CSE 230

Microprocessor 8086

Lecture 5

Micro computers

Slides in this course are taken mainly from Mazidi book and other non copy righted sources

Some slides are taken from the Mazidi book slides and from Dr. Ali Ziya Alkar (Ph.D University of Colorado, Boulder)

Microprocessor

- Assume that an imaginary CPU has registers called A,B,C, and D.
- It has an 8-bit data bus and a 16-bit address bus.
- Therefore the CPU can access memory from addresses 0000h to FFFFh for a total of 2^16 locations
- The action to be performed by the CPU is to put a hexadecimal value 21 into register A, and add to register A values 42h and 12h.
- Assume that the code for the CPU to move a value to register A is 1011 0000b (B0h) and the code for adding a value to register A is 0000 0100b (04h)

Action	Code	Data
Move 21h to A	B0h	21h
Add 42h to A	04h	42h
Add 12h to A	04h	12h

Microprocessor

Memory Address	Content of memory
1400h	B0h
1401h	21h
1402h	04h
1403h	42h
1404h	04h
1405h	12h
1406h	F4h (the code for halt)

- Assume program is stored at memory locations starting at 1400h

Microprocessor

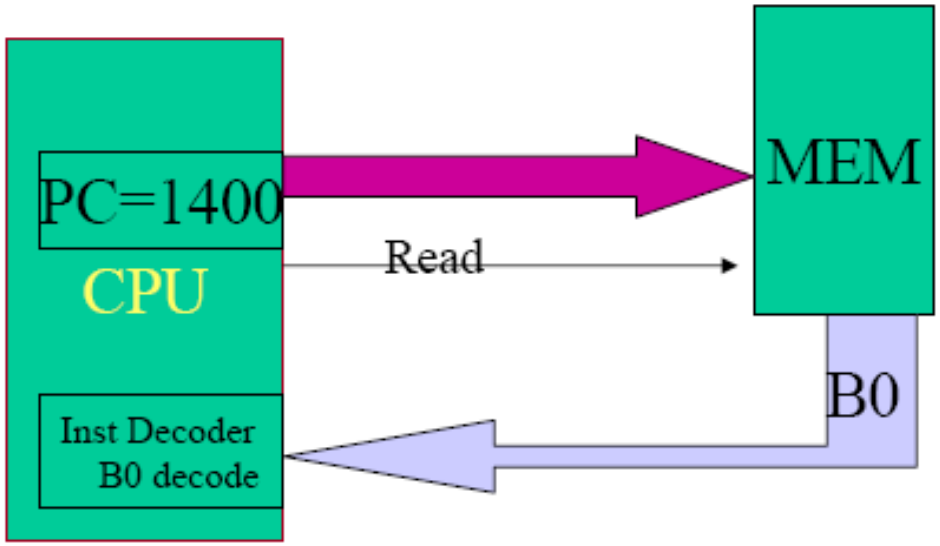
- ACTION Code Data
- Move value 21 into register A B0H21H
- Add value 42H to register A 04H42H
- Add value 12H to register A 04H12H

Memory Address	Contents of memory address
1400	(B0) the code for move to A
1401	(21) the value for A
1402	(04) the code for adding a value to A
1403	(42) the value to be added
1404	(04) the code for adding a value to A
1405	(12) the value to be added
1406	(F4) the code for halt

Microprocessors

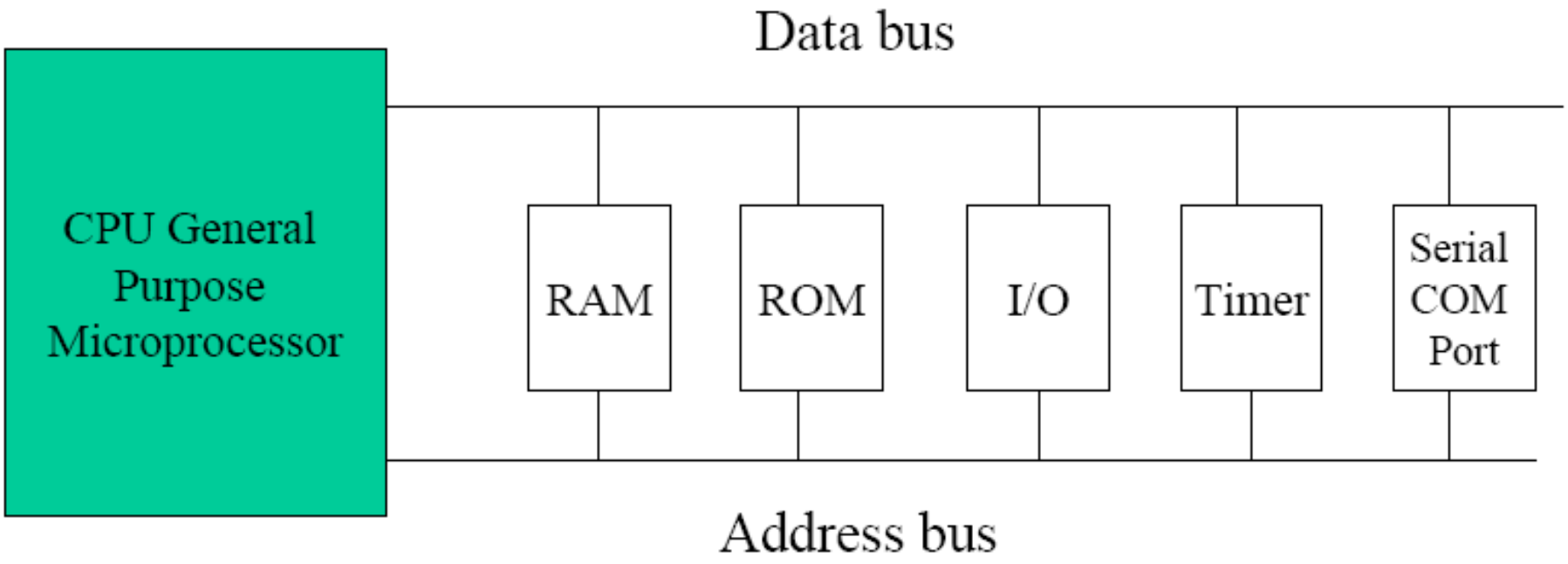
1- The CPU program counter can have any value between 0000 ->FFFF H. This one is set to start with 1400

2- The CPU puts out 1400. The memory circuitry finds the location. Activates the read signal, indicating the memory location 1400. B0 is put on the bus and brought to the CPU



