Lecture 18: 80x86 Memory Organization

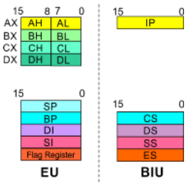
Seyed-Hosein Attarzadeh-Niaki

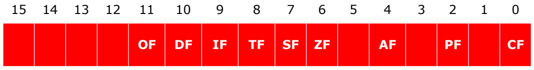
Review

- 8086 internal structure
- 8086/88 pins, signals, and buses

Summary of 8086 Registers I

8086 registers categorized into 6 groups





Sl.No.	Type	Register width	Name of register
1	General purpose register	16 bit	AX, BX, CX, DX
		8 bit	AL, AH, BL, BH, CL, CH, DL, DH
2	Pointer register	16 bit	SP, BP
3	Index register	16 bit	SI, DI
4	Instruction Pointer	16 bit	IP
5	Segment register	16 bit	CS, DS, SS, ES
6	Flag (PSW)	16 bit	Flag register

Summary of 8086 Registers II

Register	Name of the Register	Special Function
AX	16-bit Accumulator	Stores the 16-bit results of arithmetic and logic operations
AL	8-bit Accumulator	Stores the 8-bit results of arithmetic and logic operations
BX	Base register	Used to hold base value in base addressing mode to access memory data
CX	Count Register	Used to hold the count value in SHIFT, ROTATE and LOOP instructions
DX	Data Register	Used to hold data for multiplication and division operations
SP	Stack Pointer	Used to hold the offset address of top stack memory
BP	Base Pointer	Used to hold the base value in base addressing using SS register to access data from stack memory
SI	Source Index	Used to hold index value of source operand (data) for string instructions
DI	Data Index	Used to hold the index value of destination operand (data) for string operations

Outline

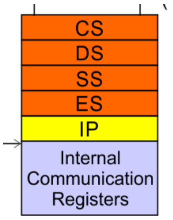
- 80x86 memory organization
  - Memory segments
- Addressing modes

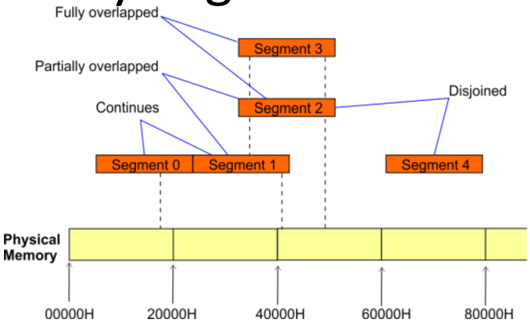
Program Segments

- A typical program on 8086 consists of at least three *segments*
  - code segment: contains instructions that accomplish certain tasks
  - data segment: stores information to be processed
  - stack segment: store information temporarily
- What is a segment?
  - A memory block includes up to 64KB. Why?
  - Begins on an address evenly divisible by 16, i.e., an address looks like in XXXX0H. Why?

Memory Segments

Segment Registers





- 8086's 1-megabyte memory is divided into segments of up to 64K bytes.
- The 8086 can directly address four segments (256K bytes within the 1M byte of memory) at a particular time.
- Programs obtain access to code and data in the segments by changing the segment register content to point to the desired segments.

Segment Registers

Code Segment Register

- CS contains the base or start of the current code segment; IP contains the distance or offset from this address to the next instruction byte to be fetched.

Data Segment Register

- Points to current data segment; most instruction operands are fetched from this segment.
- The 16-bit contents of the Source Index (SI) or Destination Index (DI) or a 16-bit displacement are used as offset for computing the 20-bit physical address.

Stack Segment Register

- Points to the current stack.
- The 20-bit physical stack address is calculated from the Stack Segment (SS) and the Stack Pointer (SP) for stack instructions such as PUSH and POP.
- In based addressing mode, the 20-bit physical stack address is calculated from the Stack segment (SS) and the Base Pointer (BP).

Extra Segment Register

- Points to the extra segment in which data (in excess of 64K pointed to by the DS) is stored.
- String instructions use the ES and DI to determine the destination's 20-bit physical address.

