8086 Microprocessor

Features, Architecture, Addressing modes

Introduction

Intel's first **16-bit** microprocessor - **8086**

8086 can address **one megabytes** (1MB=2^20 bytes) of memory (as it has 20 address lines).

- Another feature in 8086 - presence of a small 6-byte instruction queue - so instructions fetched from memory are placed in it, before they are executed.
- higher execution speed - larger memory size

Architecture of 8086

- It is subdivided into two units - The execution unit (**EU**) and The bus interface unit (**BIU**)
- The execution unit (EU) includes : ALU - eight **16-bit** general purpose registers - a 16 bit flag register - a control unit.
- The bus interface unit (BIU) includes : **adder** for address calculations - four 16-bit **segment registers** (CS, DS, SS and ES) - a 16 bit **instruction pointer** (IP) - a 6 byte instruction queue and bus control logic.

8086 ARCHITECTURE

- The BIU **fetches** instructions, reads and writes data, and computes the 20-bit address.

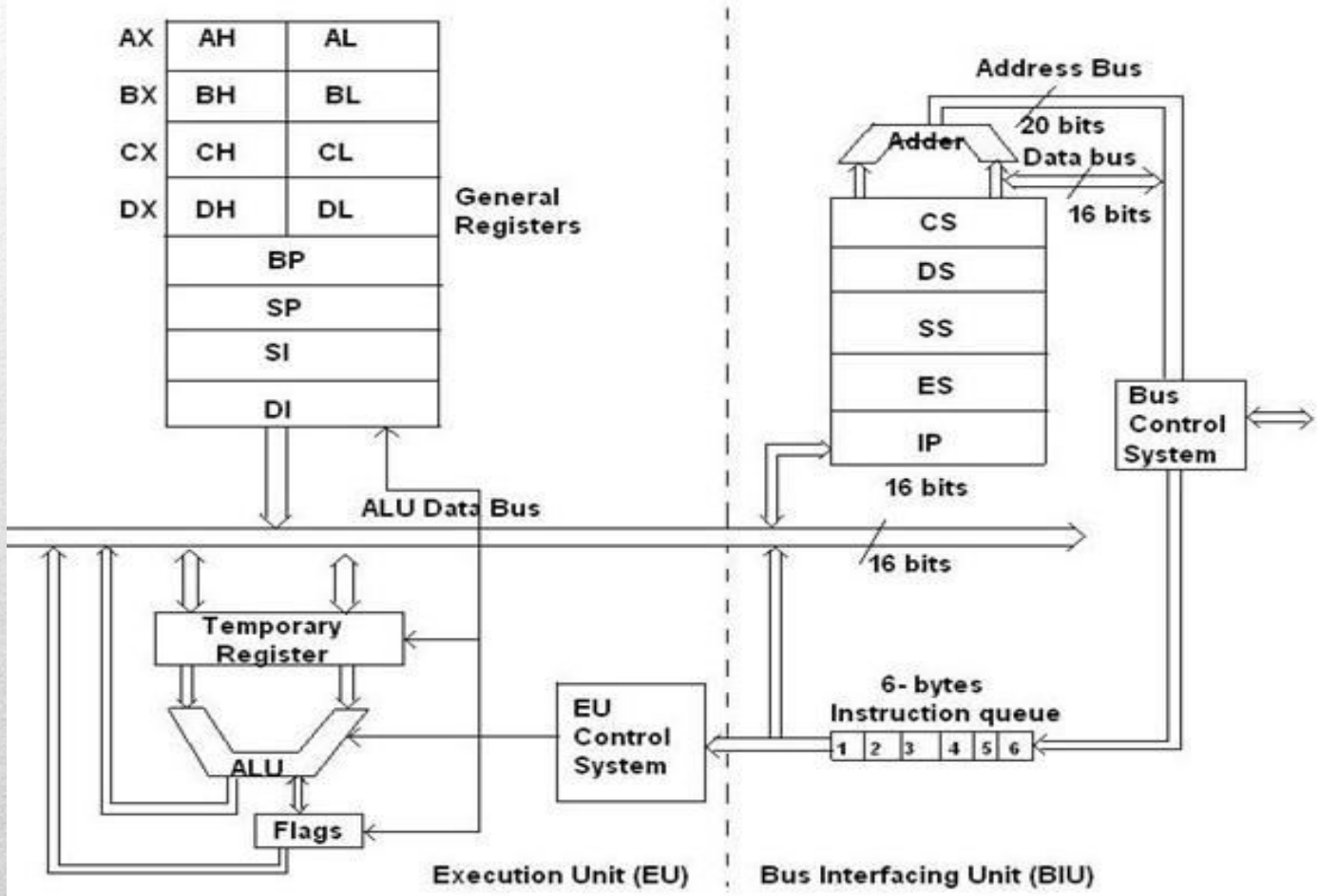
- The EU **decodes** and **executes** the instructions using the 16-bit ALU.

The BIU contains the following registers:

IP - Instruction Pointer
CS - Code Segment Register
DS - Data Segment Register
SS - Stack Segment Register
ES - Extra Segment Register

The BIU fetches **instructions using the CS** and **IP**, written CS:IP, to construct the 20-bit address.

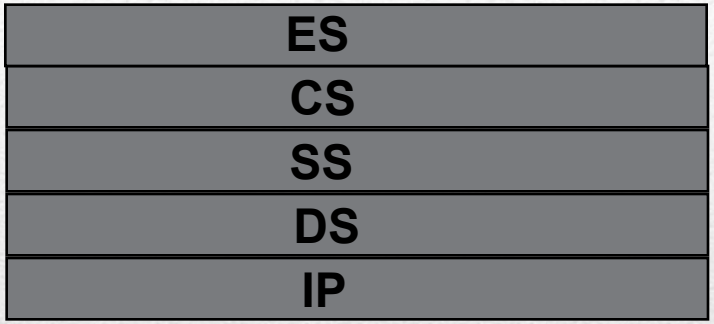
Functional Block diagram of 8086



Execution Unit (EU)

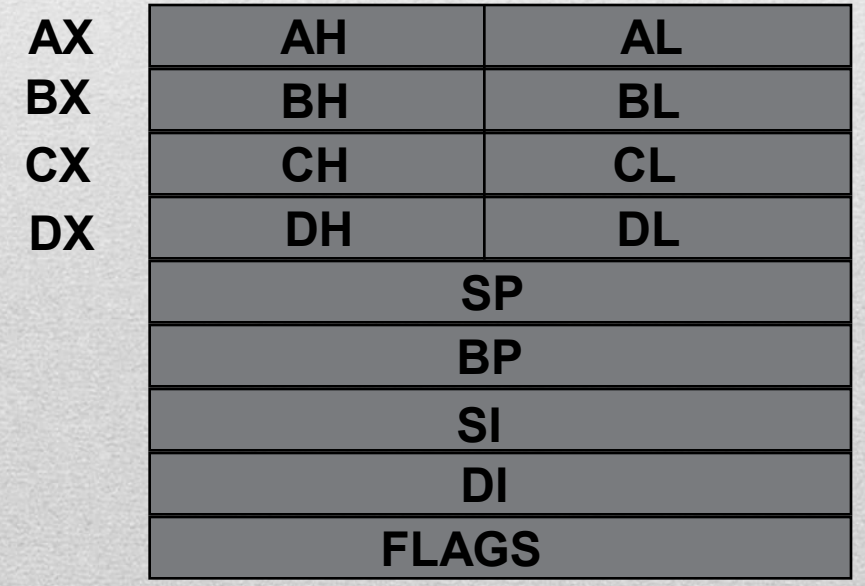
- The EU consists of **eight** 16-bit general purpose registers - AX, BX, CX, DX, SP, BP, SI and DI.
- AX, BX, CX and DX - can be divided into two 8-bit registers - AH, AL, BL, BH, CH, CL, DH and DL
- General purpose registers - can be used to store 8 bit or 16 bit data during program execution.

