Introduction to Assembly Language Programming

- Reference Book: Assembly Language Programming and Organization of IBM PC
- Ytha Yu
- Charles Marut

Chapter -1

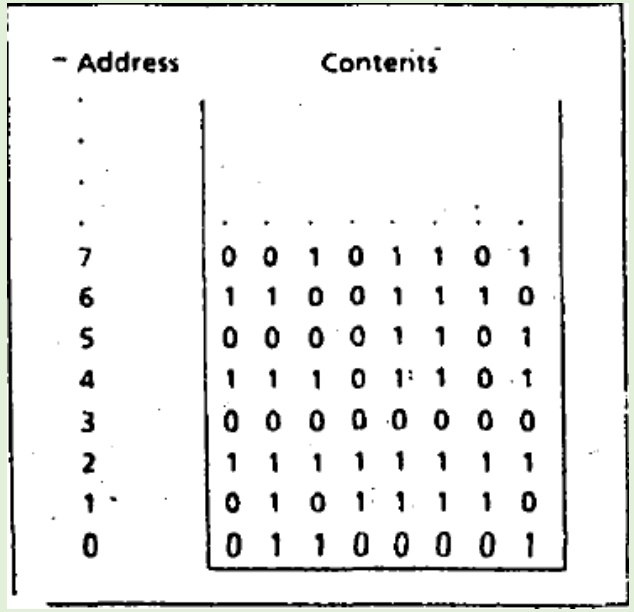
Microcomputer Systems

Component of a Microcomputer Systems

- Composed of Digital circuits
- Mainly consist of 3 parts
- Central processing unit (CPU)
- Memory circuits
- I/O circuits

Memory

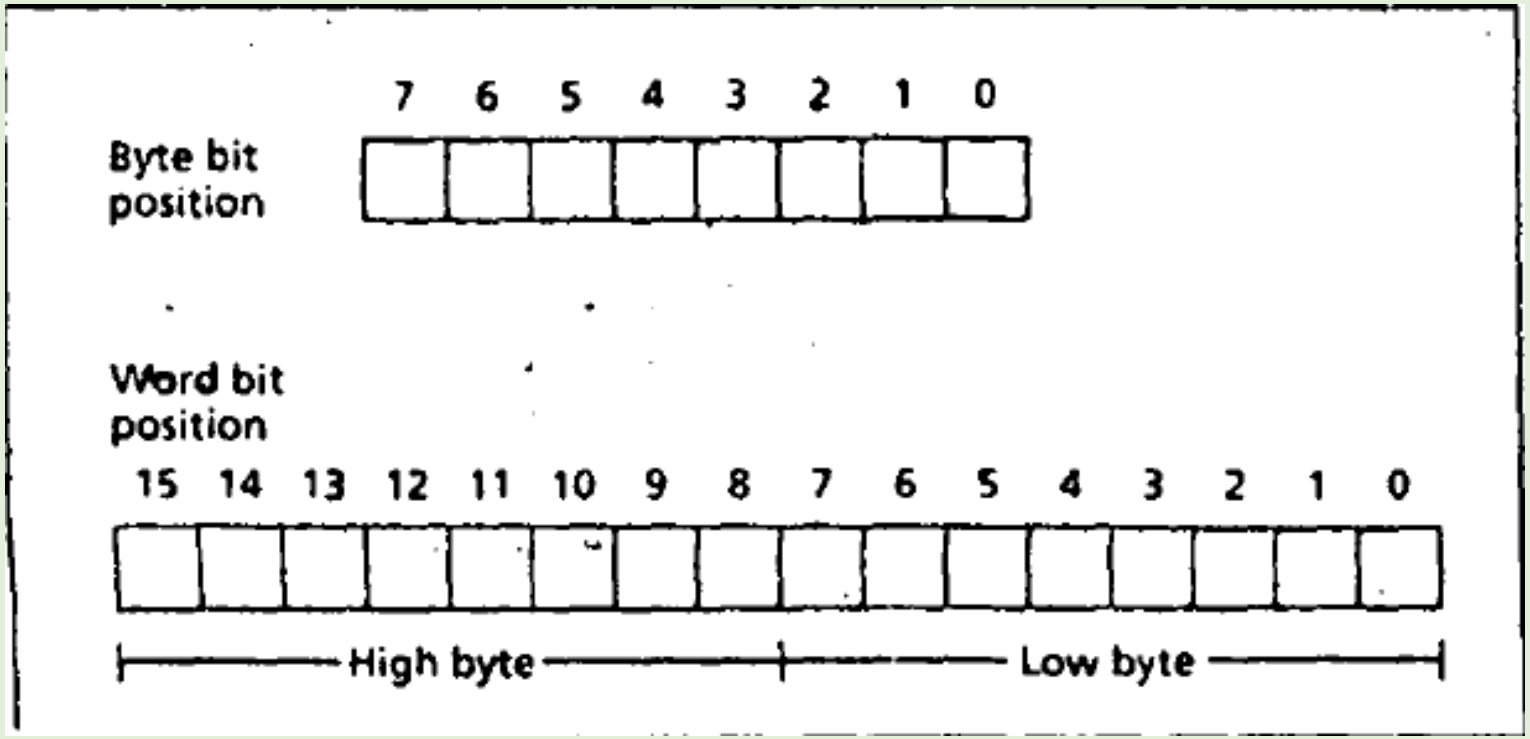
- A memory circuit element can store one bit of data
- Organized into groups of 8 bits = 1 byte
- Each memory byte is identified by a number called **address**
- Data stored in a memory byte is called its **contents**
- Address of a memory byte is fixed
- Content of a memory byte is not fixed and subject to change



