Organization of the IBM Personal Computers

IBM Personal Computers

The IBM Personal Computers are based on the Intel 8086 family of microprocessors

The Intel 8086 Family of Microprocessors

The Intel 8086 family of microprocessors include

- **8086,**
- **8088,**
- **80186,**
- **80188,**
- **80286,**
- **80386,**
- **80386SX,**
- **80486, and**
- **80486SX.**

Registers

- *Information inside the microprocessor* **is stored in registers**
- **Classified** *according to the functions* **they perform**
  - *Data registers* **hold data for an operation.**
  - *Address registers* **hold the address of an instruction or data.**
  - *Status register* **keeps the current status of the processor.**

Registers in 8086 Microprocessors

- **has *four general data registers***
- **address registers are divided into *segment, pointer and index registers* in 8086**
- **status register is called *FLAGS register***
- **in total, there are *fourteen 16-bit registers***

Data Registers:
AX, BX, CX, DX

- **These four registers are available to the programmer *for general data manipulation.***
- **Even though the processor can operate on data stored in memory, the same instruction is *faster* (requires fewer clock cycles*) if the data are stored in registers*.**
- **The high and low bytes of the data registers *can be accessed separately.* This arrangement gives us more registers to use *when dealing with byte-size data.***

Data Registers:
AX, BX, CX, DX

AX	AH	AL	Accumulator
BX	BH	BL	Base
CX	CH	CL	Count
DX	DH	DL	Data

AX (Accumulator Register)

- ***High byte*** **of AX is called** ***AH***
- ***Low byte*** **of AX is called** ***AL***
- **Preferred register in** ***arithmetic, logic and data transfer*** **instructions because it** ***generates shortest machine codes***
- **In multiplication and division,** ***one of the numbers involved must be in AX or AL***
- **Input and output operations also require AX and AL**

BX (Base Register)

- ***high byte*** **of BX is called** ***BH***
- ***low byte*** **of BX is called** ***BL***
- **also serves as an address register**

CX (Count Register)

- ***high byte*** **of CX is called** ***CH***
- ***low byte*** **of CX is called** ***CL***
- **Program** ***loop constructions*** **are facilitated by the use of CX, which serves as a** ***loop counter.***
- **CL ls used as a count in instructions that** ***shift and rotate bits*****.**

DX (Data Register)

- ***high byte*** **of DX is called** ***DH***
- ***low byte*** **of DX is called** ***DL***
- **Used in** ***multiplication and division***
- **Also used in** ***I/O operation***

Segment Registers:
CS, DS, SS, ES

Address registers store addresses of instructions and data in memory.

Memory Organization

- **Memory is a *collection of bytes*.**
- **Each memory byte *has an address*, starting with 0.**
- **The 8086 processor assigns a *20-bit physical address* to its memory locations.**
- **The addresses are *too big to fit* in a 16-bit register or memory word. The 8086 gets around this problem by *partitioning its memory into segments.***

Memory Segments

- **Physical available memory is *divided into a number of logical segments.***
- **A memory segment is a block of *2^{16} (or 64 K)* consecutive memory bytes.**
- **Each segment is *identified by a segment number*, starting with 0.**
- **A segment number is *16 bits*, so the highest segment number is $FFFF_h$.**
- **Within a segment, a memory location is specified *by an offset***
- **Offset is the number of bytes *from the beginning of the segment.***
- **With a 64 KB segment, the offset can be given as a *16-bit number*. The first byte in a segment has offset 0. The last offset in a segment is $FFFF_h$.**

Segment:Offset Address

- **A memory location may be specified by providing a segment number and an offset, written in the form *segment:offset*; this is known as a *logical address*.**
- **For example, A4FB:4872_h means offset 4872_h within segment $A4FB_h$.**
- **To obtain a 20-bit physical address, the 8086 microprocessor *shifts the segment address 4 bits to the left and then adds the offset.***
- **Thus the physical address for A4FB:4872 is $A9822_h$**

