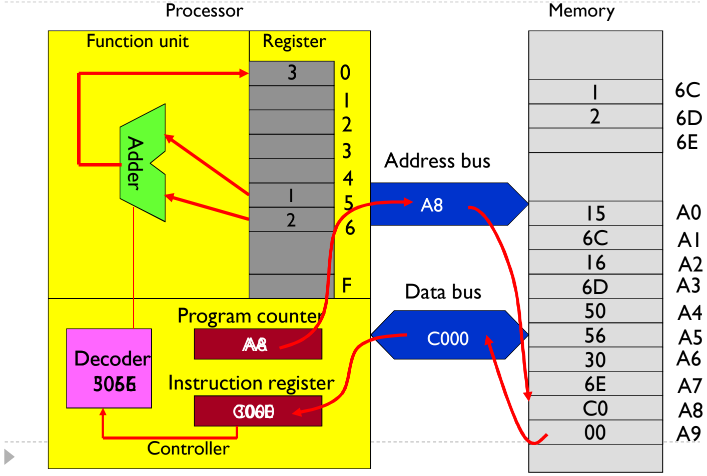
Exec Inst1+Decode Inst2+Fetch Inst3



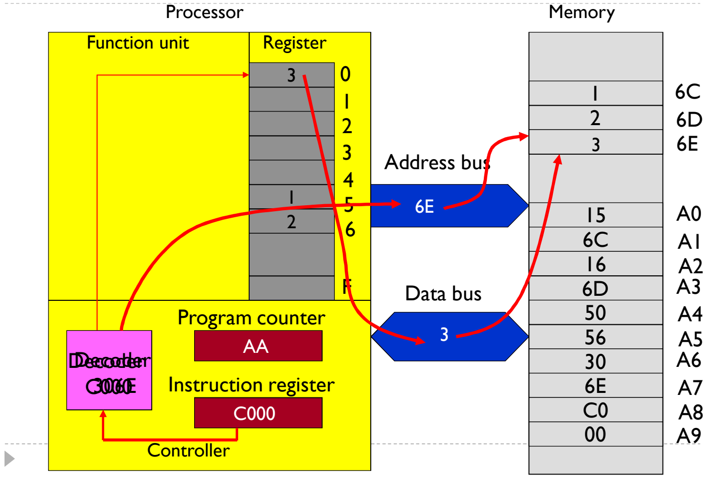
Exec Inst2+Decode Inst3+Fetch Inst4



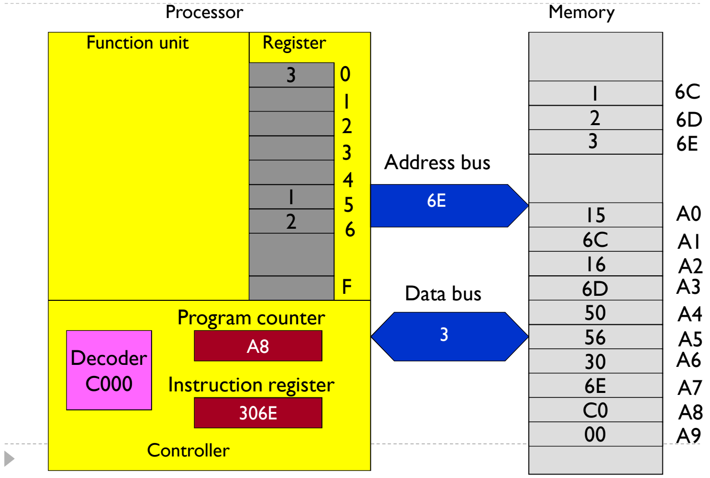
Exec Inst3+Decode Inst4+Fetch Inst5



Execute Instr4+Decode Inst5



Execute Instruction 5



Pipeline

- Since the hardware used in each stage is separated, CPU can overlap the stages

	Clk 1	Clk2	Clk 3	Clk 4	Clk 5	Clk 6	Clk 7	Clk8	Clk 9	
Fetch	Inst 1	Inst 2	Inst 3	Inst 4	Inst 5	Inst 6	Inst 7	Inst 8	Inst 9	…
Decode		Inst 1	Inst 2	Inst 3	Inst 4	Inst 5	Inst 6	Inst 7	Inst 8	…
Execute			Inst 1	Inst 2	Inst 3	Inst 4	Inst 5	Inst 6	Inst 7	…

- The more stages, the better throughput ?
  - Throughput = # executed instructions/time
  - Pentium 4 had a 35-stage pipeline.