Memory of Intel 8086 Microprocessor

- 8086 processor uses 20 bits of address
- How many memory bytes can be accessed?
- Answer : 1 MB
- Two bytes form a word
- The lower address of the two memory bytes is used as the address of the memory word.
- Example: So, memory word of address 2 is made up of the memory bytes with the address 2 and 3

Bit Position

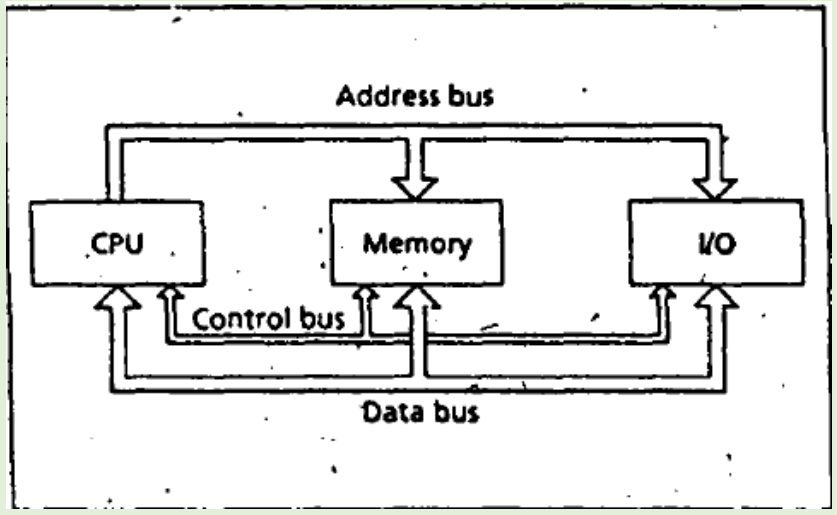


Memory Operations

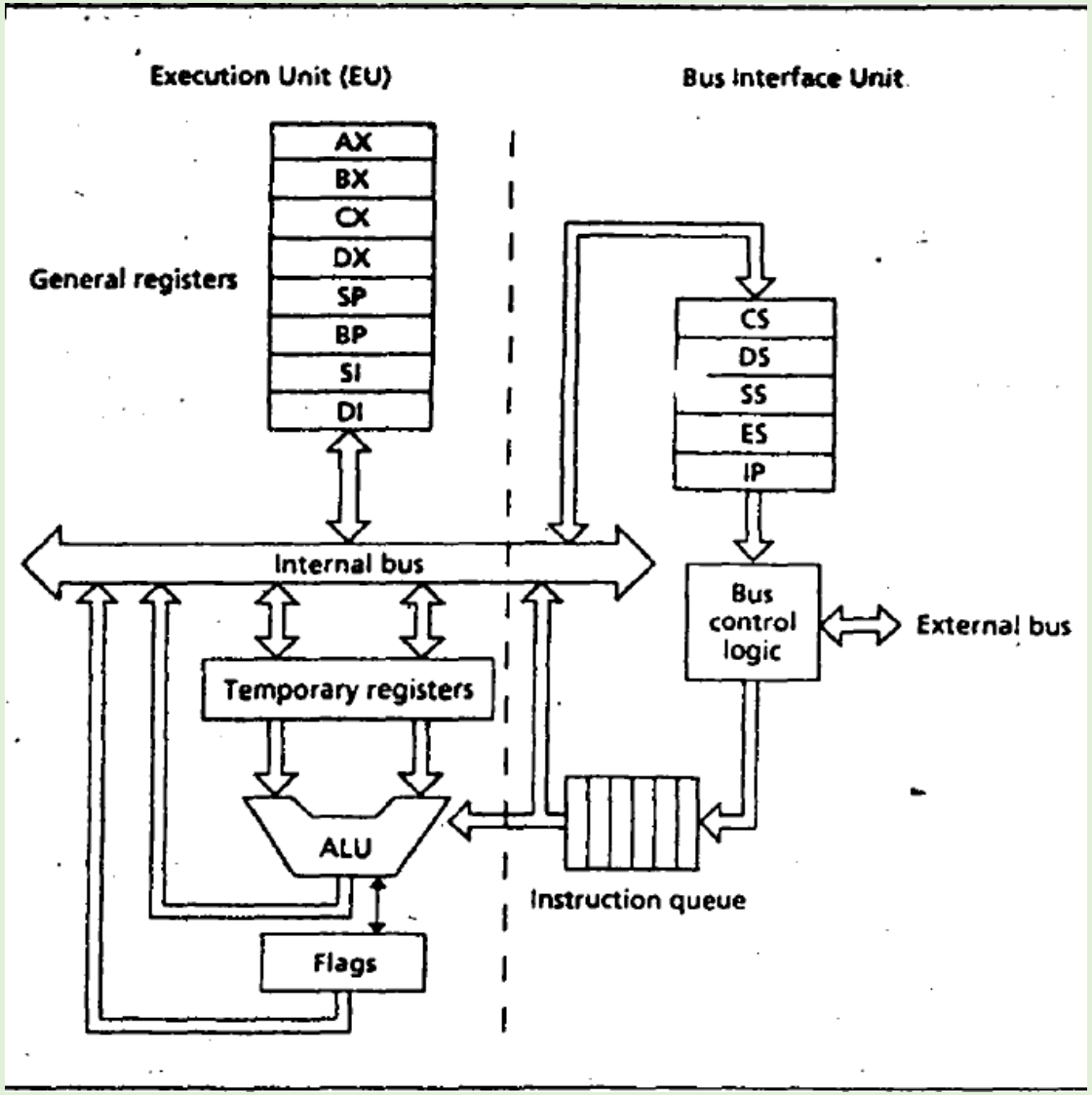
- Read: fetch the contents of a location. Processor only gets a copy of the data.
- Write: the data written become the new contents of the location; original contents are thus lost.

RAM, ROM and Buses

- RAM: Random Access Memory
- ROM: Read Only Memory
- Buses: processor communicates with memory and I/O circuits using signals that travel along a set of wires or connections called buses that connect the different components.
- Three types of buses
- Address bus
- Data bus
- Control Bus



Intel 8086 Microprocessor Organization



EU and BIU

- EU (Execution Unit) :

- Contains a circuit called ALU

- Registers : like a memory location except referred with a name rather than number

- BIU (Bus Interface Unit): facilitates communication between the EU and the memory or I/O circuits.

- Responsible for transmitting address, data and control signals on the buses.

Instruction Execution

- Machine Instruction has two parts: an **opcode** and **operands**