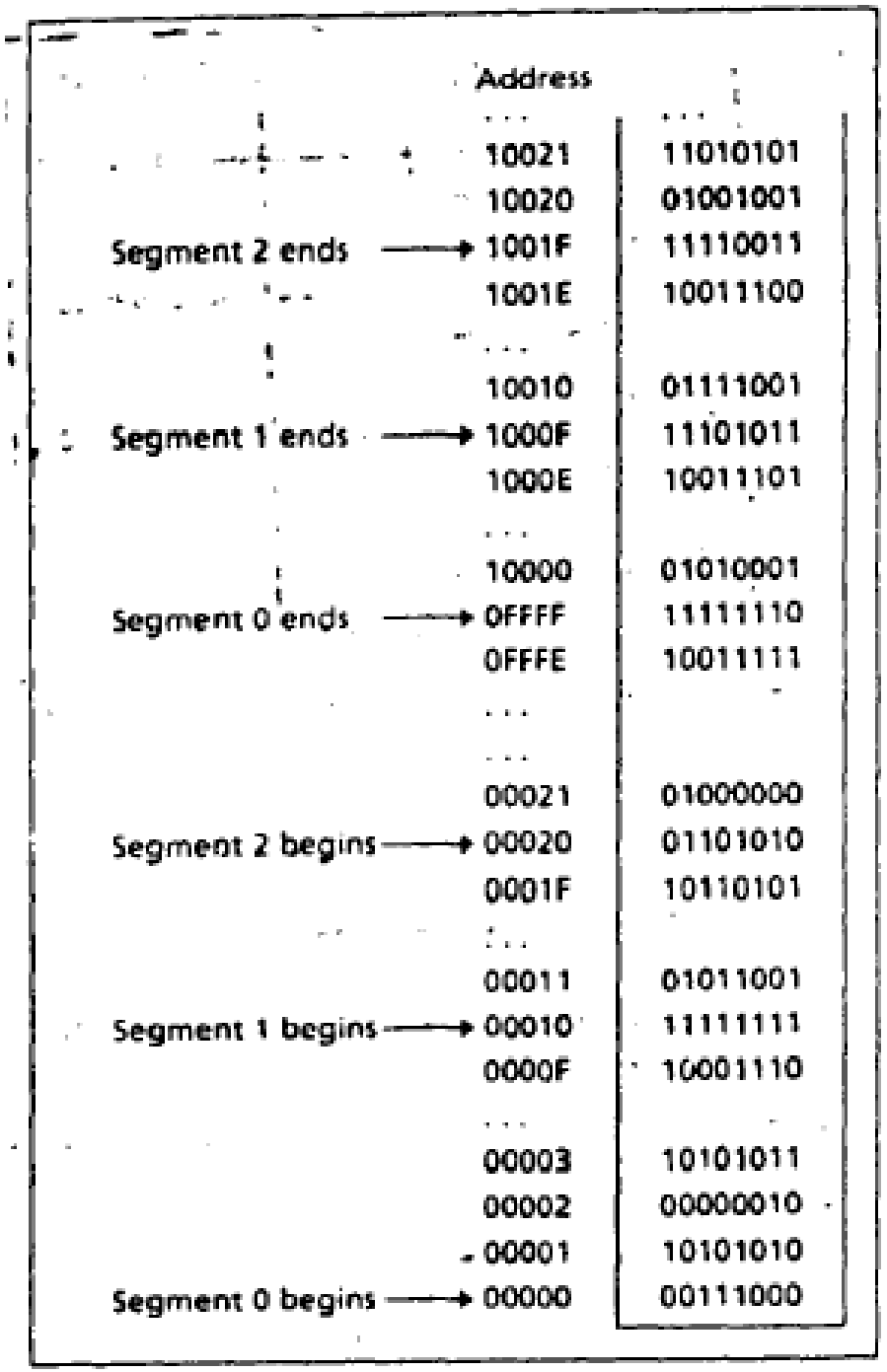
Segment:Offset Address

- **A memory location may be specified by providing a segment number and an offset, written in the form *segment:offset*; this is known as a *logical address*.**
- **For example, A4FB:4872_h means offset 4872_h within segment $A4FB_h$.**
- **To obtain a 20-bit physical address, the 8086 microprocessor *shifts the segment address 4 bits to the left and then adds the offset.***
- **Thus the physical address for A4FB:4872 is $A9822_h$**

Location of Segments

- **Segment 0 starts at address 0000:0000 = 0000_h and ends at 0000:$FFFF_h$ = $0FFFF_h$.**
- **Segment 1 starts at address 0001:0000 = 00010_h and ends at 0001:FFFF = $1000F_h$.**
- **Thus we see that there are a lot of *overlapping between segments.***

Location of Segments



Location of Segments

For the memory location whose physical address is specified by $1256A_h$, give the address in segment:offset form for segments 1256_h and 1240_h.

Location of Segments

For the memory location whose physical address is specified by $1256A_h$, give the address in segment:offset form for segments 1256_h and 1240_h.

Let X be the offset in segment 1256_h . Thus we have,

$$1256A_h = 12560_h + X$$

$$\text{or, } X = 1256A_h - 12560_h$$

$$\text{or, } X = A_h$$

Thus, required address is 1256:000A

Location of Segments

A memory location has physical address 80FD2h. In what segment does it have offset BFD2h?

Location of Segments

A memory location has physical address 80FD2h. In what segment does it have offset BFD2h?

We know that,

$$\text{physical address} = \text{segment} \times 10_h + \text{offset}$$

Thus,

$$\text{segment} \times 10_h = \text{physical address} - \text{offset}$$

$$\text{or, segment} \times 10_h = 80FD2_h - BFD2_h$$

$$\text{or, segment} \times 10_h = 75000_h$$

Therefore, segment = 7500_h

Program Segments

- **The program's code, data, and stack are loaded into different memory segments namely the *code segment, data segment, and stack segment* respectively.**
- **To keep track of the various program segments, the 8086 is equipped with four segment registers *to hold segment numbers.***
- **The CS, DS and SS registers contain the *code, data, and stack segment numbers* respectively.**
- **If a program needs to access *a second data segment*, it can use *the ES (extra segment) register.***
- **At any given time, only those memory locations addressed by the four segment registers are accessible; that is, *only four memory segments are active.***

Code Segment(CS)

Points at the segment containing the current program.