Logical & Physical Address

Physical address

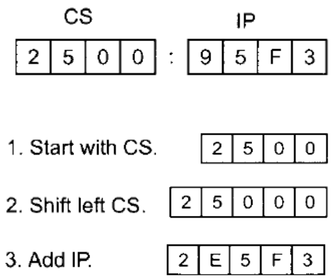
- 20-bit address that is actually put on the address bus
- A range of 1MB from 00000H to FFFFFH
- Actual physical location in memory

Logical address

- Consists of a *segment value* (determines the beginning of a segment) and an *offset address* (a relative location within a 64KB segment)
- E.g., an instruction in the code segment has a logical address in the form of
  CS (code segment register):IP (instruction pointer)

Logical & Physical Address

- logical address -> physical address
  - Shift the segment value left one hex digit (or 4 bits)
  - Then adding the above value to the offset address
  - One logical -> only one physical
- Segment range representation
  - Maximum 64KB
  - logical *2500:0000 – 2500:FFFF*
  - Physical *25000H – 34FFFH* (*25000 + FFFF*)



CS IP

2500 : 95F3

1. Start with CS. 2500

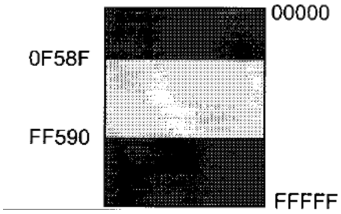
2. Shift left CS. 25000

3. Add IP. 2E5F3

Physical Address Wrap-around

- When adding the offset to the shifted segment value results in an address beyond the maximum value *FFFFFH*
- *E.g., what is the range of physical addresses if CS=FF59H?*
  - Answer:

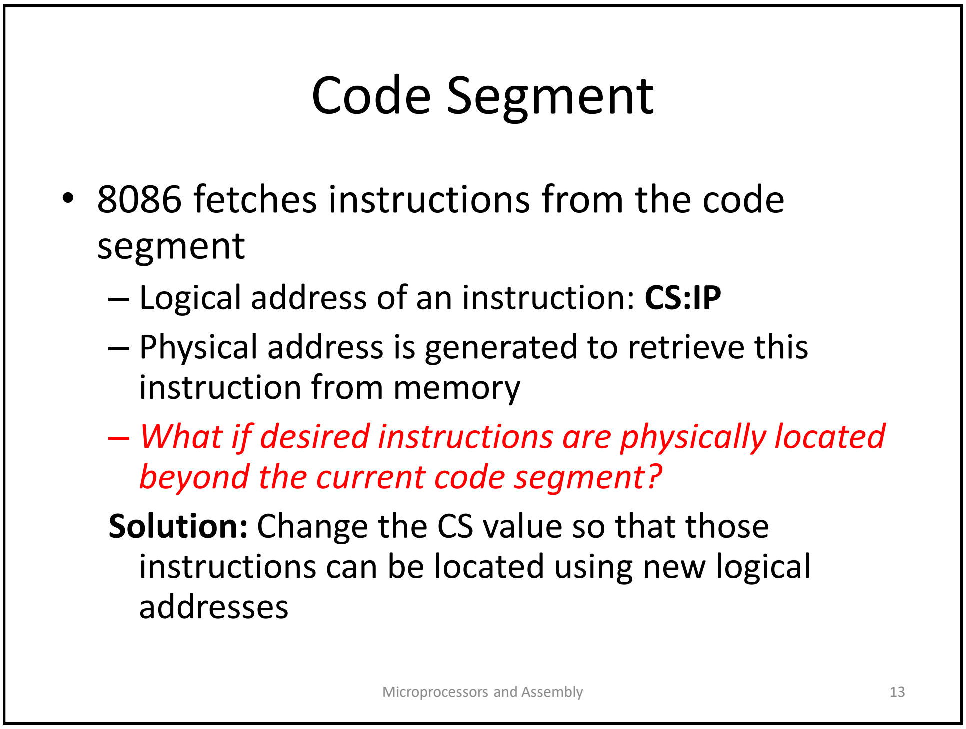
    The low range is FF590H, and the range goes to FFFFFH and wraps around from 00000H to 0F58FH (FF590+FFFF).



Logical & Physical Address

- Physical address -> logical address ?
  - One physical address can be derived from different logical addresses
  - E.g.,

