NEUB CSE 321 Lecture 5: Memory Organization and Addressing

Memory segmentation

Since 8086 is a 16 bit processor, meaning its registers are 16 bit wide, and the processor supports 20 bit addressing, the address information cannot be stored in a single registers and so memory segmentation is necessary to address 1MByte of memory. In this process the main memory of computer is divided into different segments and each segment has its own base address.

Segmentation is used to increase the execution speed of computer system so that processor can able to fetch and execute the data from memory easily and fast.

In memory, data is stored as bytes. Each byte has a specific address. Intel 8086 has 20 lines address bus. With 20 address lines, the memory that can be addressed is 2^{20} bytes. 2^{20} = 1,048,576 bytes (1 MB). 8086 can access memory with address ranging from 00000 H to FFFFF H.

In 8086, memory has four different types of segments. These are:

- Code Segment
- Data Segment
- Stack Segment
- Extra Segment

The segments registers and offset registers necessary to locate memory in a specific segment are discussed below:

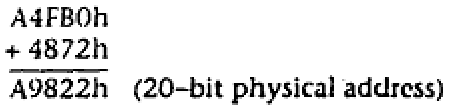
- Code Segment (CS) register is a 16-bit register containing address of 64 KB segment with processor instructions
  - The processor uses CS segment for all accesses to instructions referenced by instruction pointer (IP) register. The IP is updated every time an instruction is executed so that the processor can execute the next instruction.
- Data Segment (DS) register is a 16-bit register containing address of 64KB segment with program data
  - By default, the processor assumes that all data referenced by general registers (AX, BX, CX, DX) and index register (SI, DI) is located in the data segment
- Stack Segment (SS) register is a 16-bit register containing address of 64KB segment with program stack
  - By default, the processor assumes that all data referenced by the stack pointer (SP) and base pointer (BP) registers is located in the stack segment. But Base pointer can also be used to generate address of other segments
- Extra Segment (ES) register is a 16-bit register containing address of 64KB segment, usually with program data
  - There is a class of instructions, called *string operations*, that use DI to access memory locations addressed by ES

The table summarizes the segment registers and offset registers necessary to generate the address in a specific segment

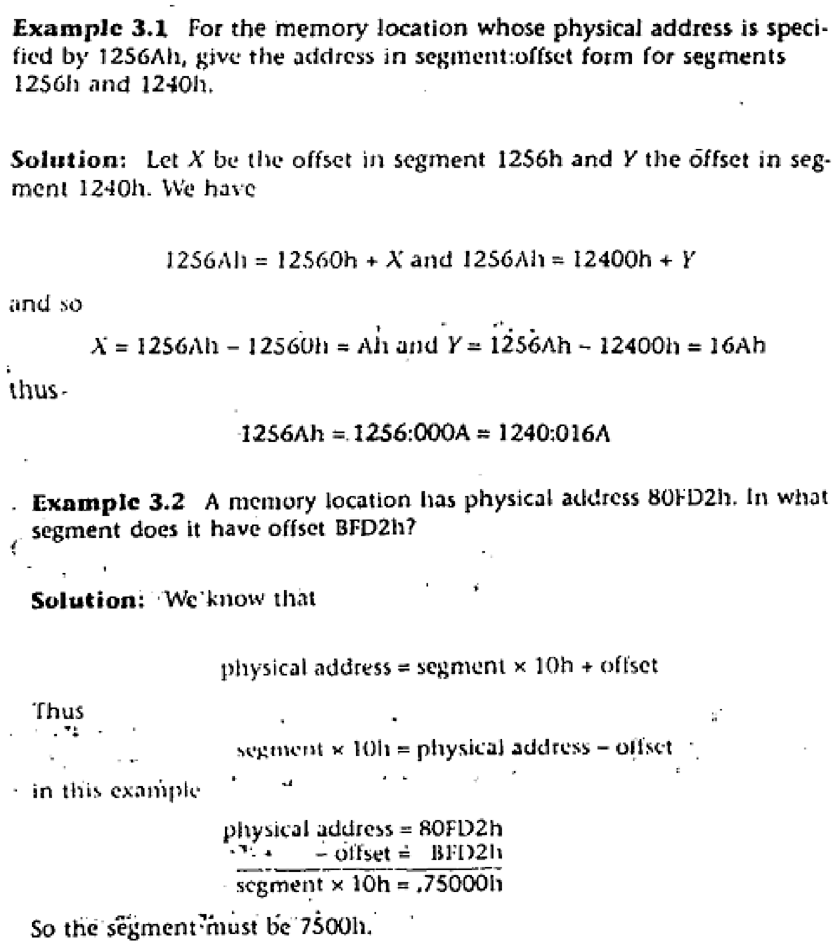
Memory segment	Segment register	Offset register
Code segment	Code segment Register (CSR)	Instruction Pointer (IP)
Data segment	Data segment Register (DSR)	Source index (SI)/ Destination index (DI)
Stack segment	Stack segment Register (SSR)	Stack Pointer (SP)/ Base Pointer (BP)
Extra segment	Extra segment Register (ESR)	Destination Index(DI)

Physical address generation in memory segmentation

- Logical Address is specified as **segment: offset**
- Physical address is obtained by shifting the segment address 4 bits to the left and adding the offset address.
- **A4FB:4872h** means offset **4872h**within segment **A4FBh**
- Thus the physical address of the logical address **A4FB:4872** is:



```
  A4FB0h
+ 4872h
  A9822h   (20-bit physical address)
```



Example 3.1 For the memory location whose physical address is specified by 1256Ah, give the address in segment:offset form for segments 1256h and 1240h.

