Chapter 5

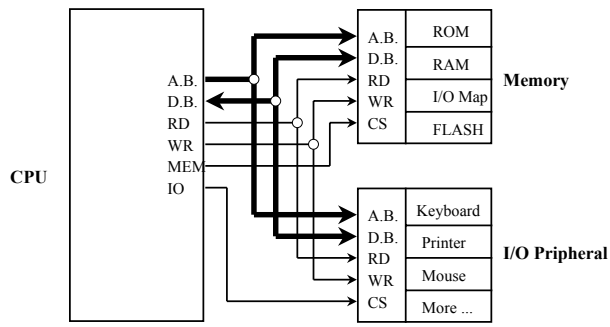
Computer Organization

Process Control

Outline

- **General Architecture of Computer**
  - CPU - MEM - I/O
  - Peripheral
- **Memory Hierarchy**
  - Main Memory
  - Auxiliary Memory
- **Input-Output Interface**

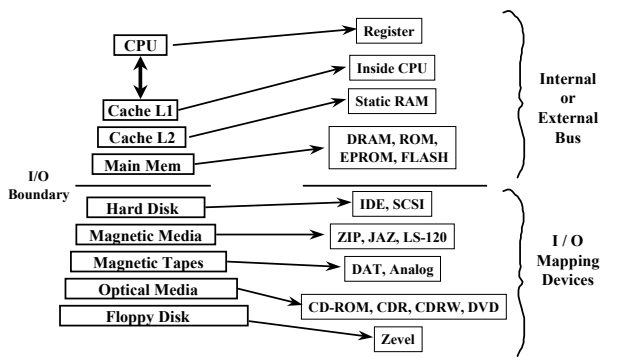
CPU - Memory - I/O



Outline

- **General Architecture of Computer**
  - CPU - MEM - I/O
  - Peripheral
- **Memory Hierarchy**
  - Main Memory
  - Auxiliary Memory
- **Input-Output Interface**

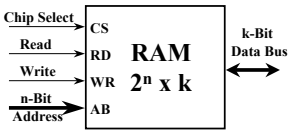
Memory Hierarchy



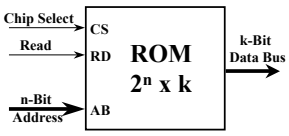
Main Memory

- RAM (Random Access Memory):
  - ***Static*** **=> Flip Flop, Fast, Expensive, No Controller.**
  - ***Dynamic*** **=> Mos Capacitor, Slow, Cheap, Refreshing controller.**
- ROM (Read Only Memory):
  - **Type: ROM, PROM, EPROM, E^2PROM, Flash.**
  - **Used: BIOS, Table, etc.**
- Not all the memory address space must be occupied, parts of them can be empty (without memory or other devices).

RAM & ROM Chips

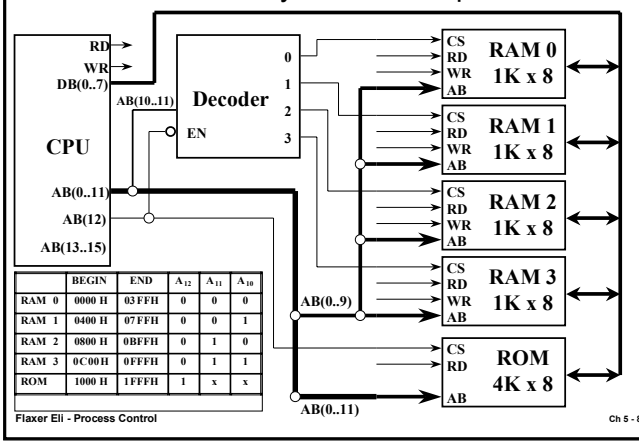


CS	RD	WR	Function	Data Bus
0	x	x	None	Tri - State
1	0	0	None	Tri - State
1	0	1	Write	From Outside
1	1	0	Read	From Memory
1	1	1	Ilegal	



CS	RD	Function	Data Bus
0	x	None	Tri -State
1	0	None	Tri -State
1	1	Read	From Memory