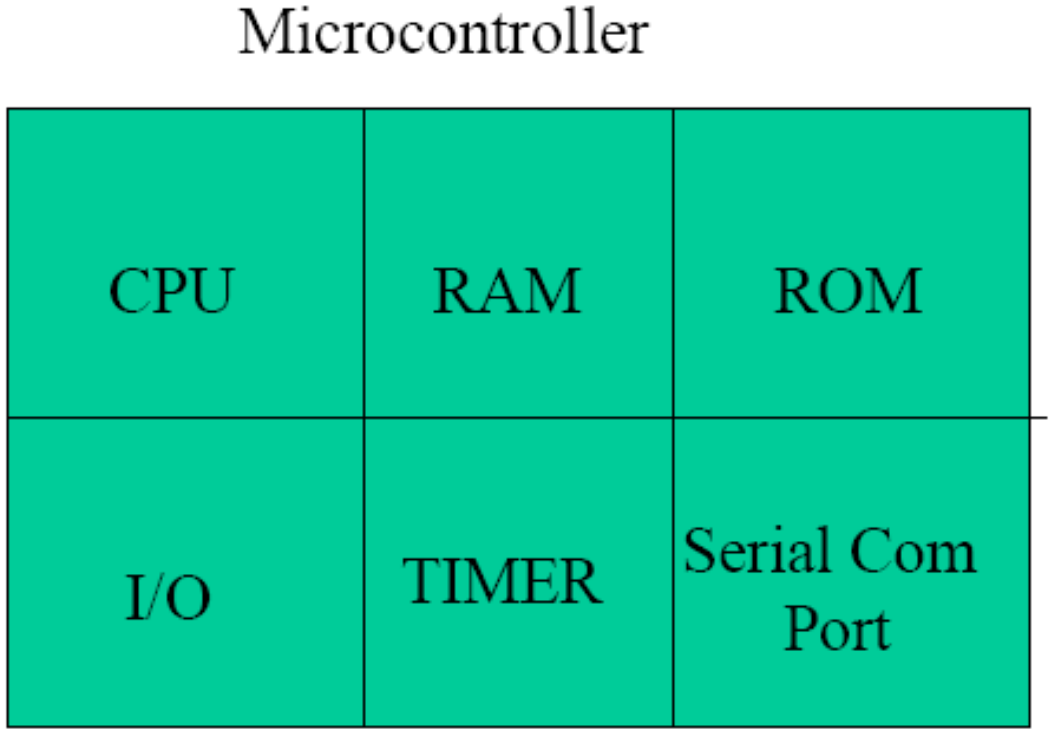
B0 is decoded internally it now knows it needs to fetch the next byte!. It brings 21h from 1401. The program counter automatically increments itself to the next location to fetch the next data/instruction.

Microprocessors



These general microprocessors contain no RAM, ROM, or I/O ports on the chip itself
Ex. Intel's x86 family (8088, 8086, 80386, 80386, 80486, Pentium)
Motorola's 680x0 family (68000, 68010, 68020, etc)

Microcontroller

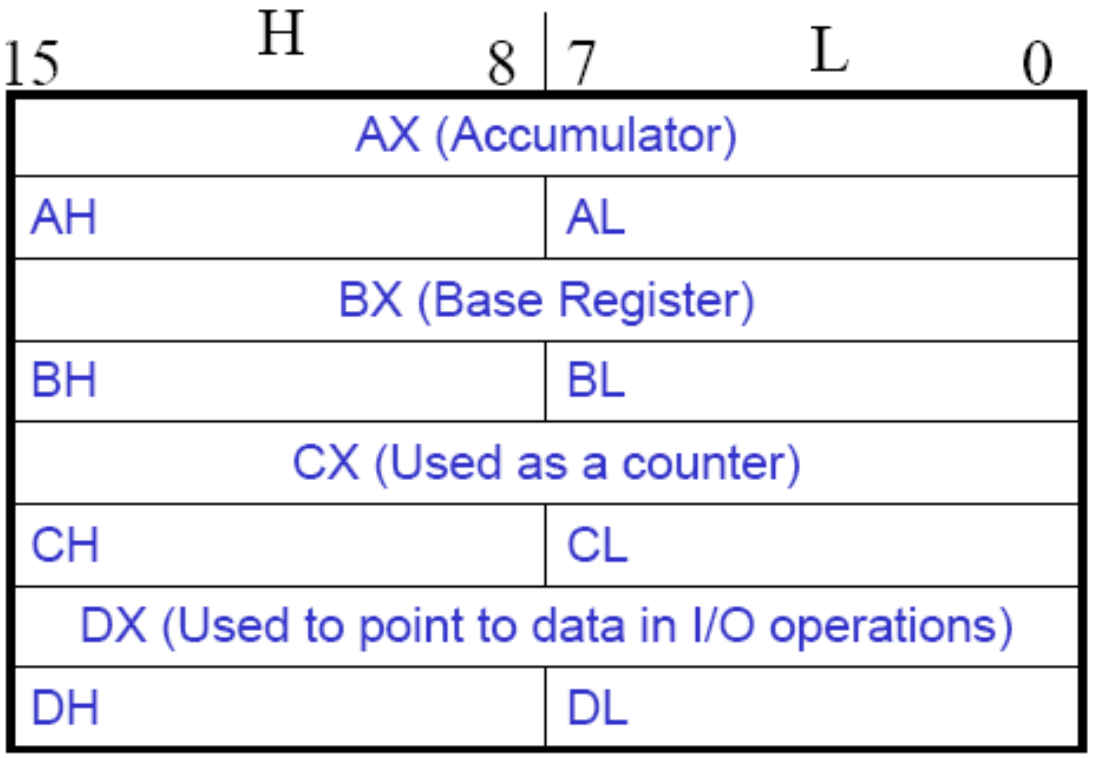


- A microcontroller has a CPU in addition to a fixed amount of RAM, ROM, I/O ports on one single chip; this makes them ideal for applications in which cost and space are critical
- Example: a TV remote control does not need the computing power of a 486

Registers of the 8086/80286 by Category

Category	Bits	Register Names
General	16	AX,BX,CX,DX
	8	AH,AL,BH,BL,CH,CL,DH,DL
Pointer	16	SP (Stack Pointer), Base Pointer (BP)
Index	16	SI (Source Index), DI (Destination Index)
Segment	16	CS(Code Segment) DS (Data Segment) SS (Stack Segment) ES (Extra Segment)
Instruction	16	IP (Instruction Pointer)
Flag	16	FR (Flag Register)

General Purpose Registers



- **Data Registers** are normally used for storing temporary results that will be acted upon by subsequent instructions
- Each of the registers is 16 bits wide (AX, BX, CX, DX)
- General purpose registers can be accessed as either 16 or 8 bits e.g., AH: upper half of AX, AL: lower half of AX

Data Registers

Register	Operations
AX	Word multiply, word divide, word I/O
AL	Byte multiply, byte divide, byte I/O, decimal arithmetic
AH	Byte multiply, byte divide
BX	Store address information
CX	String operations, loops
CL	Variable shift and rotate
DX	Word multiply, word divide, indirect I/O

Pointer and Index Registers

SP	Stack Pointer
BP	Base Pointer
SI	Source Index
DI	Destination Index
IP	Instruction Pointer

The registers in this group are all 16 bits wide
Low and high bytes are not accessible
These registers are used as memory pointers

- Example: MOV AH, [SI]

Move the byte stored in memory location whose address is contained in register SI to register AH

IP is not under direct control of the programmer

MOV Instruction

- MOV destination,source
  - **8 bit moves**
    - MOV CL,55h
    - MOV DL,CL
    - MOV BH,CL
    - Etc.
  - **16 bit moves**
    - MOV CX,468Fh
    - MOV AX,CX
    - MOV BP,DI
    - Etc.