Solution: Let X be the offset in segment 1256h and Y the offset in segment 1240h. We have

$$1256\text{Ah} = 12560\text{h} + X \text{ and } 1256\text{Ah} = 12400\text{h} + Y$$

and so

$$X = 1256\text{Ah} - 12560\text{h} = \text{Ah and } Y = 1256\text{Ah} - 12400\text{h} = 16\text{Ah}$$

thus

$$1256\text{Ah} = 1256{:}000\text{A} = 1240{:}016\text{A}$$

Example 3.2 A memory location has physical address 80FD2h. In what segment does it have offset BFD2h?

Solution: We know that

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

Thus

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

in this example

```
physical address = 80FD2h
        - offset =  BFD2h
segment × 10h    = 75000h
```

So the segment must be 7500h.

The BIU has a dedicated adder for determining physical memory addresses

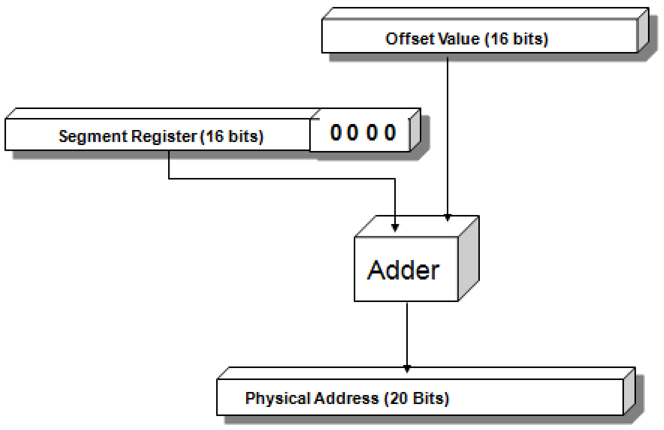
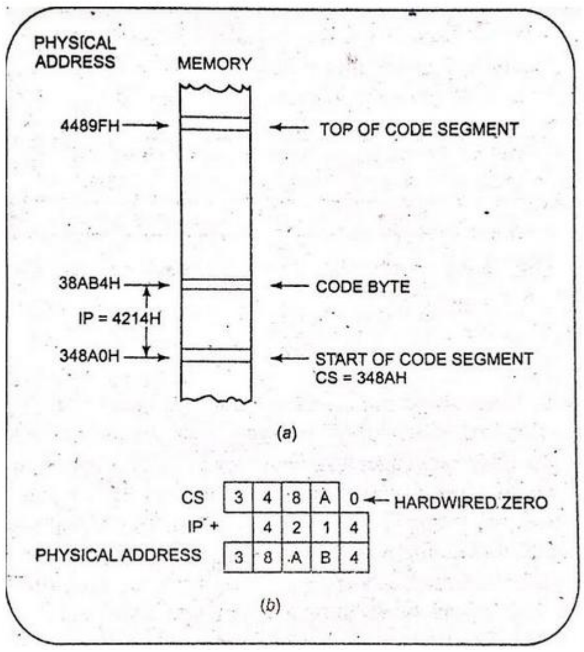
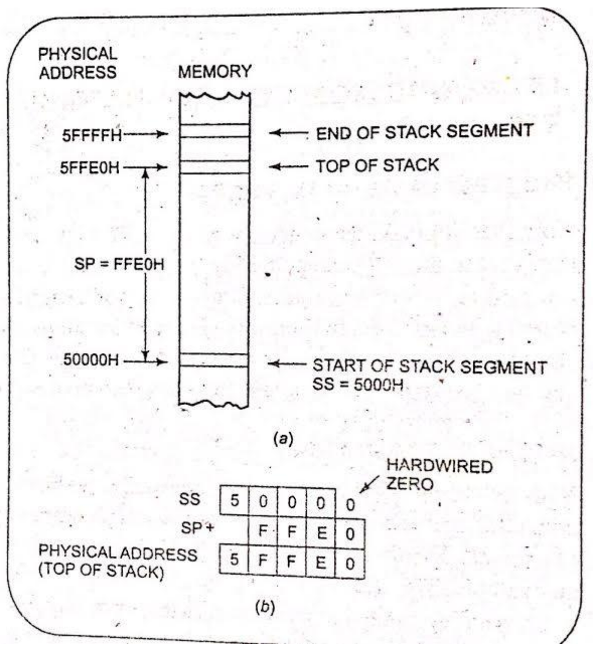


Figure 1 Generation of physical address in BIU

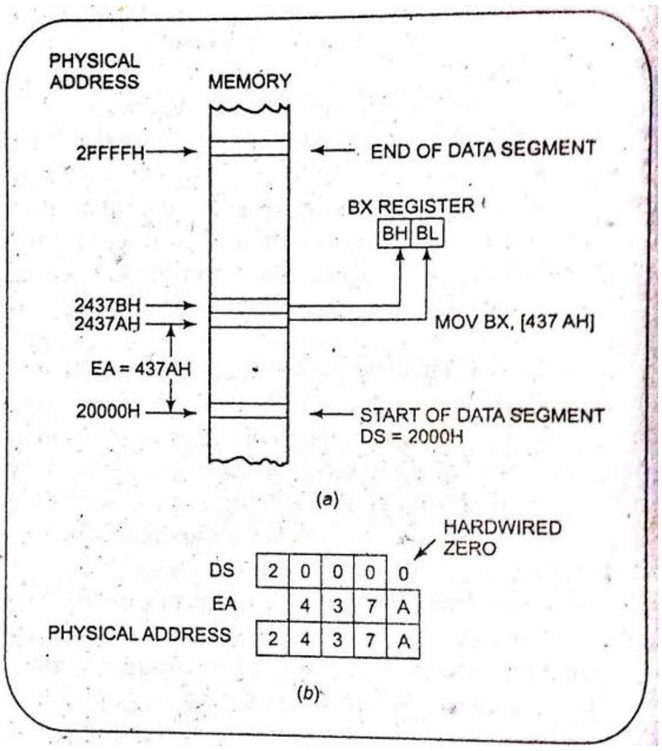
The figures below show

1. Addition of IP to CS to produce the physical address of the code byte
1. Addition of SS to SP to produce the physical address of the top of the stack
2. Addition of data segment register and effective address to produce the physical address of the data byte





1 2



3

PREPARED BY
SHAHADAT HUSSAIN PARVEZ

Types of segmentation

1. Overlapping segmentation
2. Non overlapping segmentation

Overlapping segmentation

A segment starts at a particular address and its maximum size can go up to 64 Kbytes. But if another segment starts along this 64 Kbytes location of the first segment, the two segments are said to be overlapping segment.