Memory Address Map

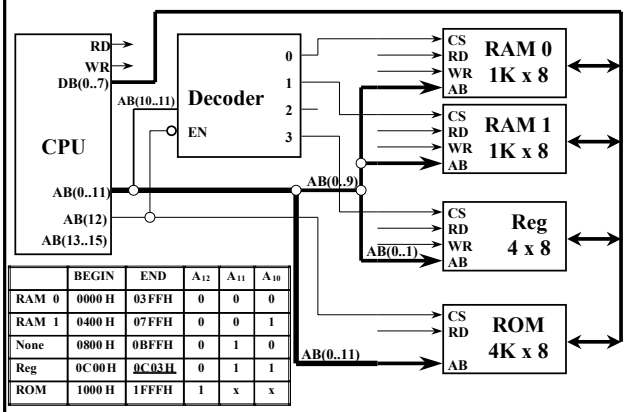


	BEGIN	END	A_{12}	A_{11}	A_{10}
RAM 0	0000 H	03FFH	0	0	0
RAM 1	0400 H	07FFH	0	0	1
RAM 2	0800 H	0BFFH	0	1	0
RAM 3	0C00 H	0FFFH	0	1	1
ROM	1000 H	1FFFH	1	x	x

Address Map Table

	BEGIN	END	A_{12}	A_{11}	A_{10}
RAM 0	0000 H	03FFH	0	0	0
RAM 1	0400 H	07FFH	0	0	1
RAM 2	0800 H	0BFFH	0	1	0
RAM 3	0C00 H	0FFFH	0	1	1
ROM	1000 H	1FFFH	1	x	x

Unoccupied Memory Address



	BEGIN	END	A_{12}	A_{11}	A_{10}
RAM 0	0000 H	03FFH	0	0	0
RAM 1	0400 H	07FFH	0	0	1
None	0800 H	0BFFH	0	1	0
Reg	0C00 H	0C03H	0	1	1
ROM	1000 H	1FFFH	1	x	x

Address Map Table

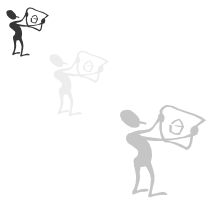
	BEGIN	END	A_{12}	A_{11}	A_{10}
RAM 0	0000 H	03FFH	0	0	0
RAM 1	0400 H	07FFH	0	0	1
None	0800 H	0BFFH	0	1	0
Reg	0C00 H	0C03H	0	1	1
ROM	1000 H	1FFFH	1	x	x

Auxiliary Memory

- Hard Disk:
  - ***IDE*** **=> 30G, Fast, Internal.**
  - ***Scsi*** **=> 100G, Ultra Fast, Internal / External.**
- TAPE :
  - ***DAT*****: Slow, 100G per cassette.**
  - ***Travan*****: Very Slow, 1G per cassette.**
- Optics:
  - ***CD, CDR, CDRW*** **=> 640M.**
  - ***DVD, DVD-RAM*** **=> 5G.**
- Removable:
  - ***Zip*** **=> 100M, 250M.**
  - ***Jaz*** **=> 1G, 2G.**
  - ***Ls-120*** **=> 120M.**
  - ***More*** **=> Floppy, Flash, ...**

Outline

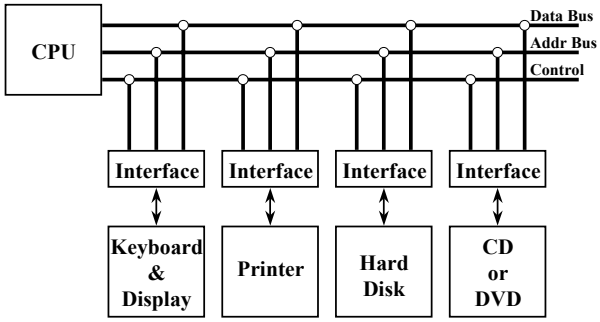
- **General Architecture of Computer**
  - CPU - MEM - I/O
  - Peripheral
- **Memory Hierarchy**
  - Main Memory
  - Auxiliary Memory
- **Input-Output Interface**



Input-Output Interface

- I/O interface provide a method for transferring information between internal storage and external I/O devices.
- To resolve the differences between CPU and peripherals (Mechanism, Transfer Rate, Format, etc) the system include interface.
- In addition, each device may have its own controller that supervise the operation of the particular mechanism in the peripheral.

I/O Bus and Interface



Input-Output Interface

- To communicate with particular device, the processor place a devise address on the address bus (***Port Addr***).
- When the interface detects ins own address, it activates the device that it controls. All others peripherals are disable in Tri-State.
- At the same time the processor provides a function code in the control lines.
- The data send / received in Bi-directional data bus.
- Each CPU has special opcode for I/O operation.