MOV Instruction

- **Data can be moved among all registers but data cannot be moved directly into the segment registers** (CS,DS,ES,SS).
  - To load as such, first load a value into a non-segment register and then move it to the segment register

```
MOV AX,2345h
MOV DS,AX
```

- **Moving a value that is too large into a register will cause an error**

```
MOV BL,7F2h       ; illegal
MOV AX,2FE456h    ; illegal
```

- **If a value less than than FFh is moved into a 16 bit register. The rest of the bits are assumed to be all zeros.**

```
MOV BX,5        ; BX = 0005 with BH  = 00 and BL = 05
```

MOV Instruction

• MOV AX,58FCH	√
• MOV DX,6678H	√
• MOV SI,924BH	√
• MOV BP,2459H	√
• MOV DS,2341H	x
• MOV CX,8876H	√
• MOV CS,3F47H	x
• MOV BH,99H	√

ADD Instruction

- ADD destination,source
- The ADD instruction tells the CPU to add the source and destination operands and put out the results in the destination

DESTINATION = DESTINATION + SOURCE

```
MOV AL,25H
MOV BL,34h
ADD AL,BL   ; (AL should read 59h once the instruction is executed)
MOV DH,25H
ADD DH,34h   ; (AL should read 59h once the instruction is executed)
```

Immediate operand

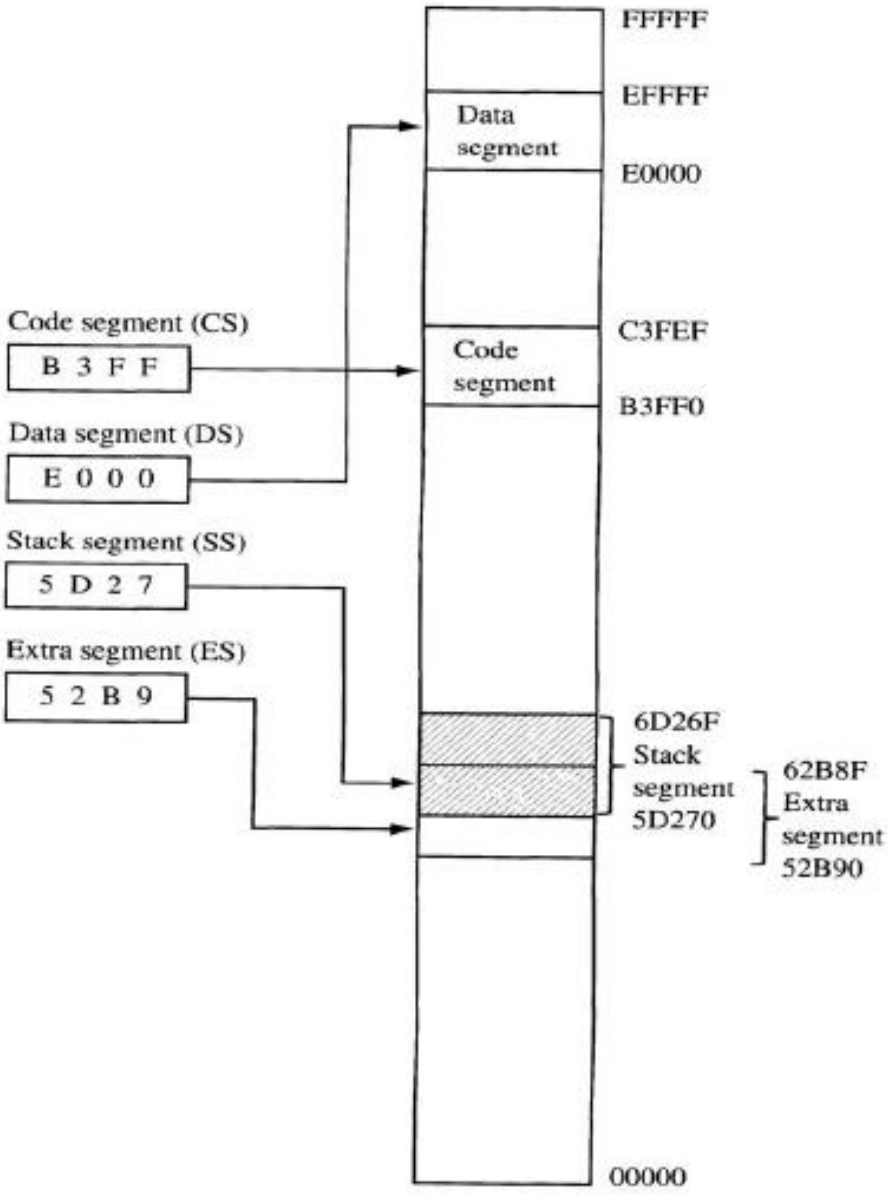
Origin and Definition of a Segment

- A segment is an area of memory that includes up to 64 Kbytes and begins on an address divisible by 16 (such an address ends with an hex digit 0h or 0000b)
  - 8085 could address 64Kbytes 16 address lines
- In the 8085, 64 K is for code, data, and stack
- In the 8086/88, 64 K is assigned to each category
  - Code segment
  - Data segment
  - Stack Segment
  - Extra Segment

Advantages of Segmented Memory

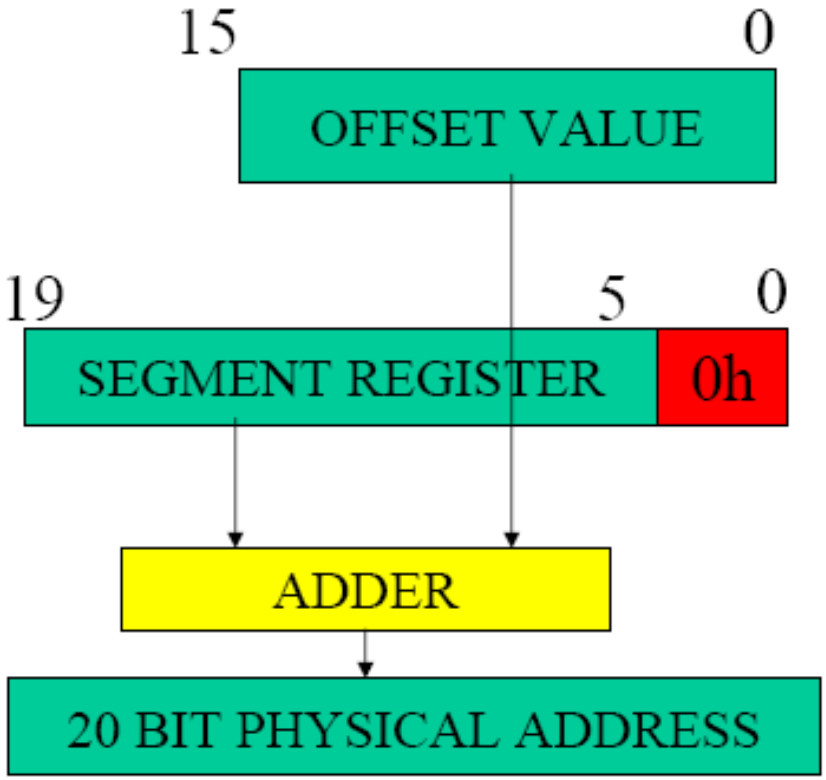
- One program can work on several different sets of data. This is done by reloading register DS to a new value.
- Programs that reference logical addresses can be loaded and run anywhere in the memory: relocatable
- Segmented memory introduces extra complexity in both hardware in that memory addresses require two registers.
- They also require complexity in software in that programs are limited to the segment size
- Programs greater than 64 KB can be run on 8086 but the software needed is more complex as it must switch to a new segment.
- Protection among segments is provided.

Segment Registers



Logical and Physical Addresses

- Addresses within a segment can range from address 0 to address FFFFh. This corresponds to the 64Kbyte length of the segment called an *offset*
- An address within a segment *logical address*
- *Ex.* Logical address 0005h in the code segment actually corresponds to B3FF0h + 5 = B3FF5h.