- Opcode specifies the type of operation and the operands are often given as memory addresses to the data to be operated on.
- The cpu goes through the following steps to execute a machine instruction(fetch execution cycle)

Fetch

1. Fetch an instruction from memory.
2. Decode the instruction to determine the operation
3. Fetch data from memory if necessary

Instruction Execution contd.

- Execute

4. Perform the operation on the data
5. Store the result in memory if needed.

Programming Languages

- Machine Language
- Assembly Language
- High Level Languages

Advantages of Assembly Languages

- Efficiency : Assembly Language is so close to machine language that a well-written assembly language program produces a faster, shorter machine language program. Some operations such as reading and writing to specific memory locations can be easily in assembly language.
- Many high-level languages accept subprograms written in assembly language.
- Understanding the way how computer “thinks”

Chapter - 2

Representation of Numbers and Characters

Number Systems

- Decimal
- Binary
- Hexadecimal

Conversion of Number Systems

- *Converting Binary and Hex to Decimal*
- *Converting Decimal* to *Binary and Hex*
- *Conversions Between Hex and Binary*

Addition and Subtraction

- Binary Addition and Subtraction
- Hexadecimal Addition and Subtraction

Integer Representation in Computer

- LSB (Least Significant Bit)
- MSB (Most Significant Bit)
- Unsigned Integer: represents a magnitude, so it is never negative.