BIU registers
(20 bit adder)



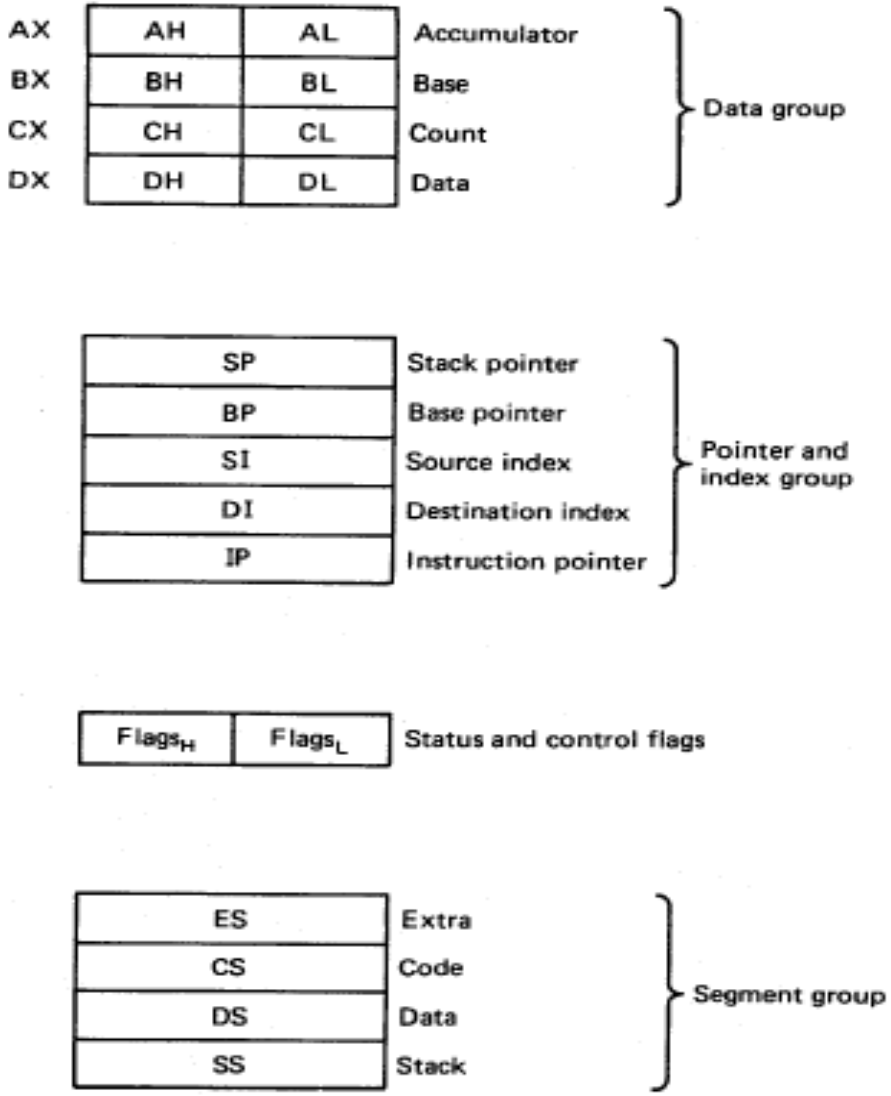
Extra Segment
Code Segment
Stack Segment
Data Segment
Instruction Pointer

EU registers



Accumulator
Base Register
Count Register
Data Register
Stack Pointer
Base Pointer
Source Index Register
Destination Index Register

8086 internal registers (16 bits size)



AX, BX, CX and DX are two bytes wide and each byte can be accessed separately

These registers are used as memory *pointers*.

Status and control Flags

Segment registers are used as base address for a segment in the 1 M byte of memory

Data Registers: AX, BX, CX, DX

- Instructions execute faster if the data is in a register
- AX, BX, CX, DX are the data registers
- Low and High bytes of the data registers can be accessed separately
  - AH, BH, CH, DH are the high bytes
  - AL, BL, CL, and DL are the low bytes
- Data Registers are general purpose registers but they also perform special functions
- AX
  - Accumulator Register
  - Preferred register to use in arithmetic, logic and data transfer instructions because it generates the shortest Machine Language Code
  - **Must be used in multiplication and division operations**
  - **Must also be used in I/O operations**

- BX
  - Base Register
  - Also serves as an address register
  - Used in array operations
  - Used in Table Lookup operations (XLAT)
- CX
  - Count register
  - Used as a loop counter
  - Used in shift and rotate operations
- DX
  - Data register
  - Used in multiplication and division
  - Also used in I/O operations

Special functions of registers

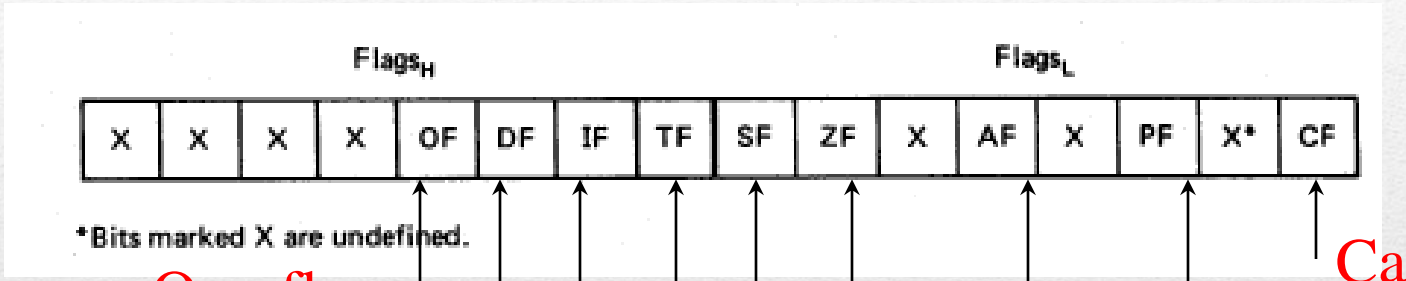
- **AX/AL** - used as **accumulator** - multiply, divide, input/output (I/O) and some of the decimal and ASCII adjustment instructions.
- **BX** - holds the **base** (offset) address of a location in memory - also used to refer the data in memory using lookup table technique with the help of XLAT instruction.
- **CX** - used to hold the **count** while executing repeated string instructions (REP/REPE/REPNE) and LOOP instruction - also used to hold the count while executing the shift and rotate instructions - count value indicates the number of times the same instructions has to be executed.
- **DX** - used to hold a part of the result during multiplication and part of the dividend before a division - also used to hold the I/O device address while executing IN and OUT instructions
- **SP - stack pointer** - used to hold the offset address of the data stored at the **top of stack segment** - used along with SS register to decide the address at which data is pushed or popped during the execution of PUSH and POP instructions.