Example 1:
Segment base value: 1234h
Offset: 0022h

12340h
+ 0022h

12362h is the physical 20 bit address

Two different logical addresses may correspond to the same physical address.
D470h in ES 2D90h in SS
ES:D470h SS:2D90h

Example

- If DS=7FA2H and the offset is 438EH

a) Calculate the physical address

7FA20 + 438E = 83DAE

b) calculate the lower range

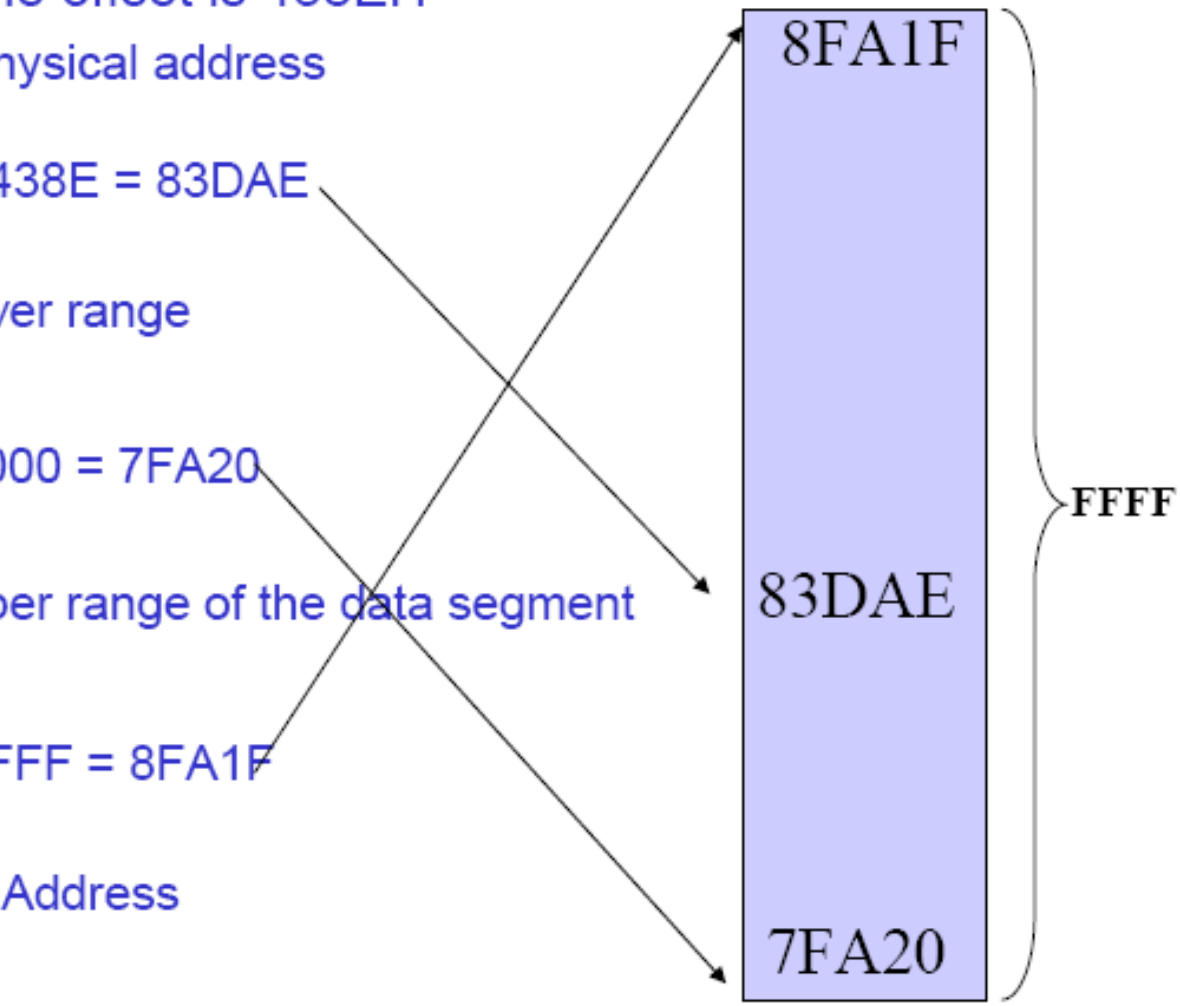
7FA20 + 0000 = 7FA20

c) Calculate the upper range of the data segment

7FA20 + FFFF = 8FA1F

d) Show the logical Address

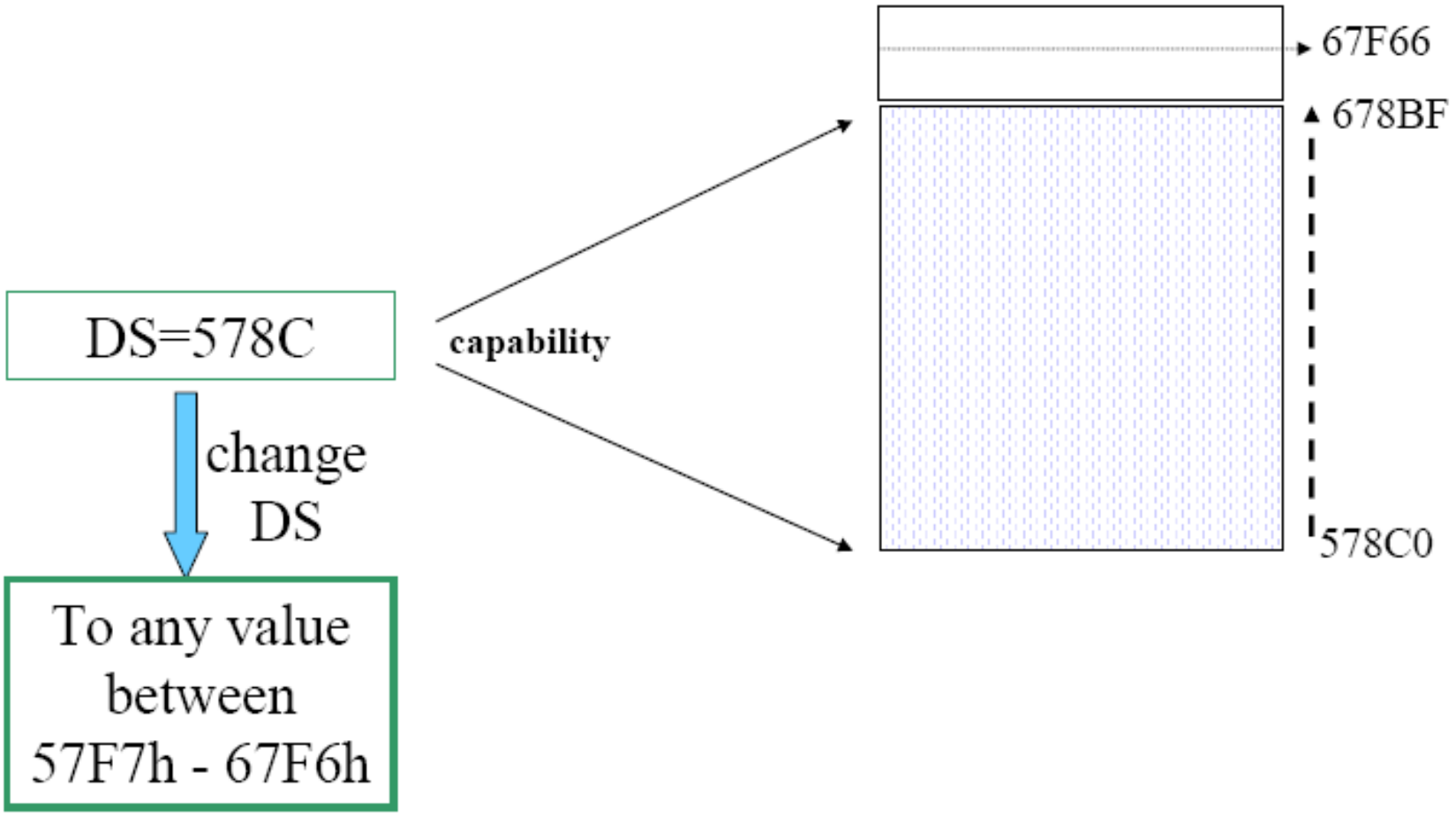
7FA2:438E



Example

Question:

Assume DS=578C. To access a Data in 67F66 what should we do?