Physical Memory Accessing

Assembler	**READ**	**WRITE**
	Mov DI , AbsAdr **Mov Al , [DI]**	**Mov DI , AbsAdr** **Mov [DI] , Al**
C & C++	**READ**	**WRITE**
char *AbsAdrPtr = AbsAdr; **MyData = (*AbsAdrPtr) ;**		**(*AbsAdrPtr) = MyData;**

Physical Memory Accessing

- If the operation system not allow direct accessing to the hardware, we must use tool kit or lib to map the physical memory to the pointer.
- Each compiler has it's own library

```
static unsigned char *kbf1;
int pt1;
int main ()
        {
        unsigned char temp;
        MapPhysicalMemory (0x417, 1, &kbf1, &pt1);
        temp = *kbf1;
        printf("%x", temp);
        UnMapPhysicalMemory (pt1);
        }
```

Physical Memory Accessing

- ReadFromPhysicalMemory (AbsAdr, Buffer, Number_Of_Byte);
- WriteToPhysicalMemory (AbsAdr, Buffer, Number_Of_Byte);

```
int main ()
        {
        unsigned char temp;
        ReadFromPhysicalMemory (0x417, temp, 1);
        WriteToPhysicalMemory (0x469, 1000, 4);
        printf("%x", temp);
        }
```


Input-Output Instruction

Assembler	IN	OUT
BYTE	In AL , Op1	Out Op1, AL
WORD	In AX , Op1	Out Op1, AX
DWORD	In EAX, Op1	Out Op1, EAX

Op1 = Immd 8 bit Addr or 16 bit address in DX.

C & C++	IN	OUT
BYTE char	inp (int Address)	outp (int Address, char Data)
WORD short	inpw (int Address)	outpw (int Address, short Data)

Input-Output Instruction

PASCAL & DELPHI 1.0

BYTE Data := Port[Address]	Port[Address] := Data
WORD Data := Portw[Address]	Portw[Address] := Data

DELPHI 2.0+ (32 bit)

Not Support Use inline assembler	Not Support Use inline assembler

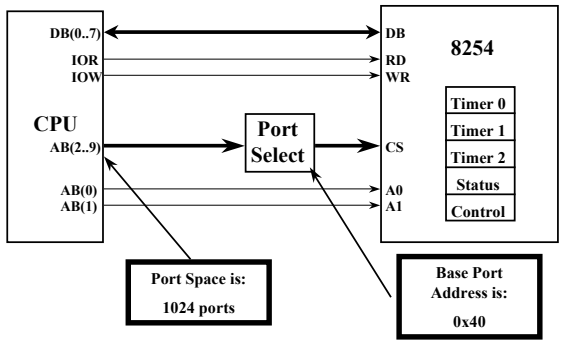
BASIC

Data% = INP(Address%)	OUT(Address%, Data%)

VISUAL BASIC

Not Support Use external DLL	Not Support Use external DLL

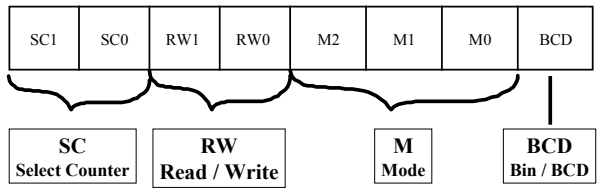
Example: 8254



8254 Addresses

Select	CS	A_1	A_0	RD	WR
Tri -State	0	x	x	x	x
Timer 0	1	0	0	x	x
Timer 1	1	0	1	x	x
Timer 2	1	1	0	x	x
Control	1	1	1	0	1
Status	1	1	1	1	0

8254 Control



8254 Control Field

SC - Select Counter

SC1	SC0	
0	0	Select Counter 0
0	1	Select Counter 1
1	0	Select Counter 2
1	1	Read - Back Command

RW - Read Write

RW1	RW0	
0	0	Counter Latch Command
0	1	Read / Write LSB
1	0	Read / Write MSB
1	1	Read / Write LSB first & MSB after

8254 Control Field

M - Mode

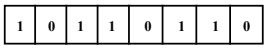
M2	M1	M0	
0	0	0	Mode 0
0	0	1	Mode 1
0	1	0	Mode 2
0	1	1	Mode 3
1	0	0	Mode 4
1	0	1	Mode 5

BCD - Binary Code Decimal

BCD	
0	Binary Counter 16 Bit
1	BCD Counter 4 Digit

I/O Programming Example (C)

- Set the Timer2 of 8254 to mode 3, Binary counting, and 16 bit interfacing. Load the counter with 0x1234.