Special functions of registers

- **BP - Base Pointer** - used to hold the offset address of the data to be read from or write into the **stack segment**
- **SI - Source Index register** - used to hold the offset address of source data in data segment while executing **string** instructions
- **DI - Destination Index register** - used to hold the offset address of destination data in extra segment while executing String instructions.
- NOTE : segment - a portion of memory where data for a program is stored - the maximum size of a segment can be 64 bytes - minimum size of a segment can be even one byte - **segment begins in memory at a memory address which is divisible by 16(i.e. 10H).**

Flag register of 8086

- The flags - classified into status flags and control flags
- CF, PF, AF, ZF, SF and OF - status flags - they indicate the status of the result that is obtained after the execution of arithmetic or logic instruction
- DF, IF and TF - control flags - they can control the operation of CPU



Overflow

Direction

Interrupt enable

Trap

Sign

Zero

Auxiliary flag

Parity flag

Carry flag

6 are status flags

3 are control flag

Function of different flags

- CF (Carry Flag) - holds the carry after 8 bit or 16 bit addition - holds borrow after 8 bit or 16 bit subtraction performed
- PF (Parity Flag) - If the lower 8 bit of the result is having odd parity (i.e. odd number of 1s) - PF is set to 0 - PF is set to 1 if the lower 8 bit of result is having even parity.
- AF (Auxiliary Carry Flag) - holds the carry after addition - the borrow after subtraction of the bits in bit position 3 - (LSB is treated as bit position 0) - used by DAA and DAS instructions to adjust the value in AL after a BCD addition or subtraction.
- ZF (Zero Flag) - indicates that the result of an arithmetic or logic operation is zero - If Z=1, the result is zero - if Z=0, the result is not zero.
- SF (Sign flag) - holds the arithmetic sign of the result after an arithmetic or logic instruction is executed - If S-0, the sign bit is 0 and the result is negative.

Function of different flags

- TF (Trap Flag) - used to debug a program using single step technique - If T flag is set (i.e. TF=1) - 8086 gets interrupted (Trap or single step interrupt) after the execution of each instruction in the program - If TF is cleared (i.e. TF=0) - the trapping or debugging feature is disabled
- DF (Direction Flag) - selects either the increment or decrement made for the DI and/or SI register during the execution of string instructions - If D=0 - registers are automatically incremented. If D=1 - the register are automatically decremented - can set or cleared using the STD or CLD instruction respectively.

Function of different flags

- IF (Interrupt Flag) - controls the operation of the ‘INTR’ interrupt pin of 8086 - If I=0 - INTR pin is disabled - if I=1 - INTR pin is enabled - I flag can be set or cleared using the instruction STI or CLI respectively.
- OF (Overflow flag) - Signed numbers are represented in 2’s complement form when the number is negative in microprocessor - When signed numbers are added or subtracted - overflow may occur - indicating that the result has exceeded the capacity of the machine - For example if the 8-bit signed data 7EH (= +126) is added with the 8-bit signed data 02H(= +2), the result is 80H(= -128 in 2’s complement form). This result indicates an overflow condition - overflow flag is set during the above signed addition - In an 8-bit register, the minimum and maximum value of the signed number that can be stored in -128 (=80H) and +127 (=7FH) respectively - In a 16 bit register, the minimum and maximum value of the signed number that can be stored is -32768 (=8000H) and +32767 (=7FFFH) respectively - For operation an unsigned data, OF is ignored.

Bus Interface Unit (BIU)

- There are **four segment registers - CS, DS, SS and ES**.
- The function of the CS, DS, SS and ES register - **indicate the starting address or base address of code segment, data segment, stack segment and extra segment in memory**
- The **code segment** - contains the instructions of a program
- **Data segment** - contains data for the program
- **Stack segment** - holds the stack of a program - which is needed while executing CALL and RET instructions - also to handle interrupts
- **Extra segment** - additional data segment - used by some of the string instructions
- - minimum size of a segment - one byte - maximum size of a segment is 64 Kbytes.
- The base address - can be obtained by adding four binary 0s to the right most portion of the content of corresponding segment register which is same as adding a hexadecimal digit 0 to the right most portion of a segment register..

Segmented Memory Architecture
(real mode)

- A segment addresses 64K of memory
- A segment register contains the starting location of a segment
  - the absolute location of a segment can be obtained by appending a hexadecimal zero
- An offset is the distance from the beginning of a segment to a particular instruction or variable

Accessing memory locations

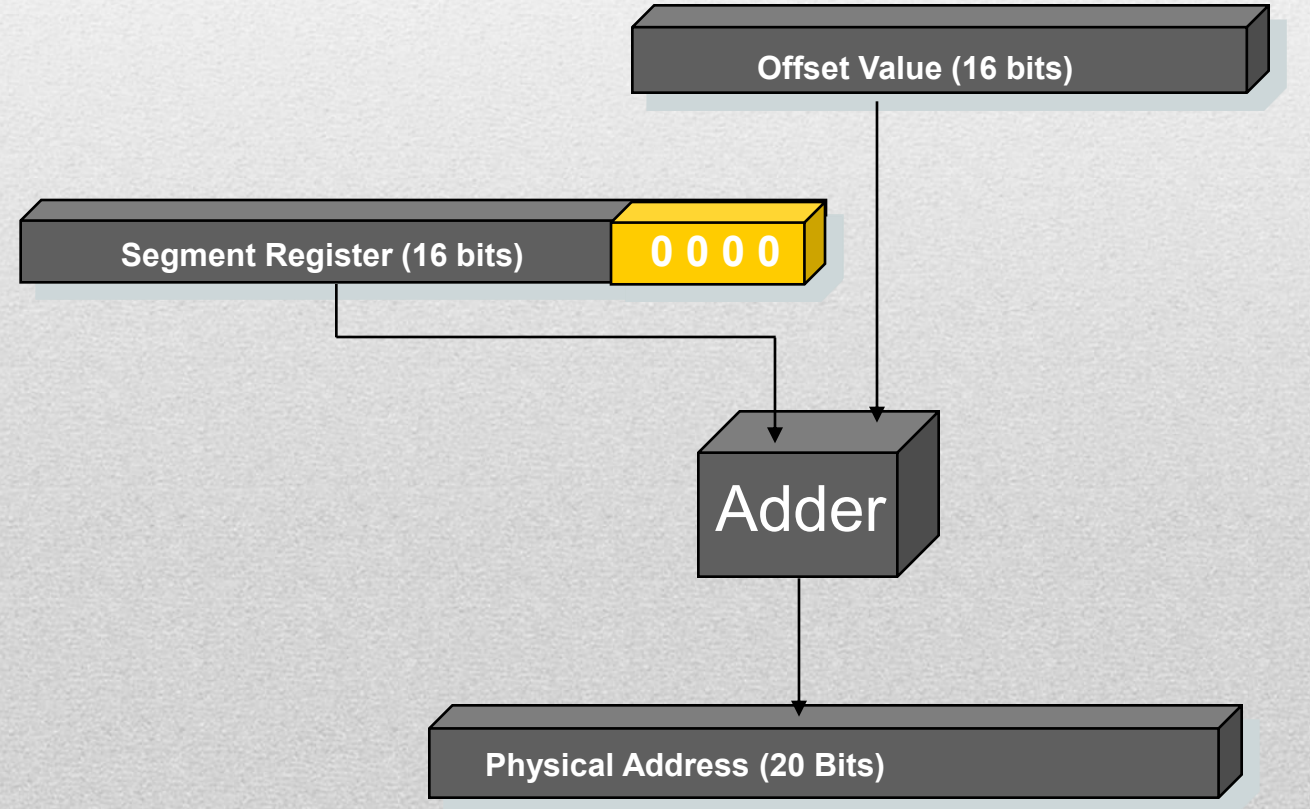
- Each address in physical memory is known as physical address. In order to access an operand (either data or instruction) from memory from a particular segment, 8086 has to first calculate the **physical address** of that operand.
- To find the physical address of that operand, the 8086 **adds the base address of the corresponding segment with an offset address** which may be either the content of a register or an 8 bit or 16 bit **displacement** given in the instruction or combination of both, depending upon the addressing mode used by the instruction.
- The 8086 designers have assigned certain register(s) as default offset register(s) for certain segment register.
- But this default assignment can be changed by the programmer as per the requirement.