- Byte : 0-255
- Word : 0-65535

- Signed Integer: can be positive or negative. The most significant bit is reserved for the sign: 1 means negative and 0 means positive. Negative integers are stored in the computer in a special way known as two's complement.

One’s Complement and Two’s Complement

- Word Interpretation of integer 5

- 5: 0000 0000 0000 0101

- one's complement of *5* = 1111 1111 1111 1010

- two's complement of 5 = 1111 1111 1111 1011

Decimal Interpretation

- Unsigned Decimal Interpretation: Just do a binary to decimal conversion.
- Signed Decimal Interpretation:

- If the msb is 0, signed decimal is same as unsigned decimal
- If the msb is 1, the number is negative, call it –N. to find N, just take the two's complement of and convert to decimal as before.

Character Representation

- ASCII Code

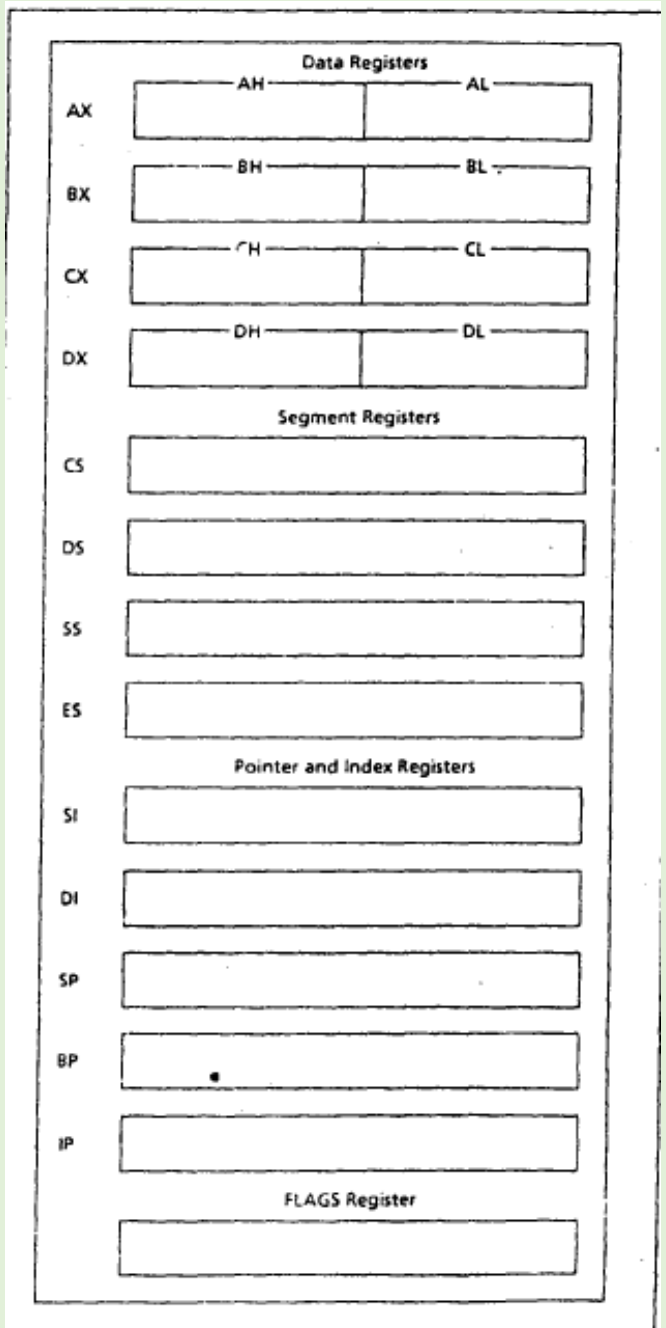
Chapter -3

Organization of the IBM Personal Computers

Registers of 8086 Microprocessors

- The registers are classified according to the functions they perform:
- Data Registers : hold data for an operation
- Address Registers: hold the address of instruction or data
- Status Registers: keeps the current status of the processor
- 8086 has four general data registers
- Address Registers are divided into ***segment, pointer and index registers.***
- Status Register is called the ***Flags*** register
- In total, fourteen ***16 bit*** registers

Registers


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

Data Registers

- AX (Accumulator Register) : AX is the preferred register to use in arithmetic, logic, and transfer instructions
- BX (Base Register) : BX serves as a base register.
- CX (Count Register): Program loop construction are facilitated by the use of CX. Other operations like shift and rotate also use CX
- DX (Data Register): DX is used in multiplication and division. It is also used in I/O operations.