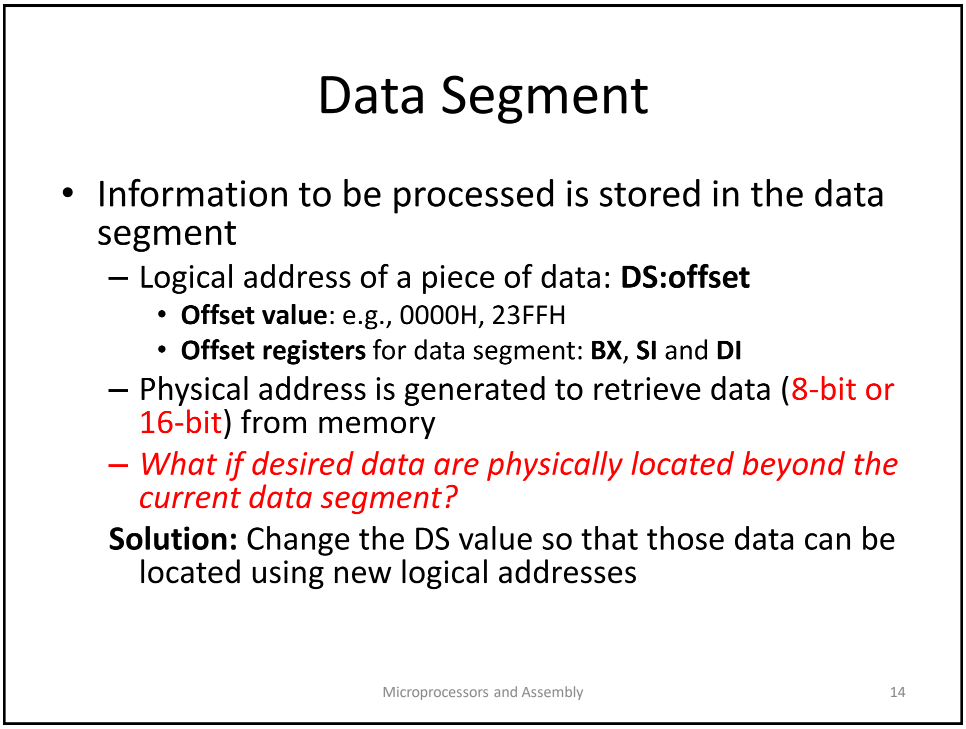
Logical address (hex)	Physical address (hex)
1000:5020	15020
1500:0020	15020
1502:0000	15020
1400:1020	15020
1302:2000	15020



Code Segment

- 8086 fetches instructions from the code segment
  - Logical address of an instruction: **CS:IP**
  - Physical address is generated to retrieve this instruction from memory
  - *What if desired instructions are physically located beyond the current code segment?*

Solution: Change the CS value so that those instructions can be located using new logical addresses



Data Segment

- Information to be processed is stored in the data segment
  - Logical address of a piece of data: **DS:offset**
    - **Offset value**: e.g., 0000H, 23FFH
    - **Offset registers** for data segment: **BX**, **SI** and **DI**
  - Physical address is generated to retrieve data (8-bit or 16-bit) from memory
  - *What if desired data are physically located beyond the current data segment?*

Solution: Change the DS value so that those data can be located using new logical addresses

Data Representation in Memory

- Memory can be logically imagined as a consecutive block of bytes
- *How to store data whose size is larger than a byte?*
  - Little endian: the low byte of the data goes to the low memory location
  - Big endian: the high byte of the data goes to the low memory location
  - E.g., 2738H

Stack Segment

- A section of RAM memory used by the CPU to store information temporarily
  - Logical address of a piece of data: **SS:SP** (special applications with **BP**)
  - Most registers (except segment registers and SP) inside the CPU can be stored in the stack and brought back into the CPU from the stack using ***push*** and ***pop***, respectively
  - Grows **downward** from upper addresses to lower addresses in the memory allocated for a program
    - *Why*? To protect other programs from destruction
    - *Note*: Ensure that the code section and stack section would not write over each other

Push & Pop

- 16-bit operation

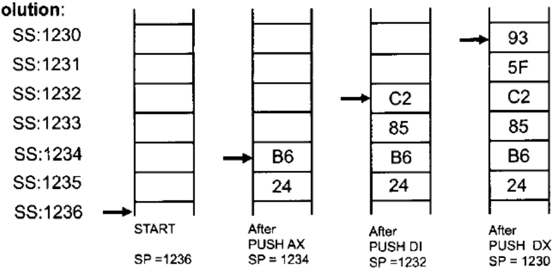
Example 1-6

Assuming that SP = 1236, AX = 24B6, DI = 85C2, and DX = 5F93, show the contents of the stack as each of the following instructions is executed:

```
PUSH  AX
PUSH  DI
PUSH  DX
```

Little endian or big endian?

Solution:



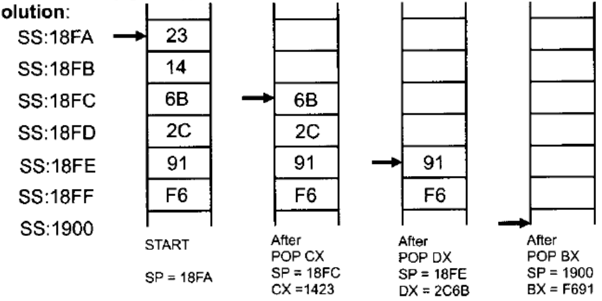
Push & Pop

Example 1-7

Assuming that the stack is as shown below, and SP = 18FA, show the contents of the stack and registers as each of the following instructions is executed:

```
POP    CX
POP    DX
POP    BX
```