Fetching of an instruction from memory

- Let us assume that the **CS** register is having the value 3000H and the **IP** register is having the value 2000H.
- To fetch an instruction from memory, the CPU calculates the memory address from where the next instruction is to be fetched, as shown below:
- **CS X 10H = 30000H (Base address of code segment)**
- **IP = 2000H (Offset address)**
- 32000H Memory address from where next instruction is taken by CPU

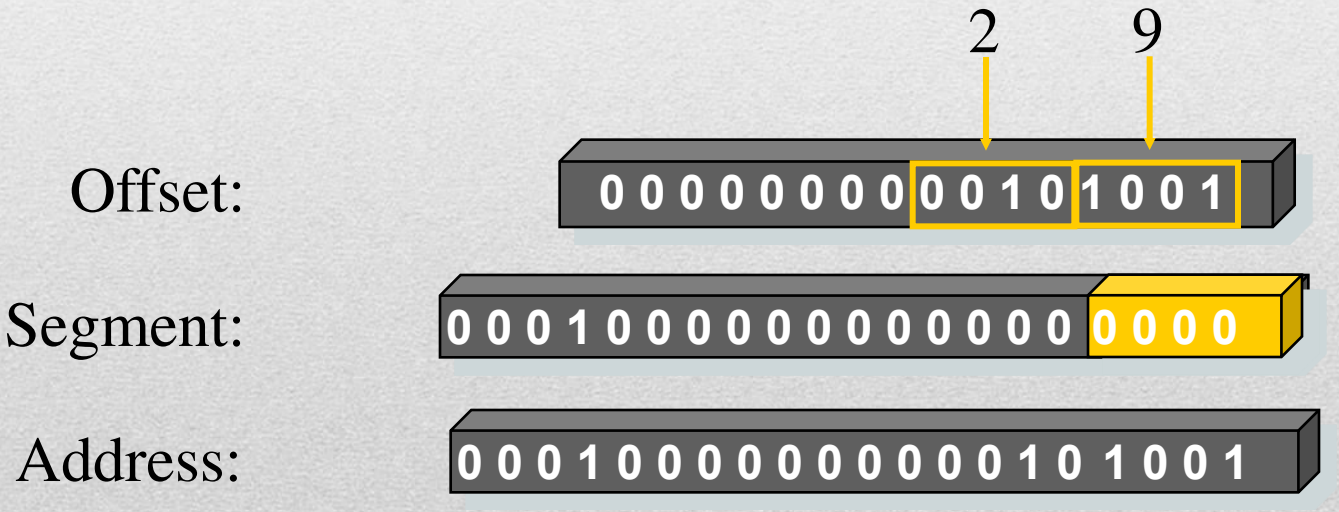
Memory Address Generation

- The BIU has a dedicated adder for determining physical memory addresses