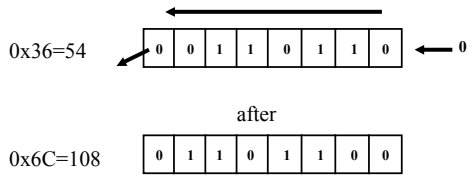


= 0xB6

```
int main ()
        {
        unsigned char temp;
        temp = 0xB6;
        outp(0x43, temp)        // write control
        outp(0x42, 0x34)        // write least
        outp(0x42, 0x12)        // write most
        }
```

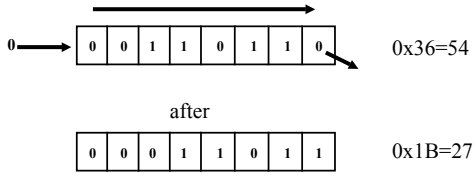

Shift Operations

- C language has two shift operations for variable and constant:
  >> (shift right) << (shift left).
- Shift left operation always insert '0' to the LSB



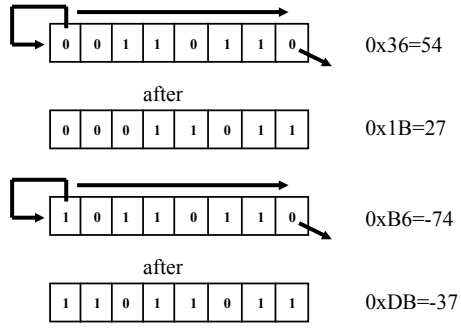
Shift Operations

- Shift right operation insert, to the MSB, value that depended in the type of the variable or the constant.
- If the type is unsigned, '0' is insert to the MSB



Shift Operations

- If the type is sign, the MSB is duplicated (sign extended).

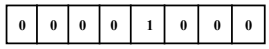


Shift Operations Example (C)

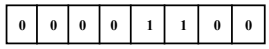
```
int main ()
        {
        unsigned char x, y;
        char a, b;
        x = 1;          // x = 00000001
        y = x << 3;     // y = 8
        x = 255;        // x = 11111111
        y = x >> 3;     // y = 31

        a = 1;          // a = 00000001
        b = a << 3;     // b = 8
        a = -1;         // a = 11111111
        b = a >> 3;     // b = -1
        }
```


Constant Shift Operations

- The shift operations work with a constant too.
- For example:
  - 1 << 3 = 8



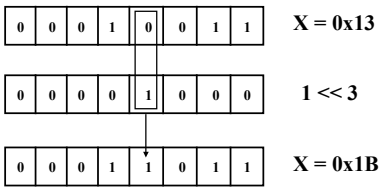
  - 48 >> 2 = 12



- Constant integer, is by default 32 bit signed.
- Unsigned constant is defined by U.

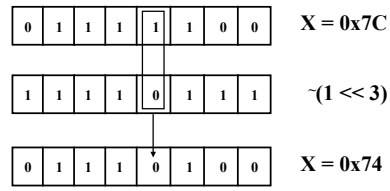
Bit Manipulations

- How to set especial bit of the variable ?
- Set the k bit:
  - **x |= (1 << k)**
  - For example set bit 3 **X |= (1 << 3)**:



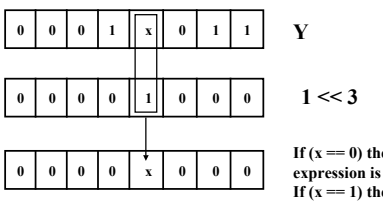
Bit Manipulations

- How to clear especial bit of the variable ?
- Clear the k bit:
  - **x &= ~(1 << k)**
  - For example clear bit 3 **X &= ~(1 << 3)**:



Bit Sensing

- How to check especial bit of the variable ?
- Check the k bit of variable Y:
  - **Y & (1 << k)**
  - For example check bit 3 **Y & (1 << 3)**:



If (x == 0) the all expression is FALSE
If (x == 1) the all expression is TRUE

Byte to Bit Conversion Example (C)

```
Void Byte2Bit (char b, char y[])
        {
        int k;
        for (k=0; k<8; k++)
           y[k] = b & (1<<k);
        }
```

```
Void Byte2Bit (char b, char y[])
        {
        int k;
        for (k=0; k<8; k++, b >>= 1)
           y[k] = b & 1;
        }
```

Bit to Byte Conversion Example (C)

```
char Bit2Byte (char y[])
        {
        int k, b;
        for (k=0, b=0; k<8; k++)
            b |= (y[k] << k);
        return(b);
        }
```

```
char Bit2Byte (char y[])
        {
        int k, b;
        for (k=7, b=0; k>=0; b |= y[k--])
            b <<= 1;
         return(b);
        }
```