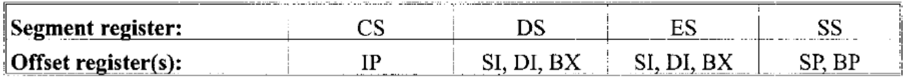
Solution:



Extra Segment

- An extra data segment, essential for string operations
  - Logical address of a piece of data: **ES:offset**
    - **Offset value**: e.g., 0000H, 23FFH
    - **Offset registers** for data segment: **BX**, **SI** and **DI**
- **In Summary,**

Table 1-3: Offset Registers for Various Segments

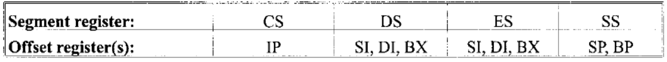


Segment register:	CS	DS	ES	SS
Offset register(s):	IP	SI, DI, BX	SI, DI, BX	SP, BP

Segment Overrides

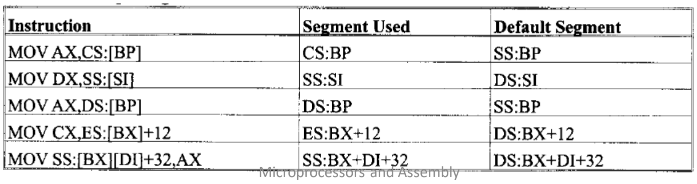
- Offset registers are used with default segment registers

Table 1-3: Offset Registers for Various Segments



Segment register:	CS	DS	ES	SS
Offset register(s):	IP	SI, DI, BX	SI, DI, BX	SP, BP

- 80x86 allows the program to override the default segment registers
  - Specify the segment register in the code

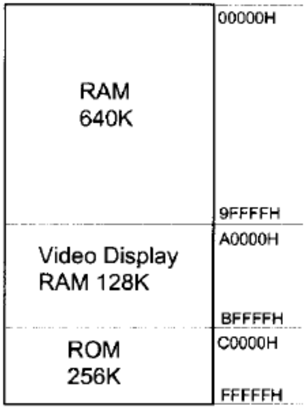


Instruction	Segment Used	Default Segment
MOV AX,CS:[BP]	CS:BP	SS:BP
MOV DX,SS:[SI]	SS:SI	DS:SI
MOV AX,DS:[BP]	DS:BP	SS:BP
MOV CX,ES:[BX]+12	ES:BX+12	DS:BX+12
MOV SS:[BX][DI]+32,AX	SS:BX+DI+32	DS:BX+DI+32

Memory Map of the IBM PC

- 1MB logical address space
- 640K max RAM
  - In 1980s, 64kB-256KB
  - MS-DOS, application software
  - DOS does memory management; you do not set CS, DS and SS
- Video display RAM
- ROM
  - 64KB BIOS
  - Various adapter cards

What about IO device addresses?



BIOS Function

- Basic input-output system (BIOS)
  - Tests all devices connected to the PC when powered on and reports errors if any
  - Load DOS from disk into RAM
  - Hand over control of the PC to DOS

- Recall that after CPU being reset, what is the first instruction that CPU will execute?

ADDRESSING MODES

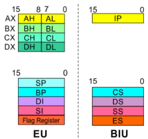
Addressing Modes

- Every instruction of a program has to operate on a data.
- The different ways in which a source operand is denoted in an instruction are known as addressing modes.

1. Register Addressing	Group I : Addressing modes for register and immediate data
2. Immediate Addressing	
3. Direct Addressing	Group II : Addressing modes for memory data
4. Register Indirect Addressing	
5. Based Addressing	
6. Indexed Addressing	
7. Based Index Addressing	
8. String Addressing	
9. Direct I/O port Addressing	Group III : Addressing modes for I/O ports
10. Indirect I/O port Addressing	
11. Relative Addressing	Group IV : Relative Addressing mode
12. Implied Addressing	Group V : Implied Addressing mode

Addressing Modes

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



Data are held within registers
No need to access memory

Example:

MOV BX, DX

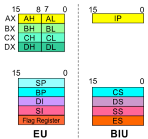
Copy the contents of DX into BX

MOV ES, AX

Copy the contents of AX into ES

Addressing Modes

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



In immediate addressing mode, an 8-bit or 16-bit data is specified as part of the instruction

Example:

MOV DL, 08H

The 8-bit data (08_H) given in the instruction is moved to DL

(DL) ← 08_H

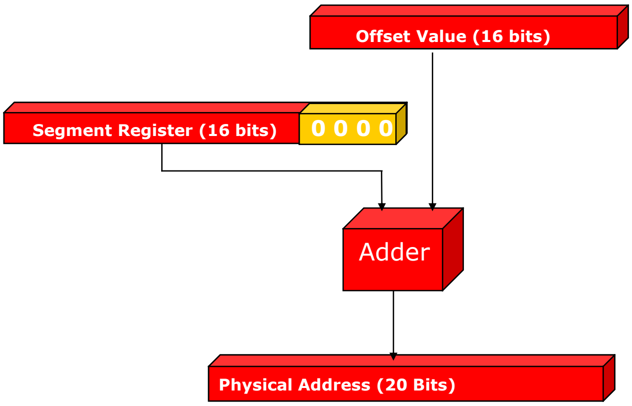
MOV AX, 0A9FH

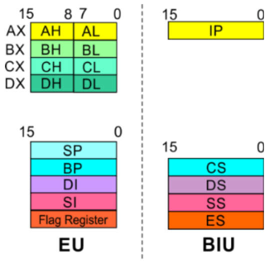
The 16-bit data ($0A9F_H$) given in the instruction is moved to AX register

(AX) ← $0A9F_H$

Immediate numbers CANNOT be moved to segment registers

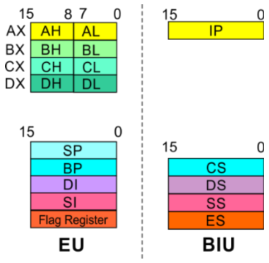
Addressing Modes: Memory Access





Addressing Modes: Memory Access

- **20 Address lines ⇒ 8086 can address up to 2^{20} = 1M bytes of memory**
- **However, the largest register is only 16 bits**
- **Physical Address will have to be calculated**
  Physical Address : Actual address of a byte in memory. i.e. the value which goes out onto the address bus.
- **Memory Address represented in the form – Seg : Offset (Eg - 89AB:F012)**
- **Each time the processor wants to access memory, it takes the contents of a segment register, shifts it one hexadecimal place to the left (same as multiplying by 16_{10}), then add the required offset to form the 20- bit address**



16 bytes of contiguous memory

89AB : F012 → 89AB → 89AB0 (Paragraph to byte → 89AB x 10 = 89AB0)

F012 → 0F012 (Offset is already in byte unit)

+ -------

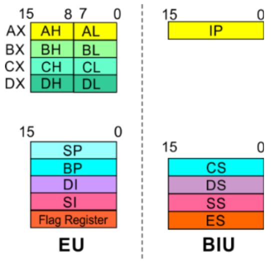
98AC2 (The absolute address)

Addressing Modes: Memory Access

- **To access memory we use these four registers: BX, SI, DI, BP**
- **Combining these registers inside [ ] symbols, we can get different memory locations (Effective Address, EA)**
- **Supported combinations:**

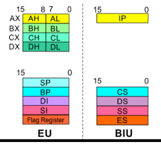
[BX + SI] [BX + DI] [BP + SI] [BP + DI]	[SI] [DI] d16 (variable offset only) [BX]	[BX + SI + d8] [BX + DI + d8] [BP + SI + d8] [BP + DI + d8]
[SI + d8] [DI + d8] [BP + d8] [BX + d8]	[BX + SI + d16] [BX + DI + d16] [BP + SI + d16] [BP + DI + d16]	[SI + d16] [DI + d16] [BP + d16] [BX + d16]

BX	**SI**	**+ disp**
BP	**DI**	



Addressing Modes

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



Here, the effective address of the memory location at which the data operand is stored is given in the instruction.

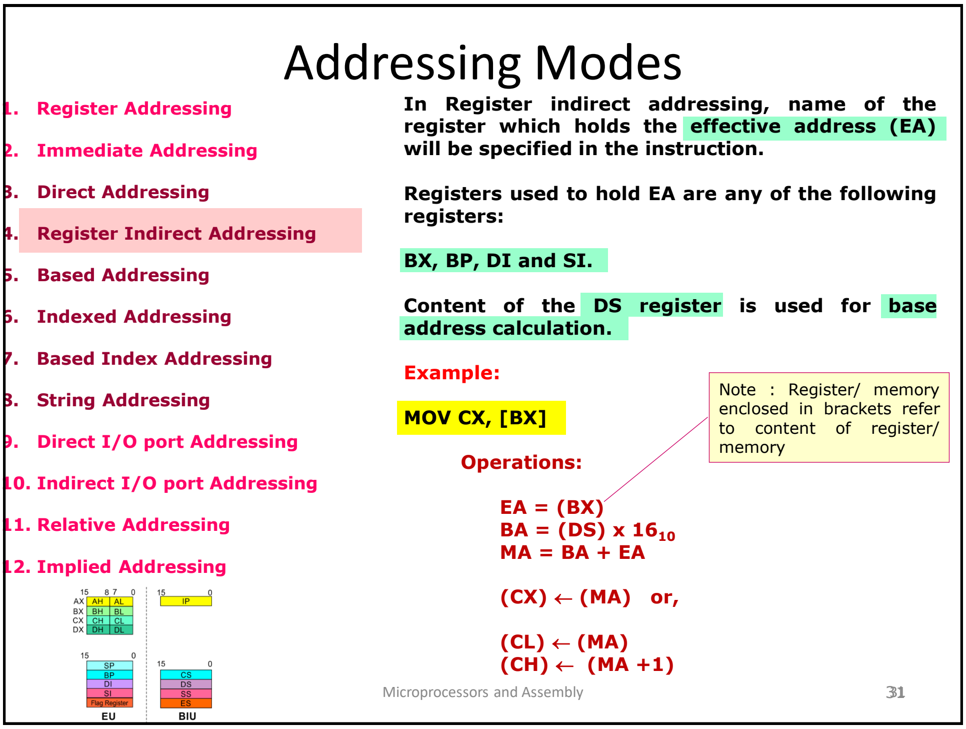
The effective address is just a 16-bit number written directly in the instruction.