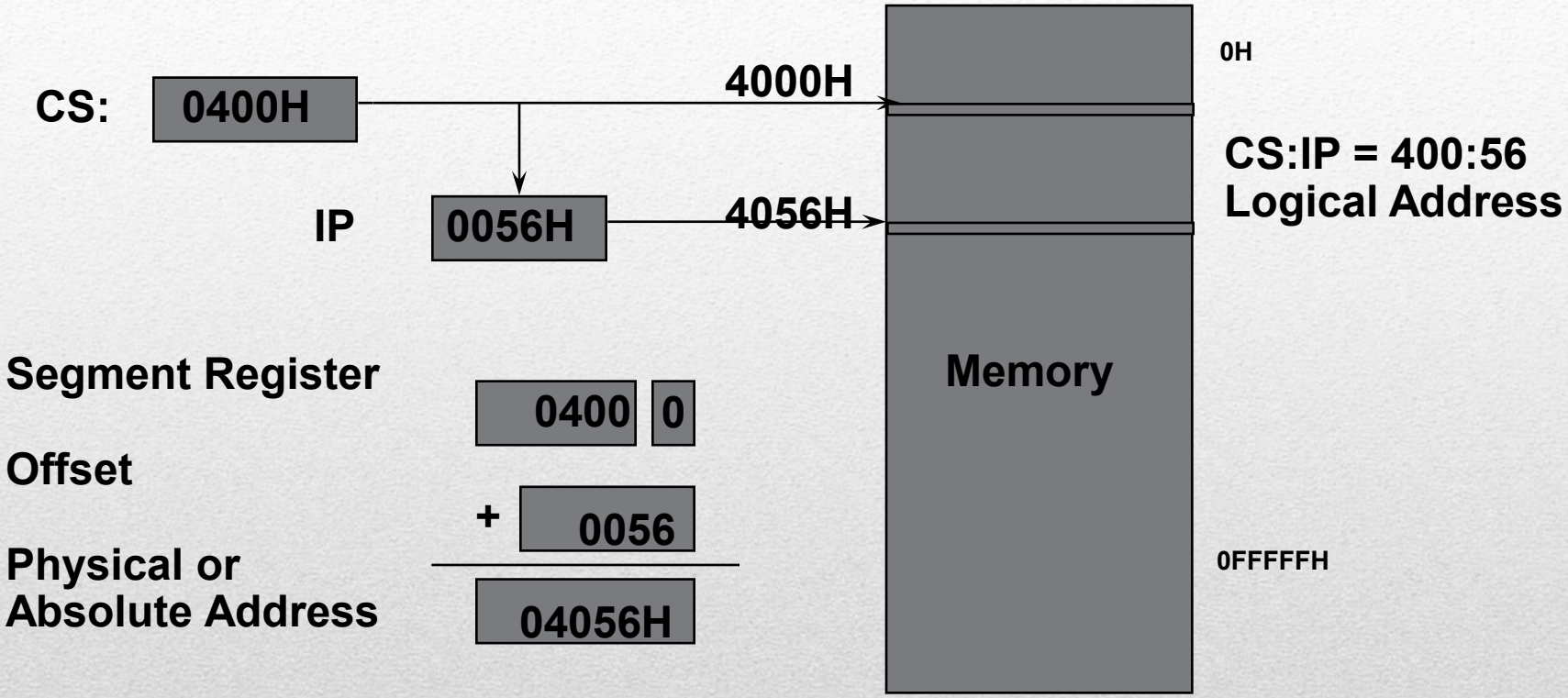


Example Address Calculation

- If the data segment starts at location 1000h and a data reference contains the address 29h where is the actual data?



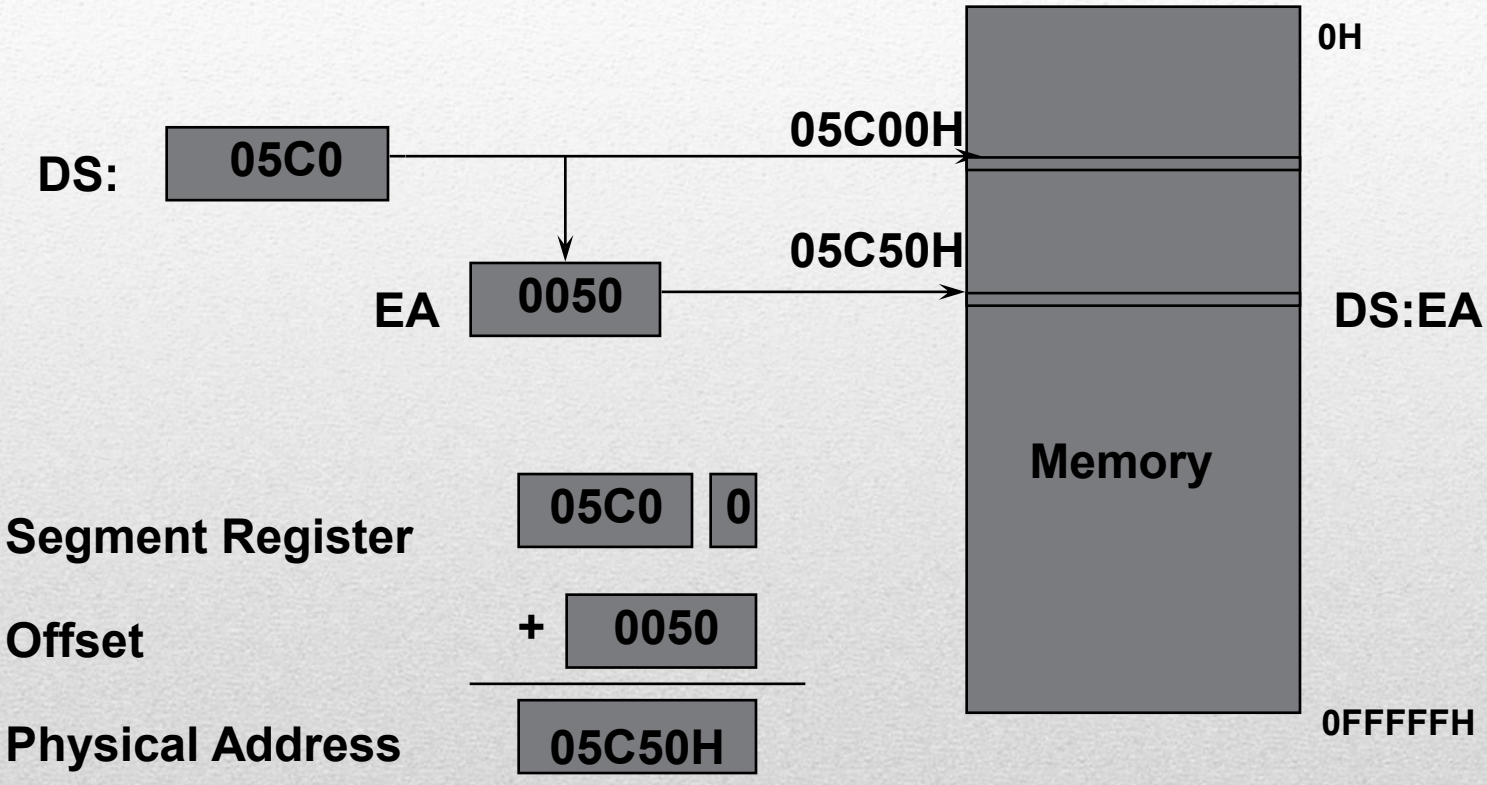
THE CODE SEGMENT



The offset is the distance in bytes from the start of the segment.
The offset is given by the IP for the Code Segment.
Instructions are always fetched with using the CS register.

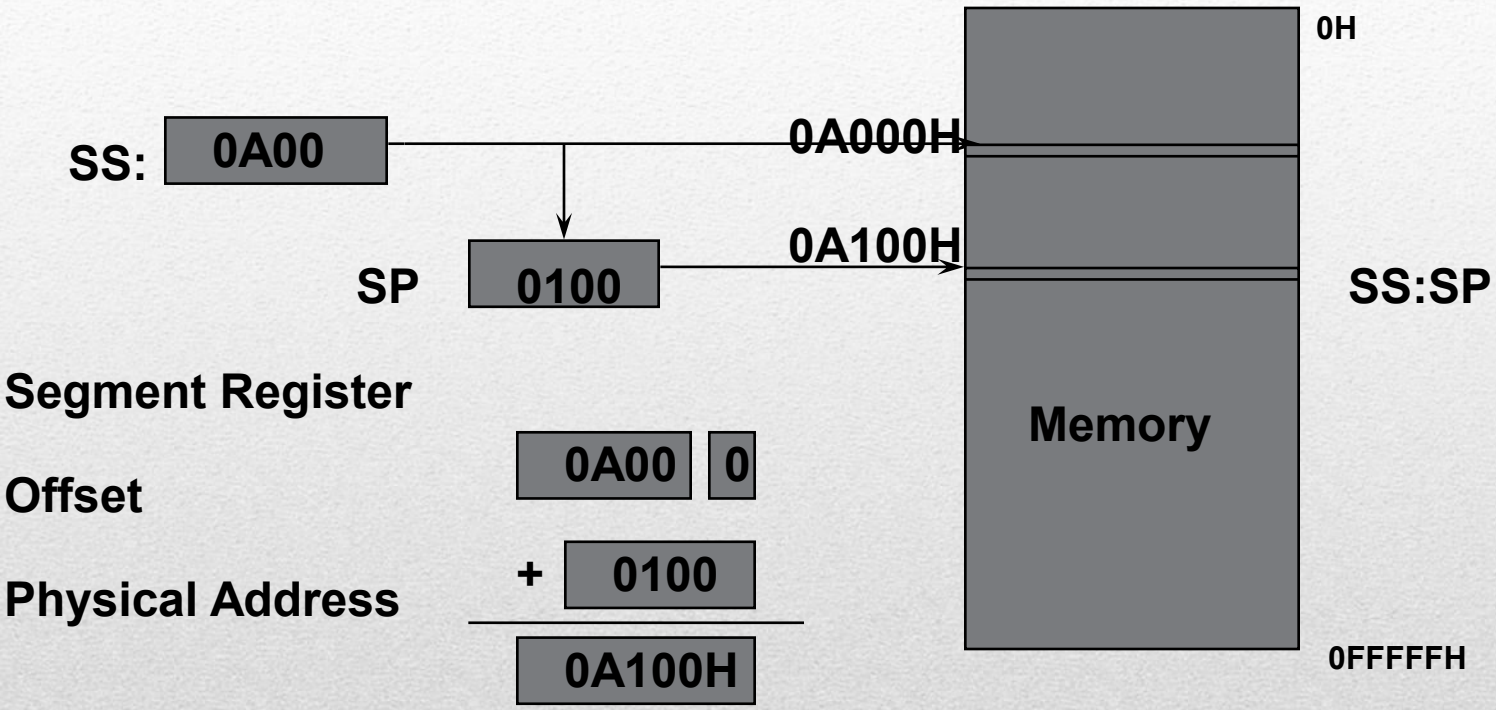
The physical address is also called the absolute address.