The area of memory from the start of the second segment to the possible end of the first segment is called as overlapped segment.

It is instructive to see the layout of the segments in memory. Segment 0 starts at address 0000:0000 = 00000h and ends at 0000:FFFFh = 0FFFFh. Segment 1 starts at address 0001:0000= 00010h and ends at 0001:FFFF = I000Fh. As we can see, there is a lot of overlapping between segments. Figure below shows the locations of the first three memory segments. The segments start every 10h= 16 bytes and the starting address of a segment always ends with a hex digit 0. We call 16 bytes a paragraph. We call an address that is divisible by 16 (ends with a hex digit 0) a paragraph boundary .

Because segments may overlap, the segment:offset form of an address is not unique, JS the following example shows.

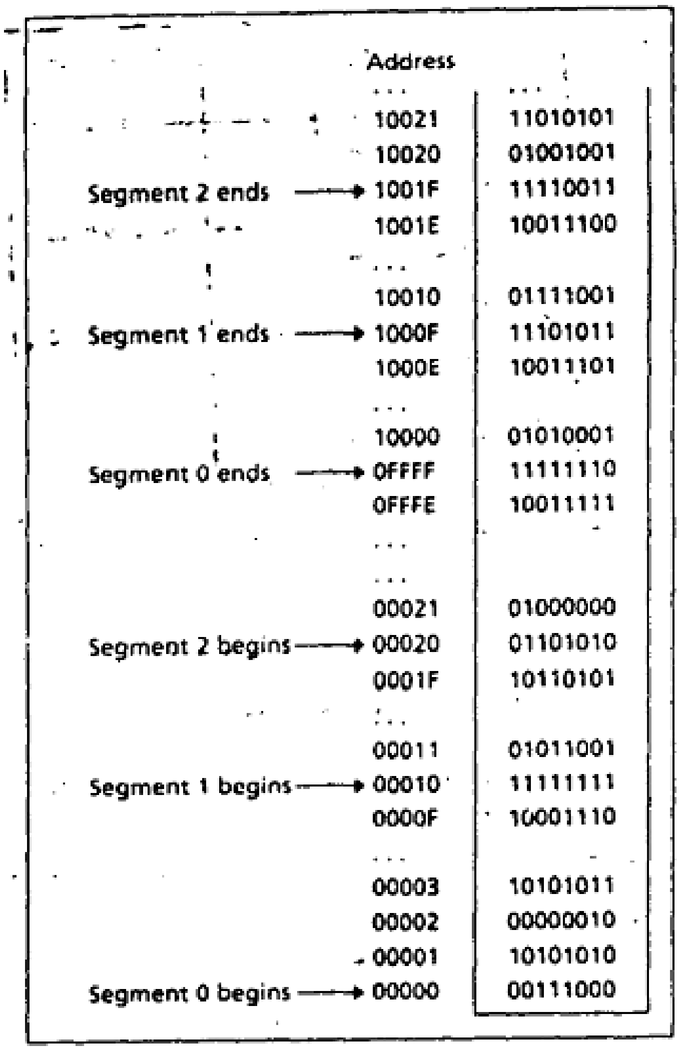


Figure 2 Example of an overlapped segmentation

Non overlapping segmentation

A segment starts at a particular address and its maximum size can go up to 64 Kbytes. But if another segment starts after this 64 Kbytes location of the first segment, the two segments are said to be Non- overlapping segment.