Code Segment

- To execute a program, the 8086 fetches the instructions (opcodes and operands) from the code segment
- The logical address is in the form CS:IP

- Example: If CS = 24F6h and IP = 634Ah, show
  - The logical address
  - The offset address

  and calculate
  - The physical address
  - The lower range
  - The upper range

Logical Address vs Physical Address in the CS

CS:IP	Machine Language	Mnemonics
1132:0100	B057	MOV AL,57h
1132:0102	B686	MOV DH,86h
1132:0104	B272	MOV DL,72h
1132:0106	89D1	MOV CX,DX
1132:0108	88C7	MOV BH,AL
1132:010A	B39F	MOV BL,9F
1132:010C	B420	MOV AH,20h
1132:010E	01D0	ADD AX,DX
1132:0110	01D9	ADD CX,BX
1132:0112	05351F	ADD AX, 1F35h

- Show how the code resides physically in the memory

Data Segment

- Assume that a program is written to add 5 bytes of data 25h,12h,15h,1Fh, and 2Bh.
- One way to do it

```
MOV AL,00h
ADD AL, 25h
ADD AL, 12h
ADD AL,15h
ADD AL,1Fh
ADD AL,2Bh
```

- Data and code are mixed in the instructions here
- The problem with it is if the data changes, the code must be searched for every place the data is included and data retyped.
- It is a good idea then to set aside an area of memory strictly for data

Data Segment

- The data is first placed in the memory locations

  DS:0200 = 25h

  DS:0201 = 12h

  DS:0202 = 15h

  DS:0203 = 1Fh

  DS:0204 = 2Bh

- Then the program is written as

```
MOV AL,0
ADD AL,[0200]   ; bracket means add the contents of DS:0200 to AL
ADD AL,[0201]
ADD AL,[0202]
ADD AL,[0203]
ADD AL,[0204]
```

- If the data is stored at a different offset address, say 450 h, the program need to be rewritten

Data Segment

- The term pointer is used for a register holding an offset address
- Use BX as a pointer

```
MOV AL,0
MOV BX,0200h
ADD AL,[BX]
INC BX
ADD AL,[BX]
INC BX
ADD AL,[BX]
INC BX
ADD AL,[BX]
INC BX
ADD AL,[BX]
```

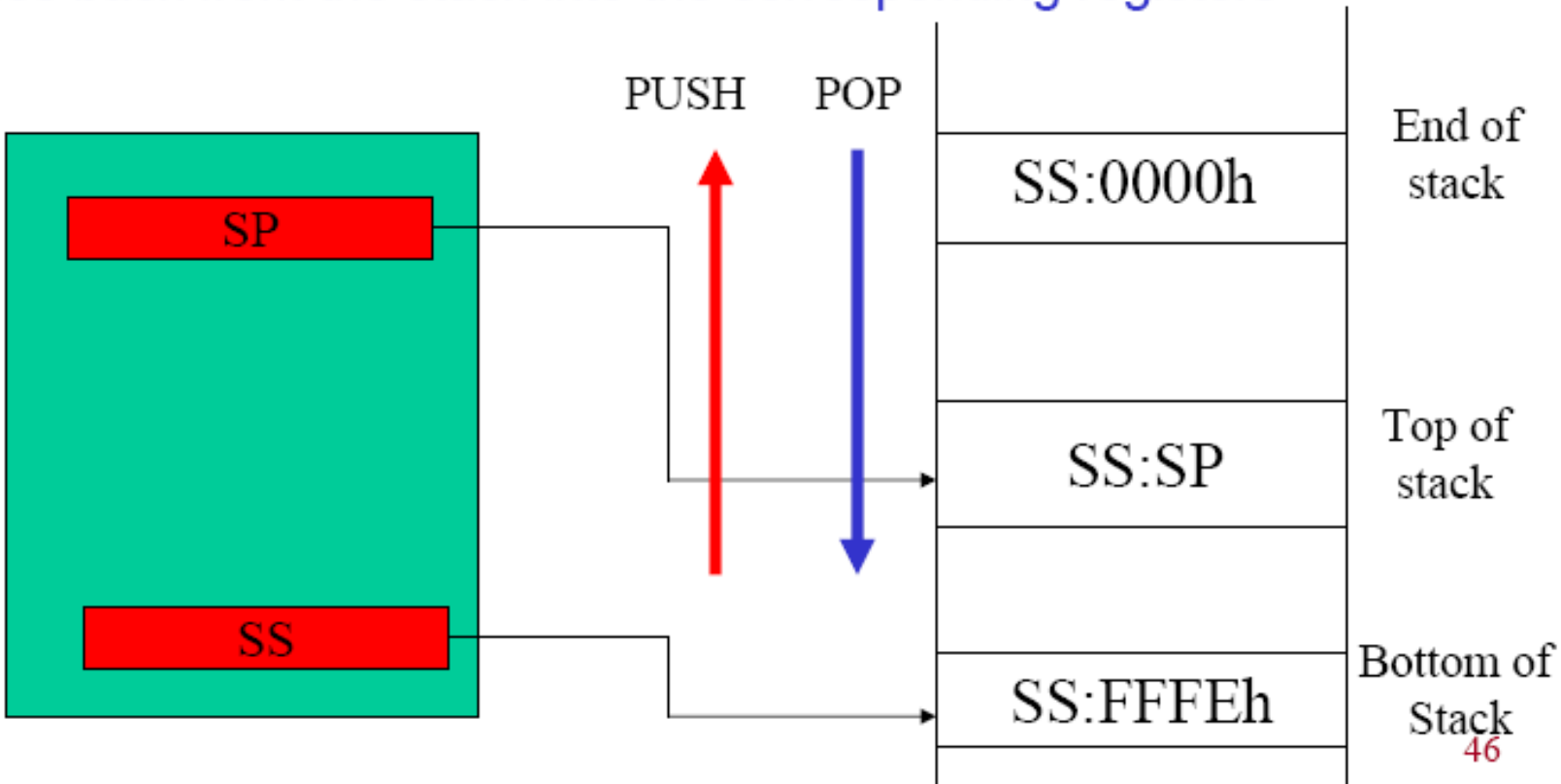
- If the offset address of data is to be changed, only one instructions will need to be modified

16 bit Segment Register Assignments

Type of Memory Reference	Default Segment	Alternate Segment	Offset
Instruction Fetch	CS	none	IP
Stack Operations	SS	none	SP,BP
General Data	DS	CS,ES,SS	BX, address
String Source	DS	CS,ES,SS	SI, DI, address
String Destination	ES	None	DI

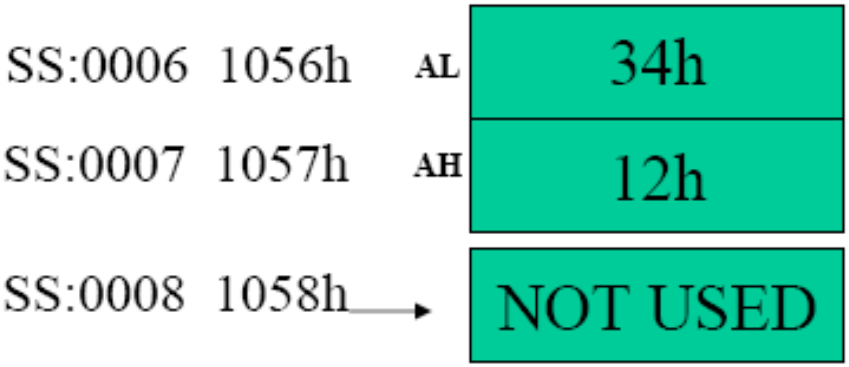
The Stack

- The stack is used for temporary storage of information such as data or addresses; for instance when a call is executed the 8088 automatically pushes the current value of CS and IP onto the stack.
- Other registers can also be pushed
- Near the end of the subroutine, pop instructions can be used to pop values back from the stack into the corresponding registers



Example for PUSH

- Given
  - SS = 0105h
  - SP = 0008h
  - AX = 1234h
  - What is the outcome of the PUSH AX instruction?
- A_{BOS} = 01050 + FFFEh = 1104h
- A_{TOS} = 01050 + 0008h = 1058h
- Decrement the SP by 2 and write AX into the word location 1056h.