THE DATA SEGMENT



Data is usually fetched with respect to the DS register.
The effective address (EA) is the offset.
The EA depends on the addressing mode.

THE STACK SEGMENT



The offset is given by the SP register.
The stack is always referenced with respect to the stack segment register.
The stack grows toward decreasing memory locations.
The SP points to the last or top item on the stack.

PUSH - pre-decrement the SP
POP - post-increment the SP

Addressing modes in 8086

- Register addressing mode
- Immediate addressing mode
- Data memory addressing modes
- Program memory addressing modes
- Stack memory addressing modes

Addressing modes in 8086

- Register addressing mode
- Immediate addressing mode
- Data memory addressing modes
- Direct addressing mode
- Base addressing mode
- Base relative addressing mode
- Index addressing mode
- Index relative addressing mode
- Based index addressing mode
- Based relative plus index addressing mode
- Program memory addressing modes
- Direct addressing mode
- Relative addressing mode
- Indirect addressing mode
- Stack memory addressing modes

Register addressing mode

- In this addressing mode, the data present in register is moved or manipulated and the result is stored in register.
- Examples:
- **MOV AL,BL** ; Move content of BL to AL
- **MOV CX,BX** ; Move content of BX to CX
- **ADD CL,BL** ; Add content of CL and BL and store result in CL
- **ADC BX,DX** ; Add content of BX, carry flag and DX, and store result in BX

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

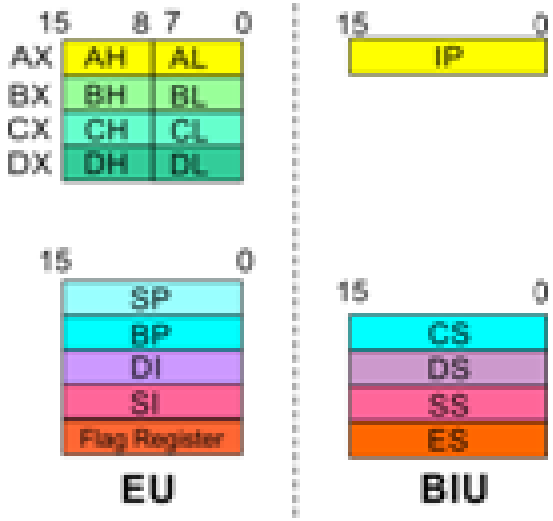
The instruction will specify the name of the register which holds the data to be operated by the instruction.

Example:

MOV CL, DH

The content of 8-bit register DH is moved to another 8-bit register CL

(CL) ← (DH)

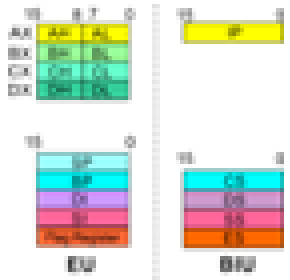


Immediate addressing mode

- In this mode, the data is directly given in the instruction.
- Examples
- **MOV AL,50H** ; Move data 50H to AL
- **MOV BX,23A0H** ; Move data 23A0H to BX
- MOV [SI],43C0H ; Move data 43C0H to memory at [SI]
- In the last example, [SI] represents the memory location in data segment at the offset address specified by SI 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



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) ← $0A9F_H$

Data memory addressing modes

- The term Effective Address (EA) represents the offset address of the data within a segment which is obtained by different methods, depending upon the addressing mode that is used in the instruction.
- Let us assume that the various registers in 8086 have the following values stored in them.

- DS = 3000H, CS= ,ES= SS= 4000H ,AX=
- BX=2000H CX= DX= BP=1000 SI=1000 DI = 3000

Direct Addressing:

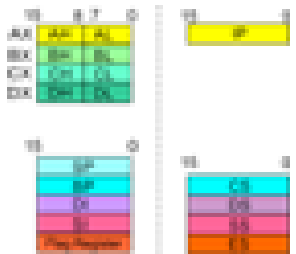
- In this mode, the 16 bit offset address of the data within the segment is directly given in the instruction.
- Examples:
- a) **MOV AL, [1000H]** **;** DS = 3000H
- EA is given within square bracket in the instruction in this addressing mode and hence EA=1000H in the above instruction. Since the destination is an 8-bit register (i.e. AL), a **byte** is taken from memory at the address given by DS x10H + EA=31000H and stored in AL.
- b) **MOV BX,[2000H]**
- EA=2000H in this instruction. Since the destination is a 16 bit register (i.e. BX), a **word** is taken from memory address DSx10H+EA =32000H and stored in BX. (Note: Since a **word contains two bytes, the byte present at memory address 32000H and 32001H are moved to BL and BH respectively.)**

Base Addressing

- In this mode, the EA is the content of BX or BP register. When **BX** register is present in the instruction, data is taken from the **data segment** and if **BP** is present in the instruction, data is taken from the **stack segment**.
- Examples:
- **MOV CL,[BX]**
- EA=(BX)=2000H
- Memory address=DSx10+(BX)=32000H. The **byte** from the memory address 32000H is read and stored in CL.
- **MOV DX,[BP]**
- EA=(BP)=1000H
- Memory address=SSx10+(BP)=41000H. The **word (2 bytes)** from the memory address 41000H is read and stored in DX.