Segment Registers

- Memory Segment: Memory segment is a block of 2^16 (or 64 k) of consecutive memory bytes.
- Each segment is identified by a segment number, starting with 0.
- A segment number is 16 bits, so the highest segment number is FFFFh.
- So, a memory location can be specified by providing a segment number and offset in **segment:offset** form which is known as logical address

Mapping from Logical Address to Physical Address

- To obtain the 20 bit address, Shift the segment address by 4 bits and add the offset address to it.

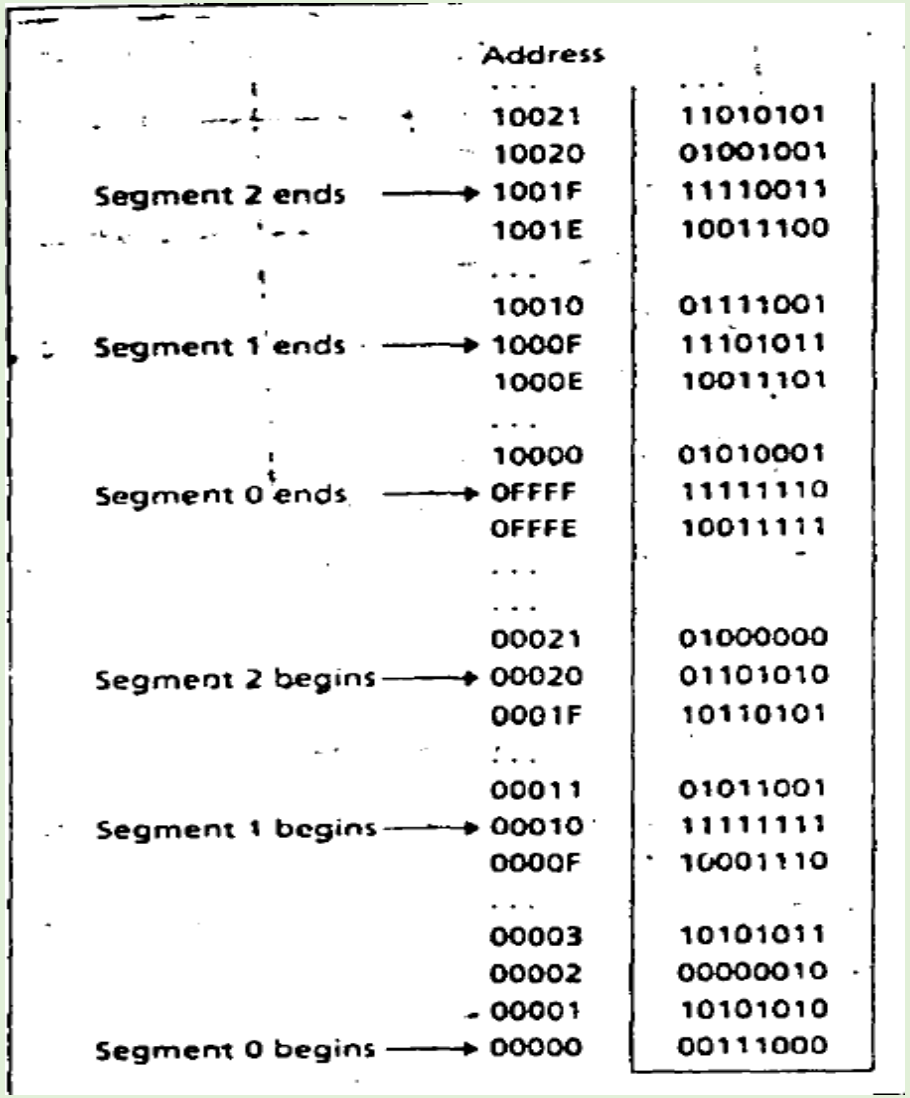
If A4FB:4872 is the address in segment:offset form, it means offset 4872h within segment A4FB h. And the physical address is:

A4FB0h

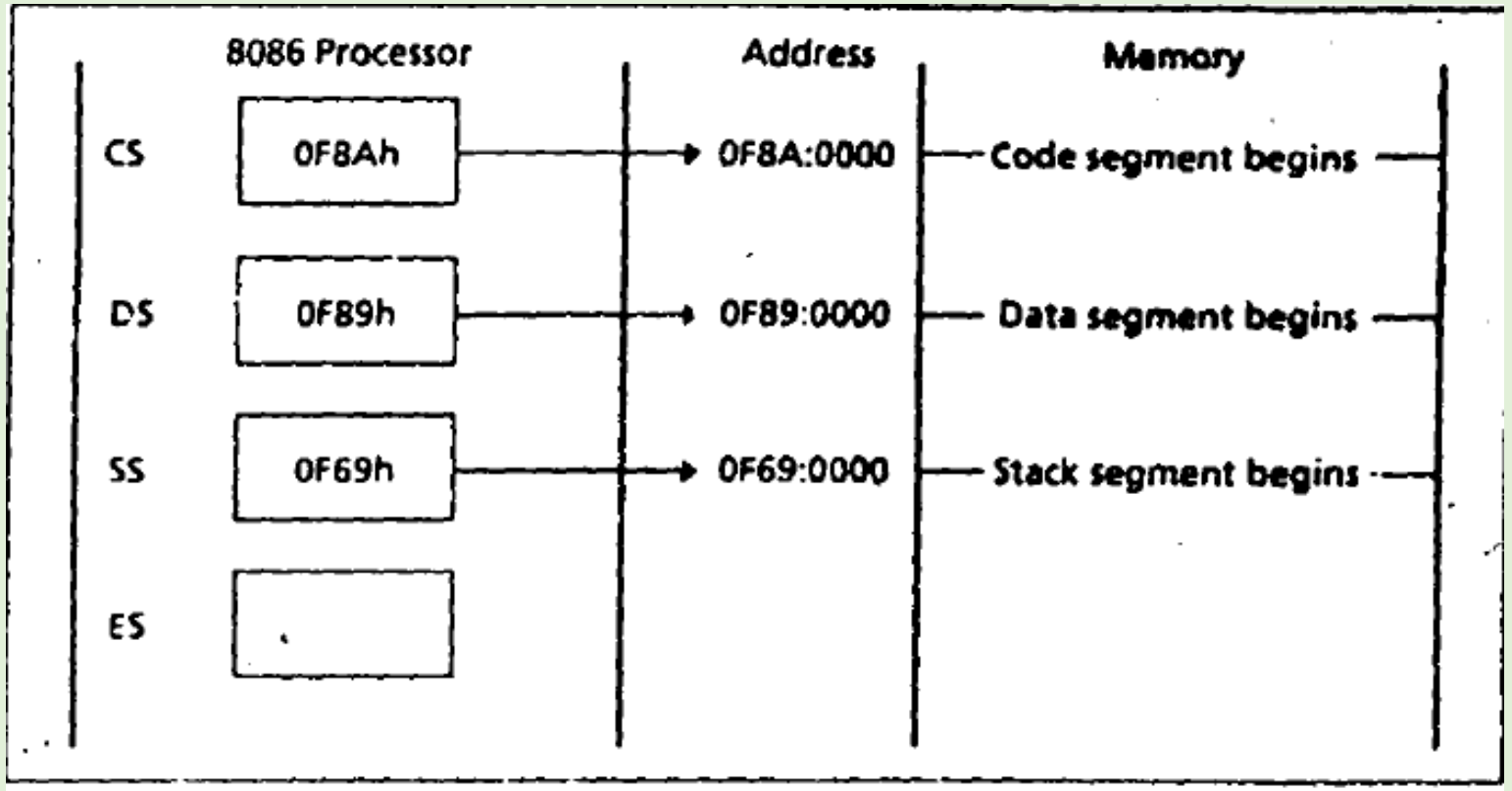
+ 4872h

A9822h

Overlapping of Segment Address



Program Segments



Pointer and Index Registers

- SP (Stack Pointer) : is used in conjunction to SS
- BP (Base Pointer): is used to primarily to access data on the stack. However, unlike SP, we can also use BP to access data in the other segments.
- SI (Source Index) : used to point to memory locations in the data segment addressed by DS.
- DI (Destination Index) : performs the same functions as SI. There is a class of instructions, called string operations that use DI to access memory locations addressed by ES.

IP and Flags Register

- To access Instructions, the 8086 uses the registers CS and IP.
- The CS register contains the segment number of the next instruction, and the IP contains the offset. IP is updated each time an instruction is executed so that it will point to the next Instruction.
- Unlike the other registers, the IP cannot be directly manipulated by an instruction
- Flags Register is used to indicate the status of a microprocessor.