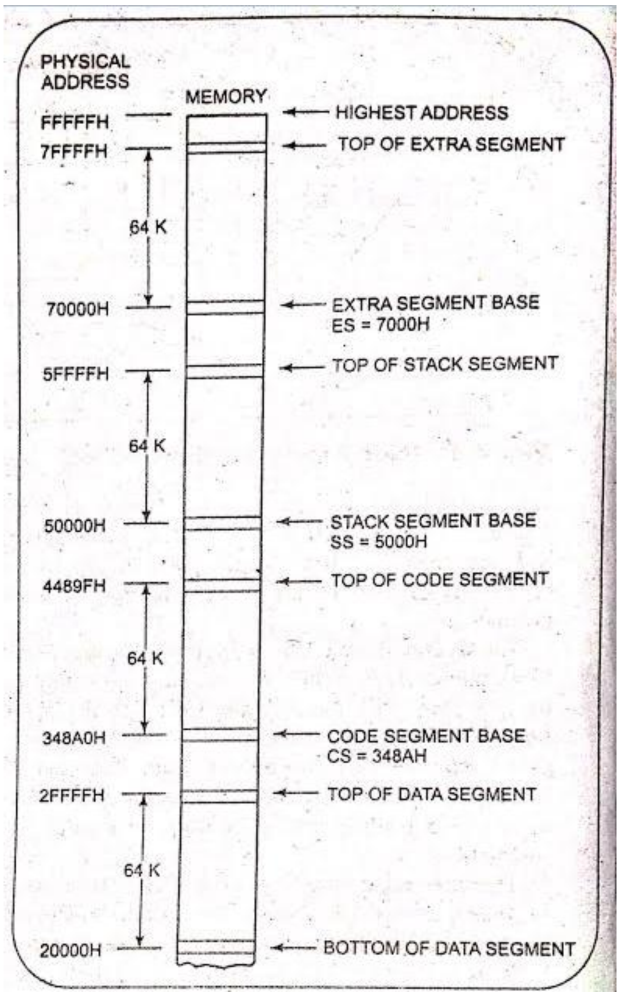
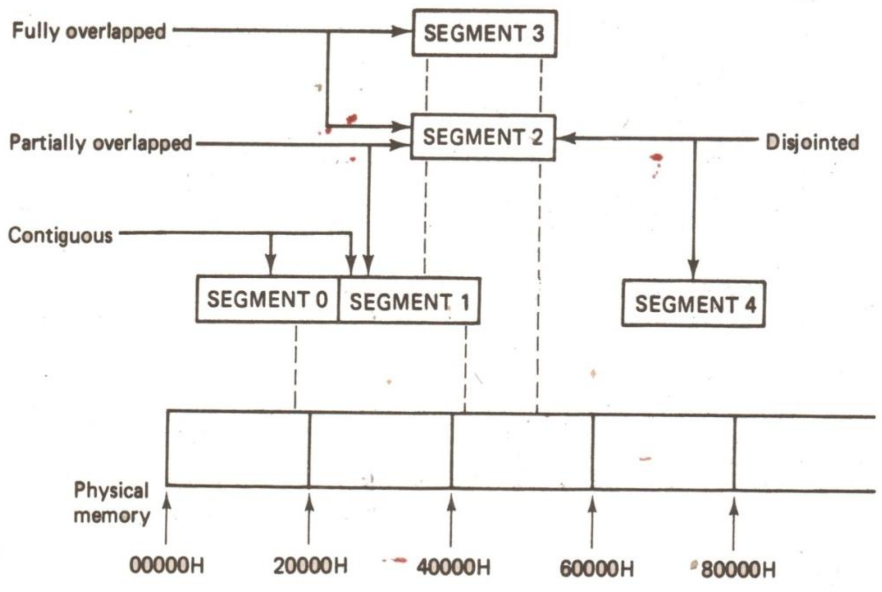


Figure 3 Example of an non overlapping segment

The figure below describes the different type of segmentation of memory



Advantages of segmentation

- Allows the memory capacity to be 1 Mbyte although the actual addresses to be handled are of 16-bit size
- Allows the placing of code data and stack portions of the same program in different parts (segments) of memory, for data and code protection.
- Permits a program and/ or its data to be put into different areas of memory each time program is executed, ie, provision for relocation may be done.

The figure below shows the how physical address and segment:offset notation is linked

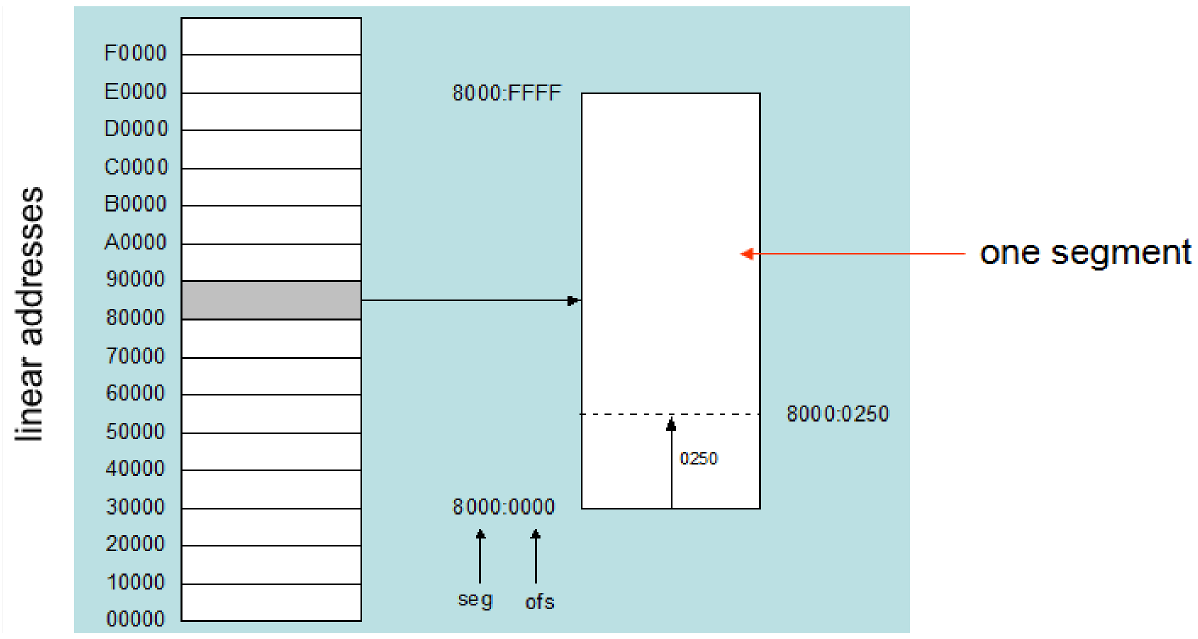


Figure 4 Linear address represented in segment:offset notation

1. What are the advantages of memory segmentation?
2. The contents of the following registers are:
   CS = 1111 H
   DS = 3333 H
   SS = 2526 H
   IP = 1232 H
   SP = 1100 H
   DI = 0020 H
   Calculate the corresponding physical addresses for the address bytes in CS, DS and SS.
3. Determine the physical address of a memory location given by 0A51:CD90h.
4. A memory location has a physical address 4A37Bh. Compute
   a. The offset address if the segment number is 40FFh.
   b. The segment number if the offset address is 123Bh.
5. What is a paragraph boundary?