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 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:

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

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

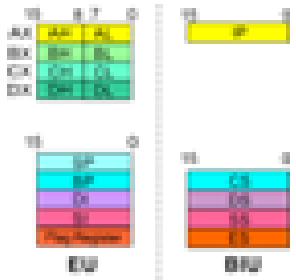
$(CL) \leftarrow (MA)$
$(CH) \leftarrow (MA + 1)$

Base Relative Addressing

- In this mode, **the EA is obtained by adding the content of the base register with an 8-bit or 16 bit displacement**. The displacement is a signed number with negative values represented in two's complement form. The 16 bit displacement can have value from -32768 to +32767 and 8 bit displacement can have the value from -128 to +127.
- **Examples:**
- **MOV AX,[BX+5]**
- EA=(BX)+5
- Memory address=DSx10H+(BX)+5 = 30000H+2000H+5=32005H
- The **word** from the memory address 32005H is taken and stored in **AX**.
- **MOV CH,[BX-100H]**
- EA=(BX)-100H
- Memory address=DSx10H+(BX)-100H = 30000H+2000H-100H=31F00H
- The **byte** from the memory address 31F00H is taken and stored in **CH**.

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 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]

Operations:

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

(AX) ← (MA) or,

(AL) ← (MA)
(AH) ← (MA + 1)

Index Addressing

- In this mode, the EA is the content of SI or DI register which is specified in the instruction. The data is taken from data segment.
- Examples:
- **MOV BL,[SI]**
- EA=(SI)=1000H
- Memory address=DSx10H+SI=30000H+1000H=31000H
- A **byte** from the memory address 31000H is taken and stored in BL.
- **MOV CX,[DI]**
- EA=(DI)=3000H
- Memory address=DSx10H+(DI)=30000H+3000H=33000H
- A **word** from the memory address 33000H is taken and stored in CX.

Index relative Addressing

- This mode is same as base relative addressing mode except that instead of BP or BX register, SI or DI register is used.
- Example
- **MOV BX,[SI-100H]**
- EA=(SI)-100H
- Memory Address=DSx10H+(SI)-100H = 30000H+1000H-100H =30F00H
- A **word** from the memory address 30F00H is taken and stored in BX.
- **MOV CL,[DI+10H]**
- EA=(DI)+10H
- Memory address=DSx10H+(DI)+10H =30000H+3000H+10H = 33010H
- A **byte** from the memory address 33010H is taken and stored in CL.

Base plus index Addressing

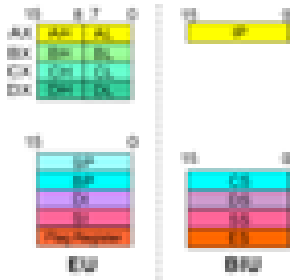
- In this mode, the EA is obtained by adding the content of a base register and index register.
- Example
- **MOV AX,[BX+SI]**
- EA=(BX)+(SI)
- Memory address=DSx10H+(BX)+(SI)=30000H+2000H+1000H
- =33000H
- A word from the memory address 33000H is taken and stored in AX.
- Base relative, index relative and base plus index addressing modes are used to access a byte or word type data from a **table of data or an array of data stored in data segment** one by one.

Base Relative plus index Addressing

- In this mode, the EA is obtained by adding the content of a base register, an index and a displacement.
- Example:
- MOV CX,[BX+SI+50H]
- EA= (BX)+(SI)+50H
- Memory address=DSx10H+(BX)+(SI)+50H
- =30000H+2000H+1000H+50H
- =33050H
- A word from the memory address 33050H is taken and stored in CX.
- Base relative plus index addressing is used to **access a byte or a word in a particular record of a particular file in memory**. A particular application program may process many files stored in the data segment.
- **Each file contains many records and a record contains few bytes or words of data. In base relative plus index addressing, base register may be used to hold the offset address of a particular file in the data segment; index register may be used to hold the offset address of a particular record within that file and the relative value is used to indicate the offset address of particular byte or word within that record.**