Example:

```
MOV   BX, [1354H]
MOV   BL, [0400H]
```

The square brackets around the 1354_H denotes the contents of the memory location. When executed, this instruction will copy the contents of the memory location DS:1354_H into BX register.

This addressing mode is called direct because the displacement of the operand from the segment base is specified directly in the instruction.



Addressing Modes

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

15
8 7
0
AX
AH
AL
BX
BH
BL
CX
CH
CL
DX
DH
DL
15
0
SP
BP
DI
SI
Flag Register
EU
15
0
IP
15
0
CS
DS
SS
ES
BIU

In Register indirect addressing, name of the register which holds the effective address (EA) will be specified in the instruction.

Registers used to hold EA are any of the following registers:

BX, BP, DI and SI.

Content of the DS register is used for base address calculation.

Example:

MOV CX, [BX]

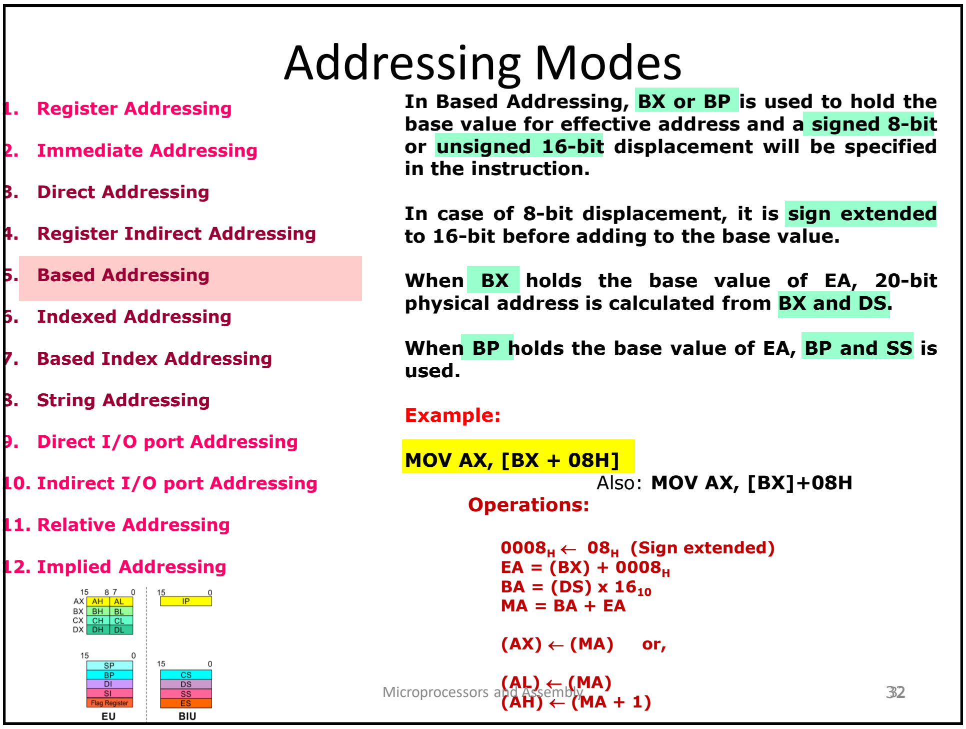
Note : Register/ memory enclosed in brackets refer to content of register/ memory

Operations:

EA = (BX)
BA = (DS) x 16_{10}
MA = BA + EA

(CX) ← (MA) or,

(CL) ← (MA)
(CH) ← (MA +1)



Addressing Modes

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

15
8 7
0
AX
AH
AL
BX
BH
BL
CX
CH
CL
DX
DH
DL
15
0
SP
BP
DI
SI
Flag Register
EU
15
0
IP
15
0
CS
DS
SS
ES
BIU

In Based Addressing, BX or BP is used to hold the base value for effective address and a signed 8-bit or unsigned 16-bit displacement will be specified in the instruction.