Addressing modes

There are several addressing modes for addressing any memory location. They are

1. **Register Addressing**
2. **Immediate Addressing**
3. **Direct Addressing**
4. **Register Indirect Addressing**
5. **Based Relative Addressing**
6. **Indexed Relative Addressing**
7. **Based Indexed Relative Addressing**
8. String addressing
9. I/O addressing
10. Relative addressing
11. Implied addressing

Register addressing mode

With the Register Addressing mode the operand to be accessed is specified as residing in an internal register of the 8086.

Example: MOV AX, BX

This code moves the contents of BX (the source operand) to AX (the destination operand).Both the source and the destination operands have been specified as the contents of internal registers of the 8086.

Immediate Addressing

The b-bit or 16-bit data is provided as a part of the instruction in the immediate addressing mode. The data occupies the immediate next byte to the instruction code in memory.

Example: MOV AL , 055H

In this instruction the operand 055H is an example of a byte wide immediate source operand. The destination operand, which consists of the contents of AL, uses register addressing. Thus this instruction employs both immediate and registers addressing modes.

Direct Addressing

The 16-bit offset address of a memory location is directly provided as a part of the instruction in direct memory addressing mode. Direct address references are usually relative to DS.

Example: MOV CX, [1234H]

This addressing moves the contents of the memory location, which is offset by 1234H from the current value in DS (DS:1234H) into internal register CX. If DS contains 2000H, and the memory location 2000H:1234H, and 2000H:1235H contains CDH and ABH respectively the execution will load ABCDH into CX. This is because the destination set is 16 bit, so the source has to be 16 bit and so the 2 bytes are combined to form the 16 bit data. The location larger is the higher byte and smaller location is the lower byte of the 16 bit data.

- To change reference register to any other use a segment override:
  - mov ch, [es:OverByte]
  - mov dh, [cs:CodeByte]
  - mov dh, [ss:StackByte]
  - mov dh, [ds:DataByte]
- An override occupies a byte of machine code which is inserted just before the affected instruction

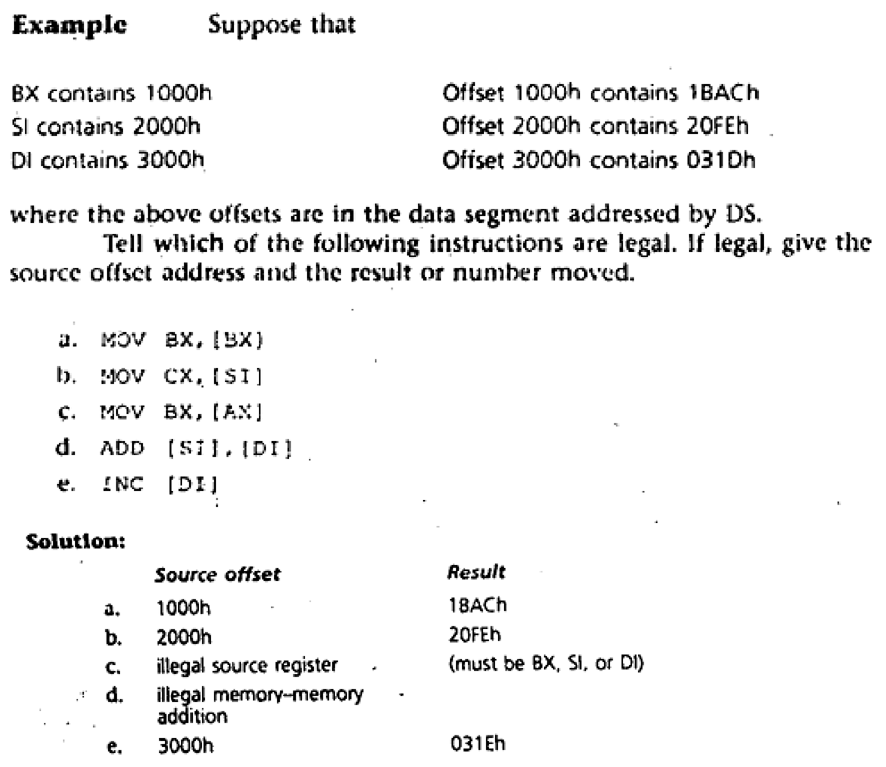
Register Indirect Addressing

Register indirect addressing is similar to direct addressing, that an effective address is combined with the contents of DS to obtain a physical address. However it differs in a way that the offset is specified. Here EA resides in either a pointer register or an index register within the 8086.The pointer register can be either a base register BX or a base pointer register BP and the index register can be source index register SI or the destination index register DI.

Example MOV AX, [SI]

This instruction moves the contents of the memory location offset by the value of EA in SI from the current value in DS to the AX register.

It is to be noted that memory to memory addition is not possible.



Example Suppose that

BX contains 1000h	Offset 1000h contains 1BACh
SI contains 2000h	Offset 2000h contains 20FEh
DI contains 3000h	Offset 3000h contains 031Dh

where the above offsets are in the data segment addressed by DS.