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]
```

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,

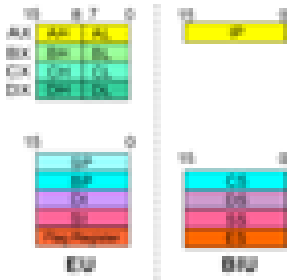
$(DL) \leftarrow (MA)$
$(DH) \leftarrow (MA + 1)$

Program memory addressing modes

- Program memory addressing modes are used with **JMP and CALL** instructions and consist of three distinct forms namely direct, relative and indirect.
- Direct Addressing: The direct program memory addressing stores both the segment and offset address where the control has to be transferred with the opcode.
- The above instruction is equivalent to JMP 32000H.
- When it is executed, the 16 bit offset value 2000H is loaded in IP register and the 16 bit segment value 3000H is loaded in CS.
- When the microprocessor calculates the memory address from where it has to fetch an instruction using the relation CSx10H+IP, the address 32000H will be obtained using the above CS and IP values.

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 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.

Relative Addressing

- The term relative here means relative to the instruction pointer (IP). Relative JMP and CALL instructions contain either an 8 bit or a 16 bit signed displacement that is added to the current instruction pointer and based on the new value of IP thus obtained, the address of the next instruction to be executed is calculated using the relation CSx10H+IP.

- The 8-bit or 16 bit signed displacement which allows a forward memory reference or a reverse memory reference. A one byte displacement is used in short jump and call instructions, and a two byte displacement is used in near jump and call instructions. Both types are considered intrasegment jumps since the program control is transferred anywhere within the current code segment.

- An 8 bit displacement has a jump range of between +127 and -128 bytes from the next instruction while a 16 bit displacement has a jump range of between -32768 and +32767 bytes from the next instruction following the jump instruction in the program. The opcode of relative short jump and near jump instructions are EBH and E9H respectively.
- While using assembler to develop 8086 program, the assembler directive SHORT and NEAR PTR is used to indicate short jump and near jump instruction respectively.
- Examples:
- a) JMP SHORT OVER
- b) JMP NEAR PTR FIND
- In the above examples, OVER and FIND are the label of memory locations that are present in the same code segment in which the above instructions are present.

Indirect Addressing

- The indirect jump or CALL instructions use either any 16 bit register (AX,BX,CX,DX,SP,BP,SI or DI) or any relative register ([BP],[BX],[DI] or [SI]) or any relative register with displacement. The opcode of indirect jump instruction is FFH. It can be either intersegment indirect jump or intrasegment indirect jump instruction
- If a 16 bit register holds the jump address of in a indirect JMP instruction, the jump is near. If the CX register contains 2000H and JMP CX instruction present in a code segment is executed, the microprocessor jumps to offset address 2000H in the current code segment to take the next instruction for execution (This is done by loading the IP with the content of CX without changing the content of CS).
- When the instruction **JMP [DI]** is executed, the microprocessor first reads a word in the current data segment from the offset address specified by DI and puts that word in IP register. Now using this new value of IP, 8086 calculates the address of the memory location where it has to jump using the relation CSx10H+IP.

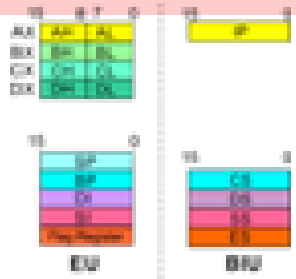
Stack memory addressing mode

- The stack holds data temporarily and also stores return address for procedures and interrupt service routines. The stack memory is a last-in, first-out (LIFO) memory. Data are placed into the stack using **PUSH** instruction and taken out from the stack using **POP** instruction.
- The **CALL** instruction uses the stack to hold the return address for procedures and **RET** instruction is used to remove return address from stack.
- The stack segment is maintained by two registers: the stack pointer (SP) and the stack segment register (SS). Always a word is entered into stack. Whenever a word of data is pushed into the stack, the higher-order 8 bits of the word are placed in the memory location specified by SP-1 (i.e. at address SSx10H + SP-1)and the lower-order 8 bits of the word are placed in the memory location specified by SP-2 in the current stack segment (SS) (i.e. at address SSx10H + SP-2). The SP is then decremented by 2. The data pushed into the stack may be either the content of a 16 bit register or segment register or 16 bit data in memory.

- Since SP gets decremented for every push operation, the stack segment is said to be growing downwards as for successive push operations, the data are stored in lower memory addresses in stack segment. Due to this, the SP is initialized with highest offset address according to the requirement, at the beginning of the program.

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



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.

Instruction set of 8086

- The instructions of 8086 are classified into **data transfer, arithmetic, logical, flag manipulation, control transfer, shift/rotate, string and machine control instructions.**

Instruction Set

8086 supports 6 types of instructions.

1. Data Transfer Instructions
2. Arithmetic Instructions
3. Logical Instructions
4. String manipulation Instructions
5. Process Control Instructions
6. Control Transfer Instructions

Instruction Set

1. Data Transfer Instructions

Instructions that are used to transfer data/ address in to registers, memory locations and I/O ports.

Generally involve two operands: Source operand and Destination operand of the same size.

Source: Register or a memory location or an immediate data
Destination : Register or a memory location.

The size should be a either a byte or a word.

A 8-bit data can only be moved to 8-bit register/ memory and a 16-bit data can be moved to 16-bit register/ memory.

Instruction Set

1. Data Transfer Instructions

Mnemonics: **MOV, XCHG, PUSH, POP, IN, OUT ...**

MOV reg2/ mem, reg1/ mem	
MOV reg2, reg1	(reg2) ← (reg1)
MOV mem, reg1	(mem) ← (reg1)
MOV reg2, mem	(reg2) ← (mem)
MOV reg/ mem, data	
MOV reg, data	(reg) ← data
MOV mem, data	(mem) ← data

XCHG reg2/ mem, reg1	
XCHG reg2, reg1	(reg2) ↔ (reg1)
XCHG mem, reg1	(mem) ↔ (reg1)

Instruction Set

1. Data Transfer Instructions

Mnemonics: MOV, XCHG, PUSH, POP, IN, OUT ...

PUSH reg16/ mem	
PUSH reg16	$(SP) \leftarrow (SP) - 2$ $MA_S = (SS) \times 16_{10} + SP$ $(MA_S ; MA_S + 1) \leftarrow (reg16)$
PUSH mem	$(SP) \leftarrow (SP) - 2$ $MA_S = (SS) \times 16_{10} + SP$ $(MA_S ; MA_S + 1) \leftarrow (mem)$
POP reg16/ mem	
POP reg16	$MA_S = (SS) \times 16_{10} + SP$ $(reg16) \leftarrow (MA_S ; MA_S + 1)$ $(SP) \leftarrow (SP) + 2$
POP mem	$MA_S = (SS) \times 16_{10} + SP$ $(mem) \leftarrow (MA_S ; MA_S + 1)$ $(SP) \leftarrow (SP) + 2$

Instruction Set

2. Arithmetic Instructions

Mnemonics: **ADD**, ADC, SUB, SBB, INC, DEC, MUL, DIV, CMP...

ADD reg2/ mem, reg1/mem	
ADC reg2, reg1	(reg2) ← (reg1) + (reg2)
ADC reg2, mem	(reg2) ← (reg2) + (mem)
ADC mem, reg1	(mem) ← (mem)+(reg1)
ADD reg/mem, data	
ADD reg, data	(reg) ← (reg)+ data
ADD mem, data	(mem) ← (mem)+data
ADD A, data	
ADD AL, data8	(AL) ← (AL) + data8
ADD AX, data16	(AX) ← (AX) +data16

Instruction Set

2. Arithmetic Instructions

Mnemonics: ADD, ADC, SUB, SBB, INC, DEC, MUL, DIV, CMP...

CMP reg2/mem, reg1/ mem	

CMP reg2, reg1	Modify flags ← (reg2) – (reg1)
	
	If (reg2) > (reg1) then CF=0, ZF=0, SF=0
	If (reg2) < (reg1) then CF=1, ZF=0, SF=1
	If (reg2) = (reg1) then CF=0, ZF=1, SF=0

CMP reg2, mem	Modify flags ← (reg2) – (mem)
	
	If (reg2) > (mem) then CF=0, ZF=0, SF=0
	If (reg2) < (mem) then CF=1, ZF=0, SF=1
	If (reg2) = (mem) then CF=0, ZF=1, SF=0

CMP mem, reg1	Modify flags ← (mem) – (reg1)
	
	If (mem) > (reg1) then CF=0, ZF=0, SF=0
	If (mem) < (reg1) then CF=1, ZF=0, SF=1
	If (mem) = (reg1) then CF=0, ZF=1, SF=0

Instruction Set

2. Arithmetic Instructions

Mnemonics: ADD, ADC, SUB, SBB, INC, DEC, MUL, DIV, CMP...

CMP reg/mem, data	

CMP reg, data	Modify flags ← (reg) − (data)
	
	If (reg) > data then CF=0, ZF=0, SF=0
	If (reg) < data then CF=1, ZF=0, SF=1
	If (reg) = data then CF=0, ZF=1, SF=0

CMP mem, data	Modify flags ← (mem) − (mem)
	
	If (mem) > data then CF=0, ZF=0, SF=0
	If (mem) < data then CF=1, ZF=0, SF=1
	If (mem) = data then CF=0, ZF=1, SF=0

Writing 8086 program ; an example

- Write a program to add a **word** type data located at offset 0800H (Least Significant Byte) and 0801H (Most Significant Byte) in the segment address 3000H to another word type data located at offset 0700H (Least Significant Byte) and 0701H (Most Significant Byte) in the same segment. Store the result at offset 0900H and 0901H in the same segment. Store the carry generated in the above addition in the same segment at offset 0902H.
-

Writing 8086 program ; an example

```
        MOV AX, 3000H
        MOV DS, AX          ; initialize DS with value 3000H
        MOV AX, [800H]      ; get first data word in AX
        ADD AX, [700H]      ; add AX with second data word
        MOV [900H], AX      ; store AX at the offset 900H & 901H
        JC   CARRY          ; if carry=1, go to the place CARRY
        MOV [902H], 00H     ; no carry; hence store 00H at the offset 902H
        JMP END             ; go to the place END
CARRY:  MOV [902H], 01H     ; store 01H at the offset 902H
END:    HLT                 ; stop
```