Example for POP

- What is the outcome of the following

  POP AX

  POP BX

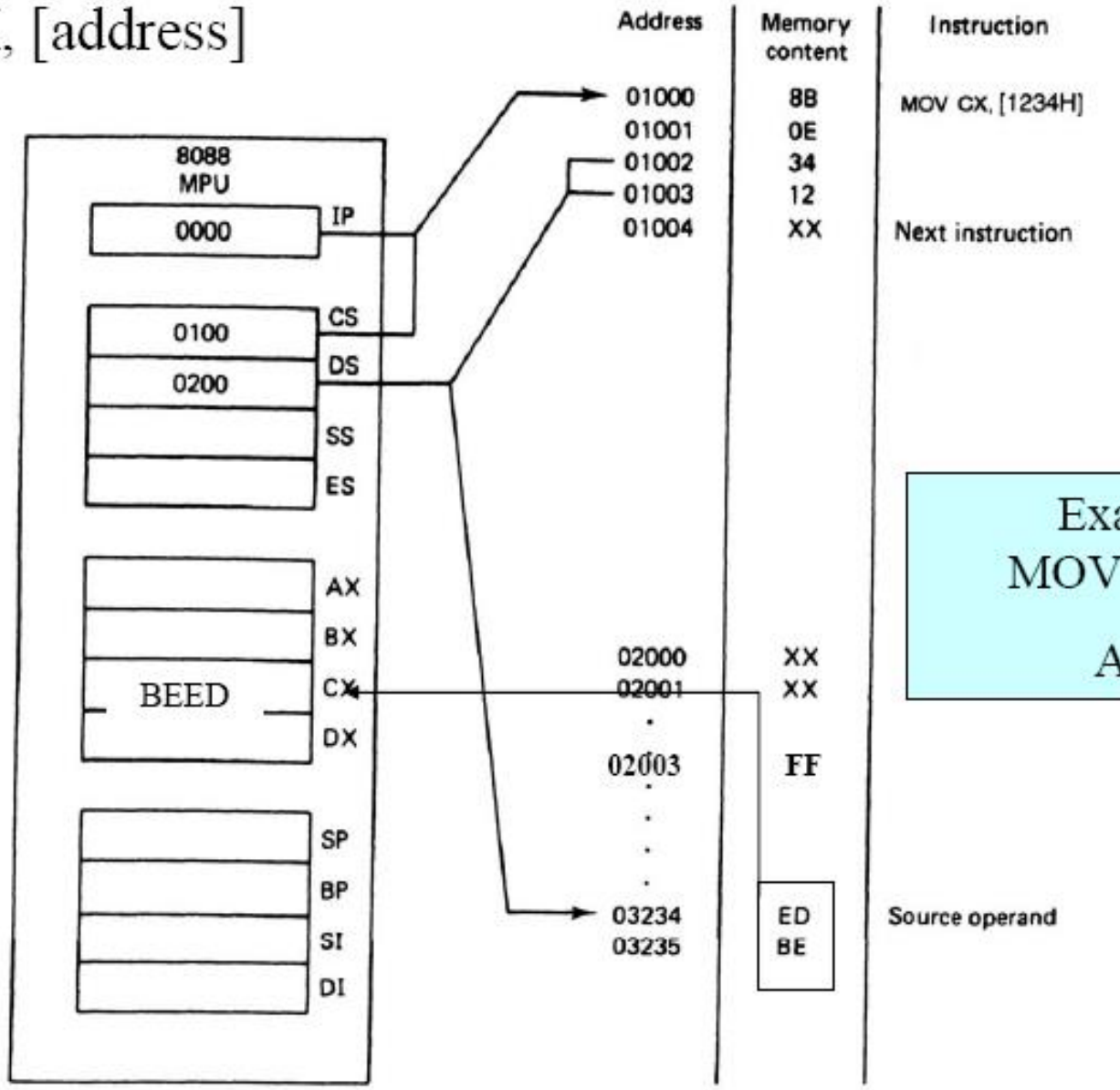
  – if originally 1058h contained AABBh?
- Read into the specified register from the stack and increment the stack pointer for each POP operation
- At the first POP
  – AX = 1234h SP = 0008h
- At the second POP
  – BX = AABBh SP = 000Ah

Addressing Modes

- When the 8088 executes an instruction, it performs the specified function on data
- These data, called operands,
  - May be a part of the instruction
  - May reside in one of the internal registers of the microprocessor
  - May be stored at an address in memory
- **Register Addressing Mode**
  - MOV AX, BX
  - MOV ES,AX
  - MOV AL,BH
- **Immediate Addressing Mode**
  - MOV AL,15h
  - MOV AX,2550h
  - MOV CX,625

Direct Addressing Mode

MOV CX, [address]