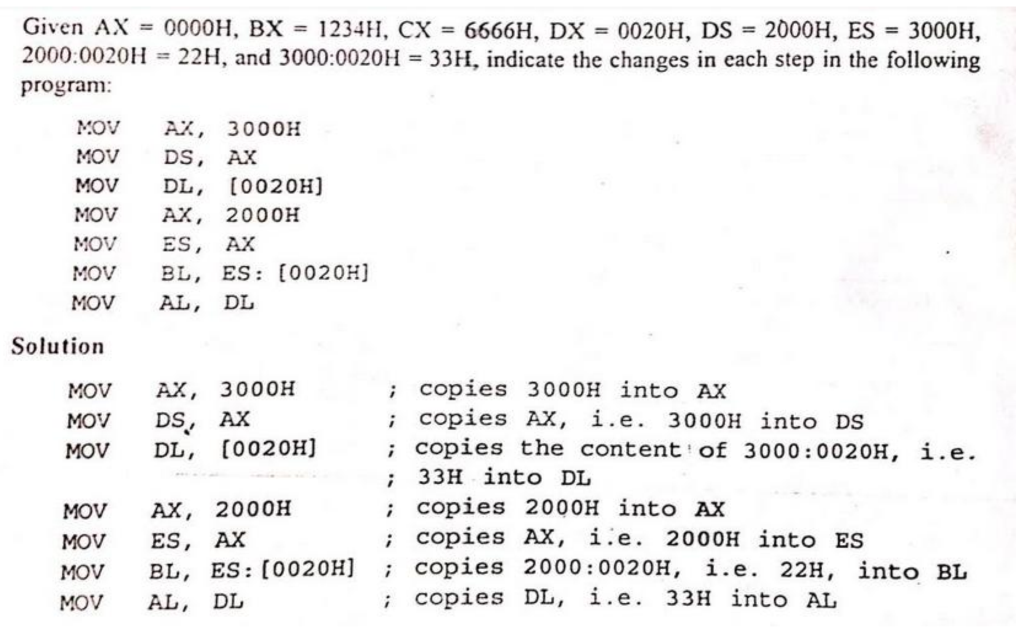
Tell which of the following instructions are legal. If legal, give the source offset address and the result or number moved.

```
a. MOV  BX, [BX]
b. MOV  CX, [SI]
c. MOV  BX, [AX]
d. ADD  [SI], [DI]
e. INC  [DI]
```

Solution:

	Source offset	*Result*
a.	1000h	1BACh
b.	2000h	20FEh
c.	illegal source register	(must be BX, SI, or DI)
d.	illegal memory-memory addition	
e.	3000h	031Eh

Another example



Given AX = 0000H, BX = 1234H, CX = 6666H, DX = 0020H, DS = 2000H, ES = 3000H, 2000:0020H = 22H, and 3000:0020H = 33H, indicate the changes in each step in the following program:

```
MOV   AX, 3000H
MOV   DS, AX
MOV   DL, [0020H]
MOV   AX, 2000H
MOV   ES, AX
MOV   BL, ES: [0020H]
MOV   AL, DL
```

Solution

```
MOV   AX, 3000H        ; copies 3000H into AX
MOV   DS, AX           ; copies AX, i.e. 3000H into DS
MOV   DL, [0020H]      ; copies the content of 3000:0020H, i.e.
                       ; 33H into DL
MOV   AX, 2000H        ; copies 2000H into AX
MOV   ES, AX           ; copies AX, i.e. 2000H into ES
MOV   BL, ES:[0020H]   ; copies 2000:0020H, i.e. 22H, into BL
MOV   AL, DL           ; copies DL, i.e. 33H into AL
```

Register indirect addressing mode working

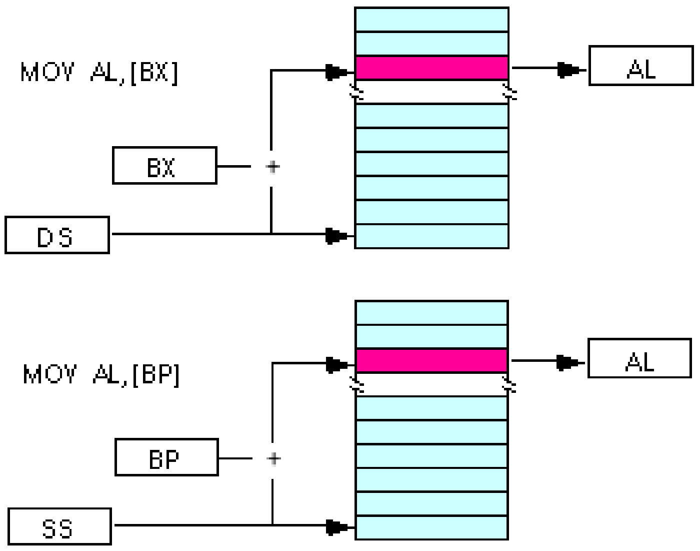


Figure 5 Register indirect addressing mode working

Based Relative Addressing

In the based addressing mode, the physical address of the operand is obtained by adding a direct or indirect displacement of 8 bit or 16 bit from the contents of either base register **BX** or base pointer register **BP** and the current value in **DS** and **SS** respectively.

Example MOV AX, D8[BX] or MOV AX, d16[BX]

The instruction copies a word from data segment memory at the offset of d8/d16 + [BX]. If d8 is 04h, and BX contains 1234H, and DS contains 2000H, then the execution of MOV AX, d8[BX] instruction copies the content of 21238H (2000H:1234H+04H) into AL and the content of 21239H into AH.

Example MOV AX, D8[BP] or MOV AX, d16[BP]

The operation performs similar operation to that of using BX, but copies a word from the stack segment.

Example MOV [BX] . BETA , AL

This instruction uses base register BX and direct displacement BETA to derive the EA of the destination operand. The based addressing mode is implemented by specifying the base register in the brackets followed by a period and direct displacement .The source operand is located in the byte accumulator AL.