In case of 8-bit displacement, it is sign extended to 16-bit before adding to the base value.

When BX holds the base value of EA, 20-bit physical address is calculated from BX and DS.

When BP holds the base value of EA, BP and SS is used.

Example:

MOV AX, [BX + 08H]

Also: **MOV AX, [BX]+08H**

Operations:

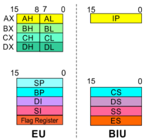
0008_H ← 08_H (Sign extended)
EA = (BX) + 0008_H
BA = (DS) x 16_{10}
MA = BA + EA

(AX) ← (MA) or,

(AL) ← (MA)
(AH) ← (MA + 1)

Addressing Modes

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



SI or DI register is used to hold an index value for memory data and a signed 8-bit or unsigned 16-bit displacement will be specified in the instruction.

Displacement is added to the index value in SI or DI register to obtain the EA.

In case of 8-bit displacement, it is sign extended to 16-bit before adding to the base value.

Example:

MOV CX, [SI + 0A2H]

Also: **MOV CX, [SI]+0A2H**

Operations:

$FFA2_H \leftarrow A2_H$ **(Sign extended)**

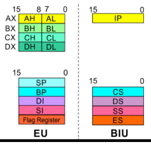
$EA = (SI) + FFA2_H$
$BA = (DS) \times 16_{10}$
$MA = BA + EA$

$(CX) \leftarrow (MA)$ **or,**

$(CL) \leftarrow (MA)$
$(CH) \leftarrow (MA + 1)$

Addressing Modes

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



In Based Index Addressing, the effective address is computed from the sum of a base register (BX or BP), an index register (SI or DI) and a displacement.

Example:

MOV DX, [BX + SI + 0AH]

Also: **MOV DX, [BX][SI]+0AH**

Operations:

$000A_H \leftarrow 0A_H$ **(Sign extended)**

$EA = (BX) + (SI) + 000A_H$
$BA = (DS) \times 16_{10}$
$MA = BA + EA$

$(DX) \leftarrow (MA)$ **or,**

$(DL) \leftarrow (MA)$
$(DH) \leftarrow (MA + 1)$

Addressing Modes

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

Employed in string operations to operate on string data.

The effective address (EA) of source data is stored in SI register and the EA of destination is stored in DI register.

Segment register for calculating base address of source data is DS and that of the destination data is ES

Example: MOVS BYTE

Operations:

Calculation of source memory location:
EA = (SI)   BA = (DS) x 16_{10} MA = BA + EA

Calculation of destination memory location:
EA_E = (DI) BA_E = (ES) x 16_{10} MA_E = BA_E + EA_E

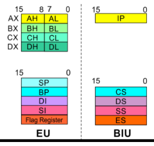
Note : Effective address of the Extra segment register

(MAE) ← (MA)

If DF = 1, then (SI) ← (SI) – 1 and (DI) = (DI) - 1
If DF = 0, then (SI) ← (SI) +1 and (DI) = (DI) + 1

Addressing Modes

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



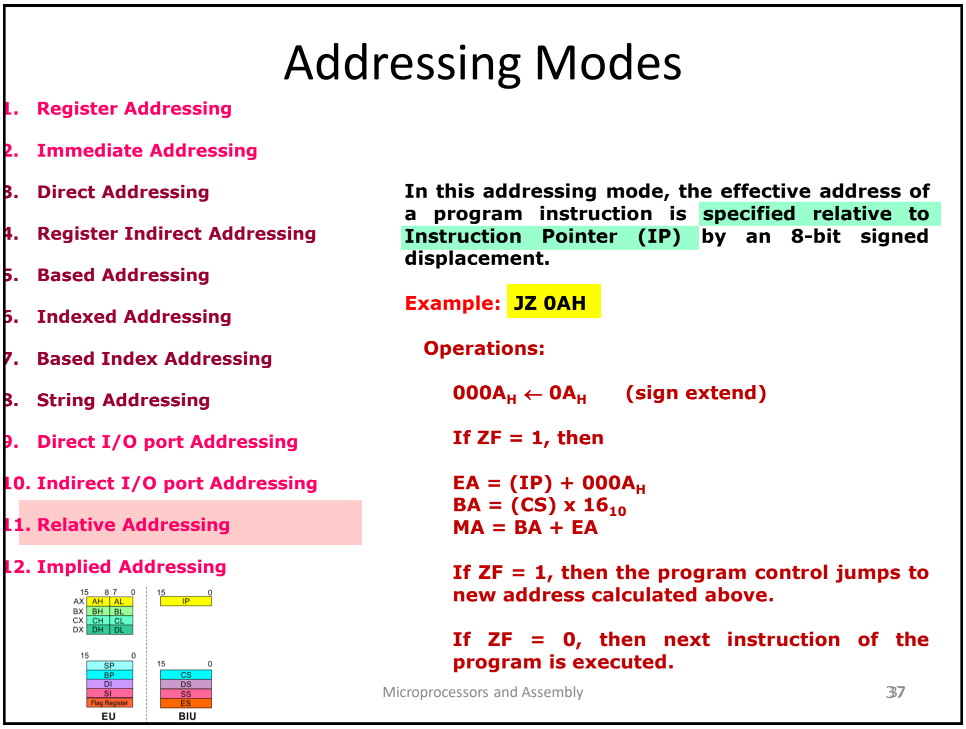
These addressing modes are used to access data from standard I/O mapped devices or ports.