Chapter - 4

Introduction to IBM PC assembly language

Statements

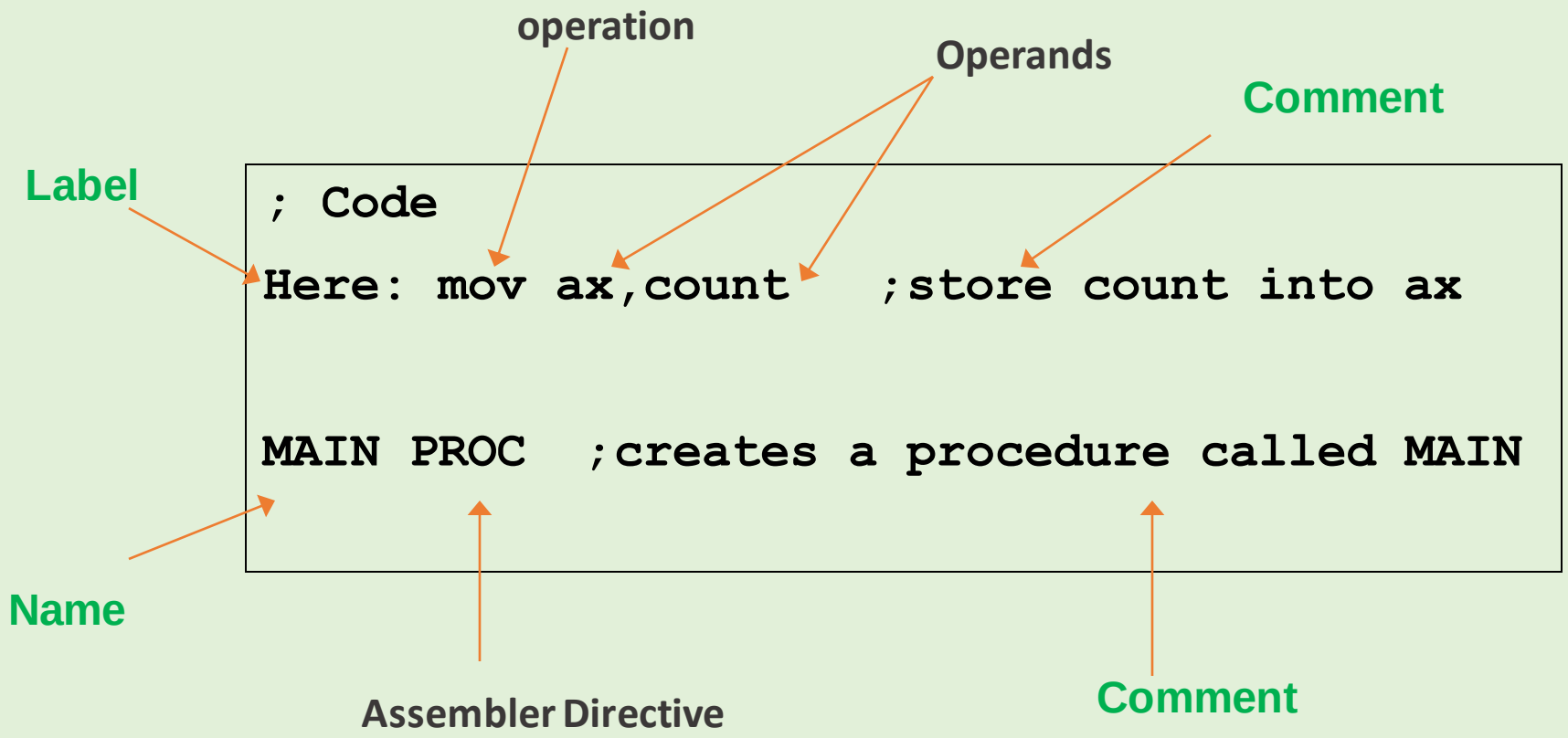
- Syntax:

  name operation operand(s) comments

  - name and comment are optional
  - Number of operands depend on the instruction
- One statement per line
  - At least one blank or tab character must separate the field.

- Each statement is either:
  - Instruction (translated into machine code)
  - Assembler Directive (instructs the assembler to perform some specific task such as allocating memory space for a variable or creating a procedure)

Statement Example



Name/Label Field

- The assembler translates names into memory addresses.
- Names can be 1 to 31 character long and may consist of letter, digit or special characters. If period is used, it must be first character.
- Embedded blanks are not allowed.
- May not begin with a digit.
- Not case sensitive

Examples of legal names	Examples of illegal names
COUNTER_1	TWO WORDS
@character	2abc
.TEST	A45.28
DONE?	YOU&ME

Operation Field: Symbolic operation (Op code)

- Symbolic op code translated into Machine Language op code
- ***Examples***: ADD, MOV, SUB

- In an assembler directive, the operation field represents Pseudo-op code
- Pseudo-op is not translated into Machine Language op code, it only tells assembler to do something.
- ***Example***: **PROC** psuedo-op is used to create a procedure