Based relative addressing mode working

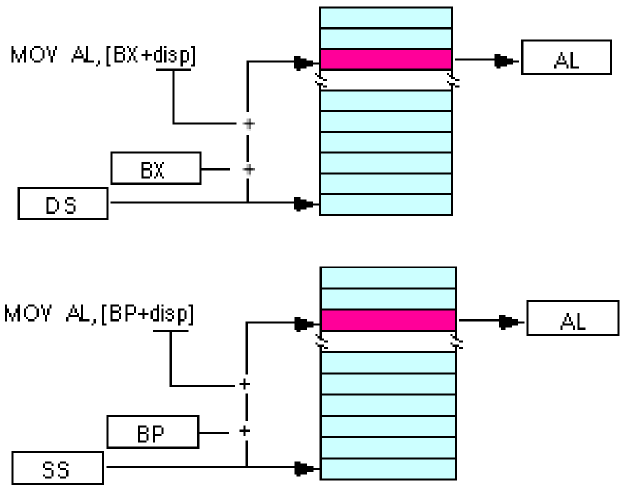


Figure 6 Based relative addressing mode working

PREPARED BY
SHAHADAT HUSSAIN PARVEZ

Indexed Relative Addressing

Indexed addressing mode works identically to the based addressing but it uses the contents of the index registers instead of BX or BP, in the generation of the physical address.

Example MOV AX, D8[SI] or MOV AX, d16[SI]

The instruction copies a word memory location at the offset of d8/d16 + [SI] in the data segment into AX. If d8 is 04h, and SI contains 1234H, and DS contains 2000H, then the execution of MOV AX, d8[BX] instruction copies the content of 21238H (2000H:1234H+04H) into AL and the content of 21239H into AH.

The working of indexed relative addressing mode is similar to that of Based relative addressing mode working, but only You may substitute si or di in the figure 6 to obtain the [si+disp] and [di+disp] addressing modes.

As seen in the above two sections based addressing and indexed addressing is similar and have similar notation. There are many notations that can be used.

- Any of the following expressions are equivalent:
    - **[register + displacement]**
    - **[displacement + register]**
    - **[register] + displacement**
    - **displacement + [register]**
    - **displacement[register]**
- The register must be **bx**, **bp**, **si**, or **di**.
- If **bx**, **si**, or **di** is used, **ds** contains the segment number
- If **bp** is used, **ss** has the segment number
- The addressing is called ***based*** if **bx** or **bp** is used; it is called ***indexed*** if **si** or **di** is used

Based Indexed Relative Addressing

Combining the based addressing mode and the indexed addressing mode together results in a new, more powerful mode known as based indexed addressing.

Example MOV AX, D8[BX][SI] or MOV AX, d16[BX][SI]

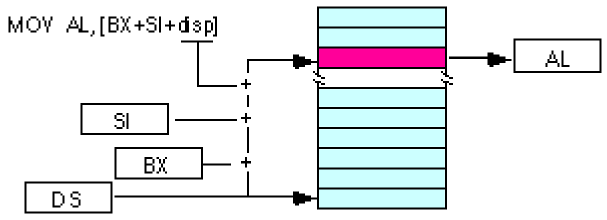
The instruction copies a word memory location at the offset of d8/d16 + [BX] + [SI] in the data segment into AX. If d8 is 04h,BX contains 0002H and SI contains 1234H, and DS contains 2000H, then the execution of MOV AX, d8[BX] instruction copies the content of 2123AH (2000H:1234H+04H) into AL and the content of 2123BH into AH.

Example: MOV AH , [BX] . BETA [SI]

Here the source operand is accessed using the based indexed addressing mode. The effective address of the source operand is obtained as EA=(BX)+BETA+(SI)

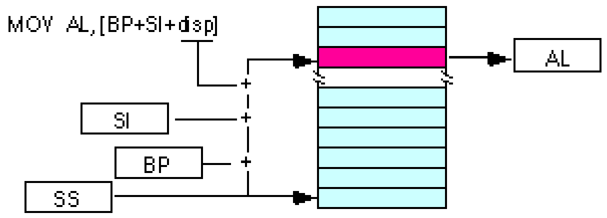
- In this mode the offset address is the sum of:
    - the contents of a base register (**bx** or **bp**)
    - the contents of an index register (**si** or **di**)
    - optionally, a variable's offset address
    - optionally, a positive or negative constant
- There are many valid ways to write the operand, some of them are:
    - **[base + index + variable + constant]**
    - **variable[base + index + constant]**
    - **constant[base + index + variable]**

The following are some examples of Based-Indexed addressing modes:



mov al, disp[bx][si]
mov al, disp[bx+di]
mov al, [bp+si+disp]
mov al, [bp][di][disp]

You may substitute di in the figure above to produce the [bx+di+disp] addressing mode.



You may substitute di in the figure above to produce the [bp+di+disp] addressing mode. Suppose bp contains 1000h, bx contains 2000h, si contains 120h, and di contains 5. Then mov al,10h[bx+si] loads al from address DS:2130; mov ch,125h[bp+di] loads ch from location SS:112A; and mov bx,cs:2[bx][di] loads bx from location CS:2007.

Use of Based, Indexed, and Base-Indexed Modes

- Based and indexed addressing mode is often used for array and string processing
- Based-indexed addressing mode can be used for two dimensional arrays
- We will discuss these in greater detail later

String addressing

Used for string instructions

EA (Effective Address) of source data is stored in SI reg. and the EA of destination data is stored in DI reg.