In direct port addressing mode, an 8-bit port address is directly specified in the instruction.

Example: IN AL, [09H]

Operations: $PORT_{addr}$ = 09_H
(AL) ← (PORT)

Content of port with address 09_H is moved to AL register



Addressing Modes

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

15 8 7 0
AX AH AL
BX BH BL
CX CH CL
DX DH DL
15 0
SP
BP
DI
SI
Flag Register
EU
15 0
IP
15 0
CS
DS
SS
ES
BIU

In this addressing mode, the effective address of a program instruction is specified relative to Instruction Pointer (IP) by an 8-bit signed displacement.

Example: JZ 0AH

Operations:

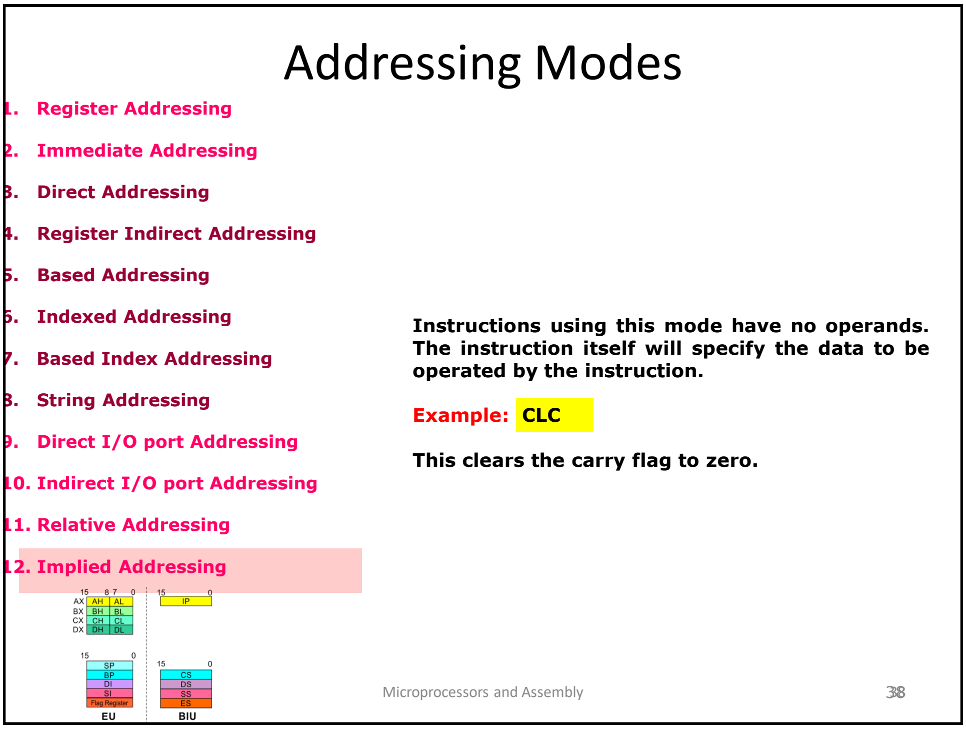
$000A_H \leftarrow 0A_H$ (sign extend)

If ZF = 1, then

$EA = (IP) + 000A_H$
$BA = (CS) \times 16_{10}$
$MA = BA + EA$

If ZF = 1, then the program control jumps to new address calculated above.

If ZF = 0, then next instruction of the program is executed.



Addressing Modes

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

15 8 7 0
AX AH AL
BX BH BL
CX CH CL
DX DH DL
15 0
SP
BP
DI
SI
Flag Register
EU
15 0
IP
15 0
CS
DS
SS
ES
BIU

Instructions using this mode have no operands. The instruction itself will specify the data to be operated by the instruction.

Example: CLC

This clears the carry flag to zero.

Next Lecture

- Assembly programming