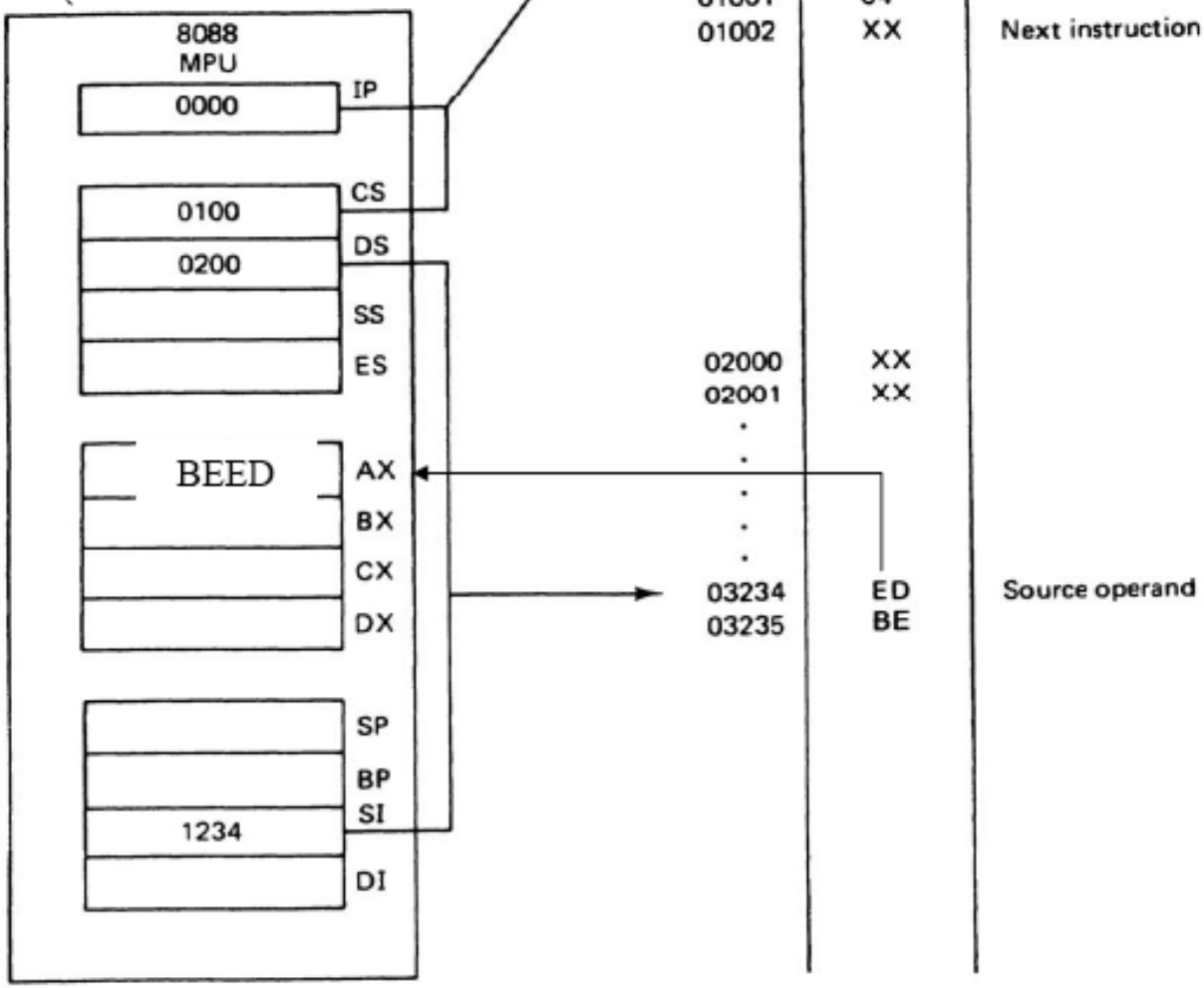


Example:
MOV AL,[03]

AL=?

Register Indirect Addressing Mode

MOV AX, { BX / DI / SI }



Example for Register Indirect Addressing

- Assume that DS=1120, SI=2498 and AX=17FE show the memory locations after the execution of:

MOV [SI],AX

DS (Shifted Left) + SI = 13698.

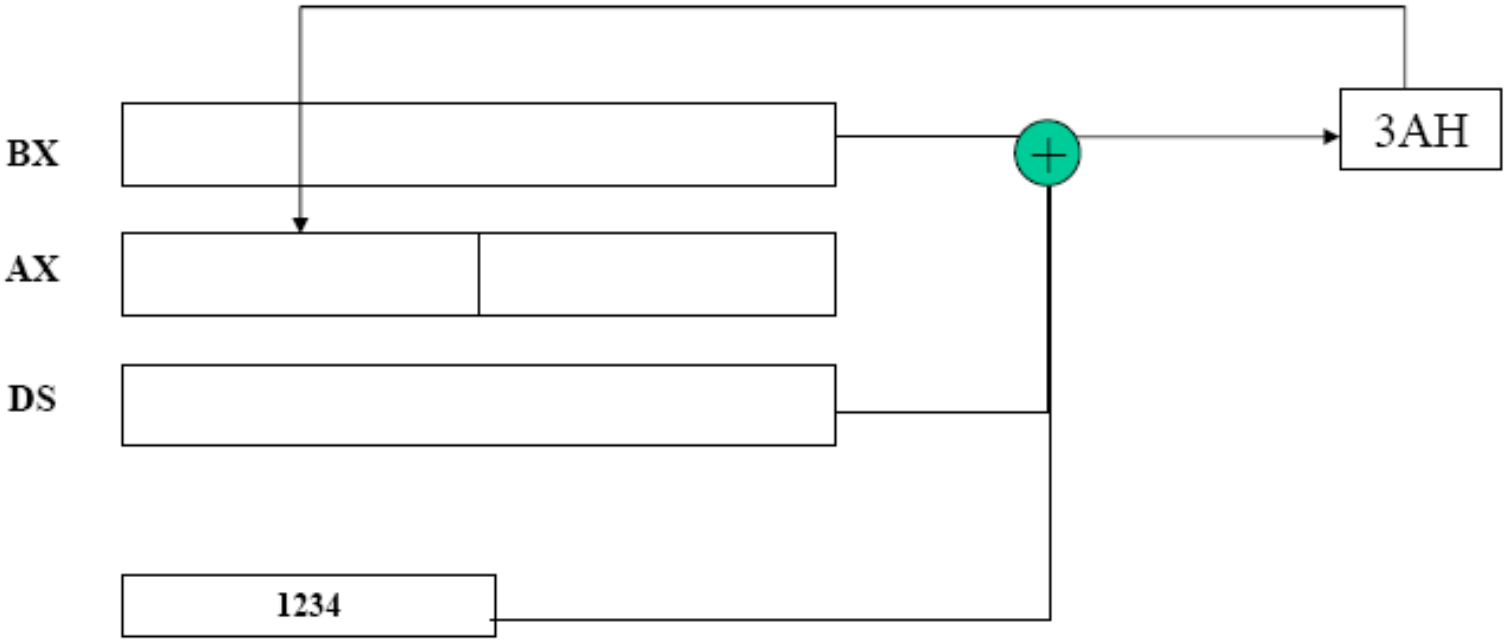
With little endian convention:

Low address 13698 → FE

High Address 13699 → 17

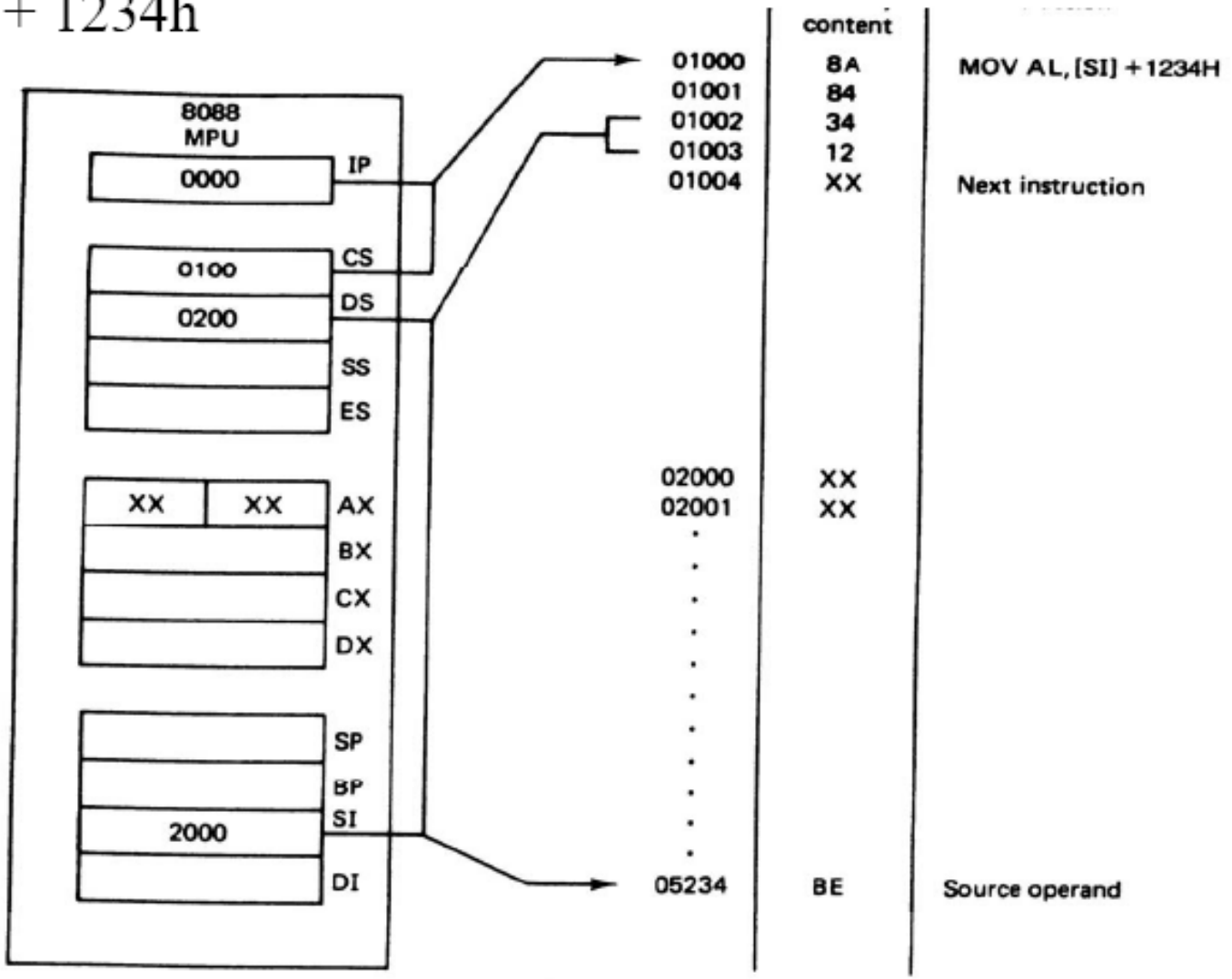
Based-Relative Addressing Mode

MOV AH, [DS:BX / SS:BP] + 1234h



Indexed Relative Addressing Mode

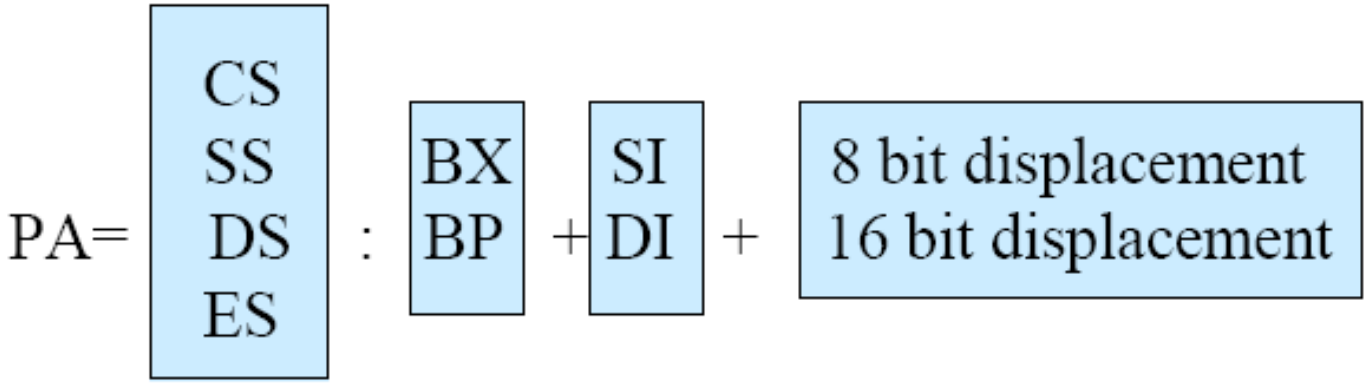
MOV AH, [$\begin{matrix}SI\\DI\end{matrix}$] + 1234h



Example: What is the physical address MOV [DI-8],BL if DS=200 & DI=30h ?
DS:200 shift left once 2000 + DI + -8 = 2028

Based-Indexed Relative Addressing Mode

- Based Relative + Indexed Relative
- We must calculate the PA (physical address)

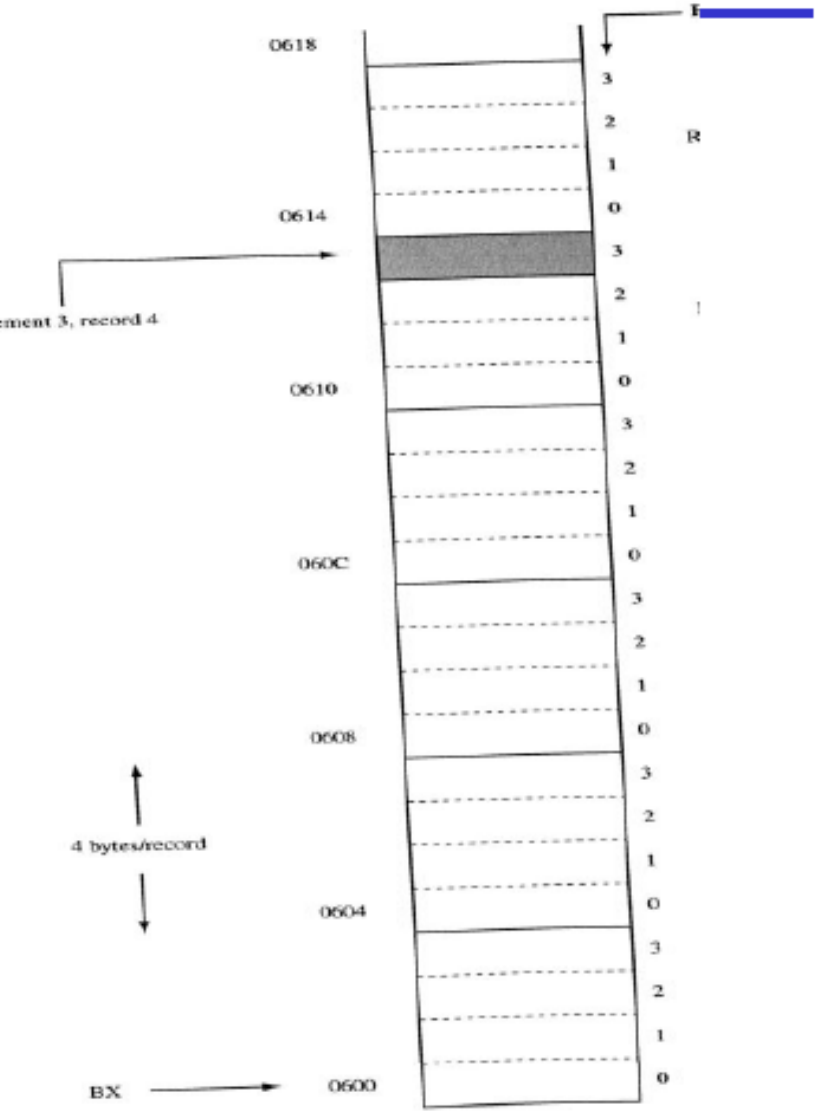


MOV AH,[BP+SI+29]
or
MOV AH,[SI+29+BP]
or
MOV AH,[SI][BP]+29

} The register order does not matter

Based-Indexed Addressing Mode

ure 4–4.
a base + index +
placement
dressing mode can
used to access a
rticular element in a
rticular record of an
ray.




```
MOV BX, 0600h
MOV SI, 0010h ; 4 records, 4 elements each.
MOV AL, [BX + SI + 3]
```

OR

```
MOV BX, 0600h
MOV AX, 004h ;
MOV CX,04;
MUL CX
MOV SI, AX
MOV AL, [BX + SI + 3]
```

Summary of the addressing modes

Addressing Mode	Operand	Default Segment
Register	Reg	None
Immediate	Data	None
Direct	[offset]	DS
Register Indirect	[BX] [SI] [DI]	DS DS DS
Based Relative	[BX]+disp [BP]+disp	DS SS
Indexed Relative	[DI]+disp [SI]+disp	DS DS
Based Indexed Relative	[BX][SI or DI]+disp [BP][SI or DI]+disp	DS SS