Data Segment(DS)

- **Generally points at segment where variables are defined.**

Stack Segment(SS)

Points at the segment containing the stack.

Extra Segment(ES)

it's up to a coder to define its usage.

Pointer and Index Registers: SP, BP SI, DI

The registers SP, BP, SI, and DI normally point to (*contain the offset addresses* of) memory locations. Unlike segment registers, the pointer and index registers *can be used in arithmetic and other operations.*

SP (Stack Pointer)

Used with SS for accessing the *stack segment*

BP (Base Pointer)

Used primarily to access *data on the stack.* Can also be used to access *data in other segments*

SI (Source Index)

Used to point to memory locations in the *data segment* addressed by DS. *Incrementation of SI* gives *consecutive memory locations*

DI (Destination Index)

Same function as SI. *String operations* use DI to access memory locations *addressed by ES*

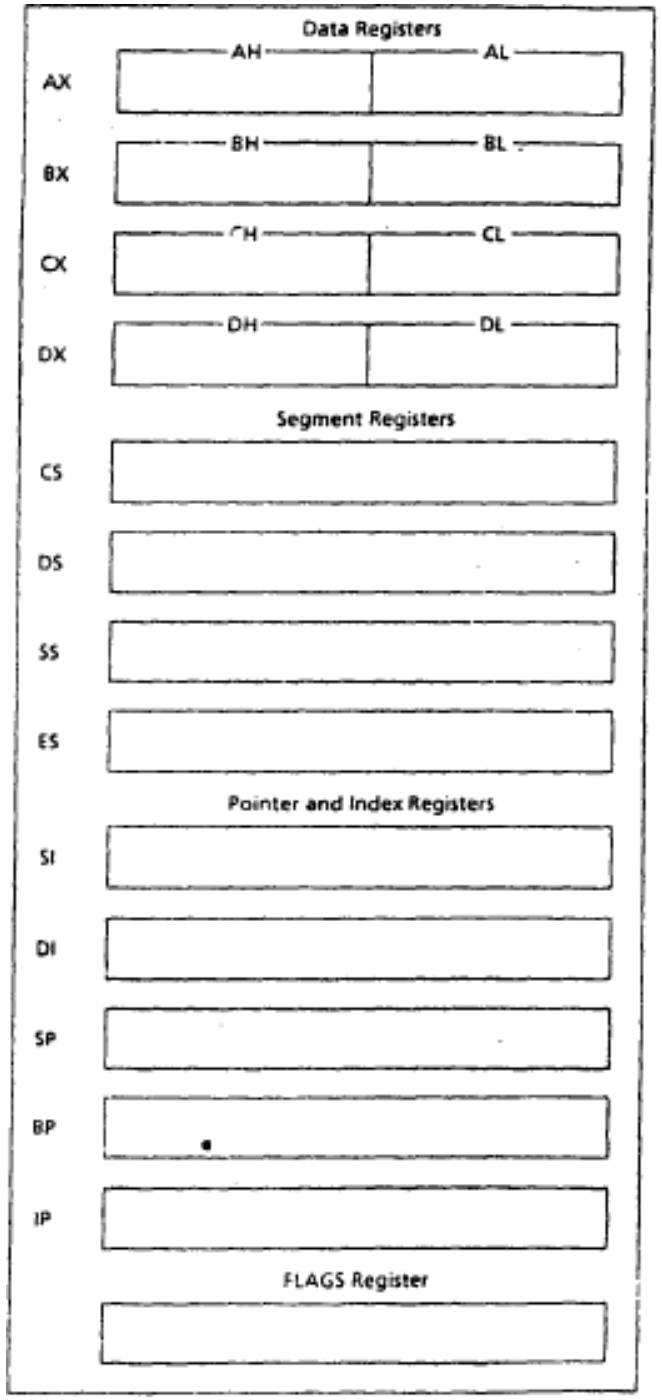
IP (Instruction Pointer)

- **To *access instructions*, the 8086 uses *CS and IP* registers.**
- ***CS contains the segment number* of the next instruction *and IP contains the offset***
- **IP is updated each time an instruction is executed**
- **Unlike other registers, IP *cannot be directly manipulated* by an instruction.**
- **That is, an instruction *cannot contain IP as its operand.***

FLAGS Register

- **Indicates *the status* of microprocessor by *setting individual bits* called flags.**
- **There are two kinds of flags: *status flags and control flags*. Status flags *reflect the result of an instruction* executed by the processor**
- **For example if an arithmetic operation produce a zero value as a result then the zero flag(ZF) is set to 1**
- **Control flags *enable or disable certain operations* of the processor**
- **For example if interrupt flag(IF) is set to zero, inputs from the keyboard are ignored by the processor**

Register


Data Registers
AX
AH
AL
BX
BH
BL
CX
CH
CL
DX
DH
DL
Segment Registers
CS
DS
SS
ES
Pointer and Index Registers
SI
DI
SP
BP
IP
FLAGS Register