I/O addressing

I/O addressing is used for I/O operation. The I/O ports can be also addressed as : ***Ex: OUT PORT_ADD, AL** or **OUT AL, PORT_ADD***

I/O addressing can be of two types

- Direct I/O addressing
- Indirect I/O addressing

Direct I/O port addressing

- Used to access data from standard I/O mapped devices or ports.
- Example: IN AL, [09H]: the content of port with address 09H is moved to AL

Indirect I/O port addressing

- Used to access data from standard I/O mapped devices or ports and the instruction will specify the name of the register which holds the port address.
- Example: OUT [DX], AX : the content of AX is moved to port whose address is specified by DX

Relative addressing

Relative Addressing Mode –Jumps and CALL instructions. Jump and call instructions can be used for short jumps within currently selected 64 KB code segment, as well as for far jumps anywhere within 1 MB of memory. All conditional jump instructions can be used to jump within approximately +127 to -127 bytes from current instruction. Ex: JMP label_name

Jump and Call Instructions.

JZ label, JC Label, JMP Label

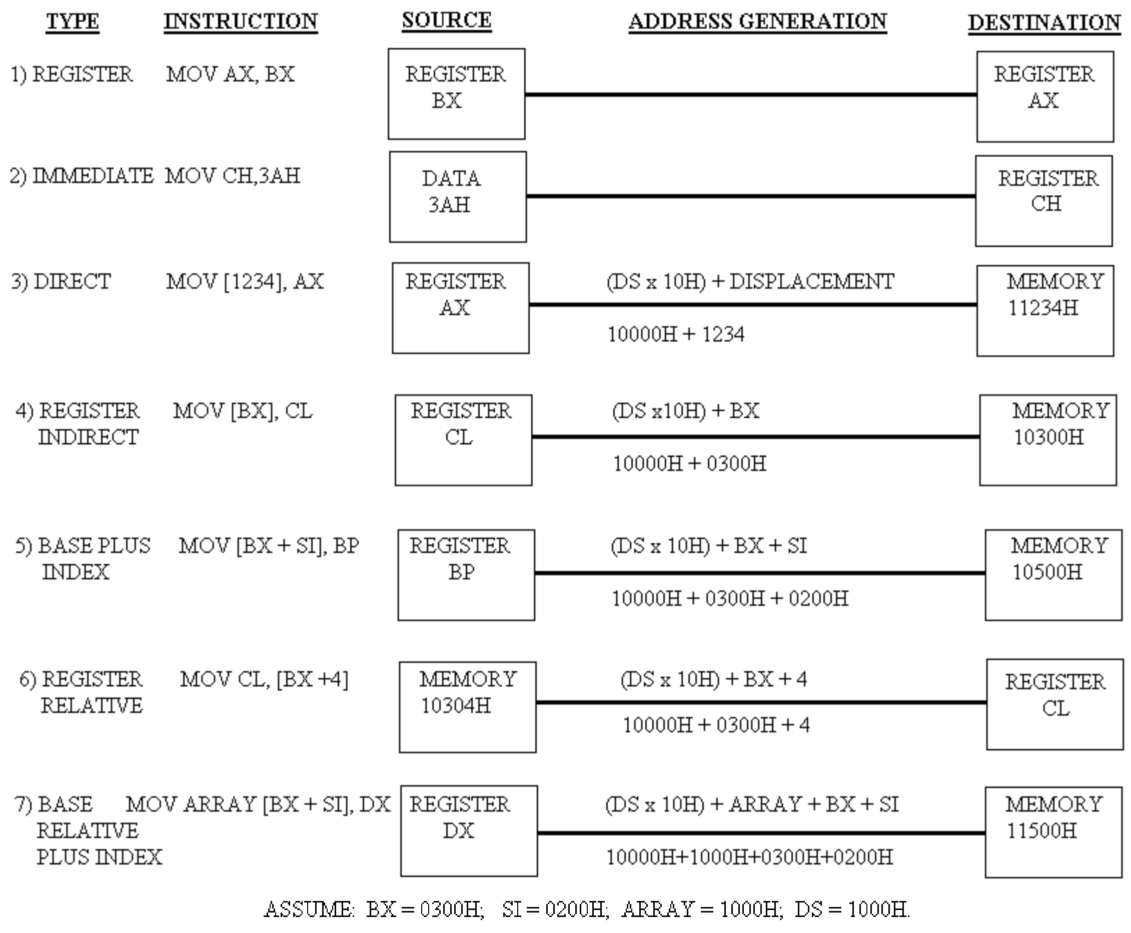
Implied addressing

The data value/data address is implicitly associated with the instruction. The instructions have no operand. Examples include

STC → Set Carry flag

CLC → Clear Carry Flag

Seven of the most commonly used addressing modes are summarized in the figure below.



TYPE	INSTRUCTION	SOURCE	ADDRESS GENERATION	DESTINATION
1) REGISTER	MOV AX, BX	REGISTER BX		REGISTER AX
2) IMMEDIATE	MOV CH,3AH	DATA 3AH		REGISTER CH
3) DIRECT	MOV [1234], AX	REGISTER AX	(DS x 10H) + DISPLACEMENT 10000H + 1234	MEMORY 11234H
4) REGISTER INDIRECT	MOV [BX], CL	REGISTER CL	(DS x10H) + BX 10000H + 0300H	MEMORY 10300H
5) BASE PLUS INDEX	MOV [BX + SI], BP	REGISTER BP	(DS x 10H) + BX + SI 10000H + 0300H + 0200H	MEMORY 10500H
6) REGISTER RELATIVE	MOV CL, [BX +4]	MEMORY 10304H	(DS x 10H) + BX + 4 10000H + 0300H + 4	REGISTER CL
7) BASE RELATIVE PLUS INDEX	MOV ARRAY [BX + SI], DX	REGISTER DX	(DS x 10H) + ARRAY + BX + SI 10000H+1000H+0300H+0200H	MEMORY 11500H

ASSUME: BX = 0300H; SI = 0200H; ARRAY = 1000H; DS = 1000H.

6. Explain the addressing modes of 8086.
7. Marut chapter 8 exercise question no 1