Operand Field

- An instruction may have zero, one or more operands.
- In two-operand instruction, first operand is destination, second operand is source.
- For an assembler directive, operand field represents more information about the directive
- ***Examples***

```
NOP             ;no operand, does nothing
INC AX          ;one operand, adds 1 to the contents of AX
ADD AX, 2       ;two operands, adds value 2 to the contents of AX
```

Comments

- Optional
- Marked by semicolon in the beginning
- Ignored by assembler
- Good practice

Program Data

- Processor operates only on binary data.
- In assembly language, you can express data in:
  - Binary
  - Decimal
  - Hexadecimal
  - Characters
- Numbers
  - For Hexadecimal, the number must begin with a decimal digit. E.g.: write 0ABCh not only ABCH.
  - Cannot contain any non-digit character. E.g.: 1,234 not allowed
- Characters enclosed in single or double quotes.
  - ASCII codes can be used
  - No difference in “A” and 41h

Contd..

- Use a radix symbol (suffix) to select binary, octal, decimal, or hexadecimal

```
6A15h          ; hexadecimal
0BAF1h         ; leading zero required
32q            ; octal
1011b          ; binary
35d            ; decimal (default)
```

Variables

- Each variable has a data type and is assigned a memory address by the program.
- Possible Values:
  - Numeric, String Constant, Constant Expression, ?
  - **8 Bit Number Range**: Signed (-128 to 127), Unsigned (0-255)
  - **16 Bit Number Range:** Signed (-32,678 to 32767), Unsigned (0-65,535)
  - **?** To leave variable uninitialized

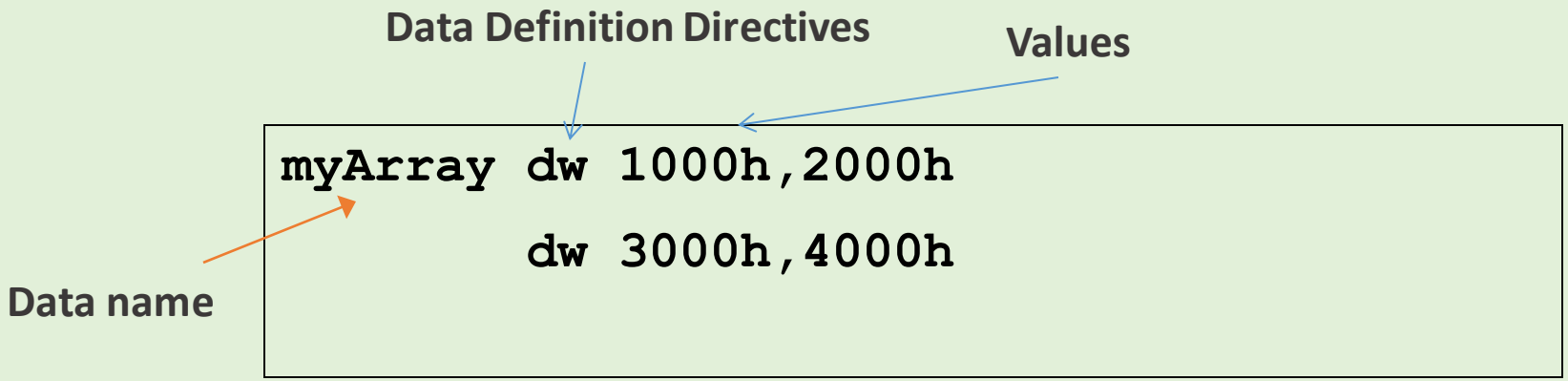
Contd..

- Syntax

  variable_name type initial_value

  variable_name type value1, value2, value3

- Data Definition Directives Or Data Defining Pseudo-ops
  - DB, DW, DD, DQ, DT



Remember: you can skip variable name!

Contd..

Examples	Bytes	Description	Pseudo-ops
var1 DB ‘A’ Var2 DB ? array1 DB 10, 20,30,40	1	Define Byte	**DB**
var2 DW ‘AB’ array2 DW 1000, 2000	2	Define Word	**DW**
Var3 DD -214743648	4	Define Double Word	**DD**

Note:
Consider

var2 DW 10h

Still in memory the value saved will be 0010h

Arrays

- Sequence of memory bytes or words
- **Example 1:**

  B_ARRAY DB 10h, 20h, 30h

Symbol	Address	Contents
B_ARRAY	0200h	10h
B_ARRAY+1	0201h	20h
B_ARRAY+2	0202h	30h

***If B_ARRAY is assigned offset address 0200h by assembler**

Example 2

- W_ARRAY DW 1000, 40, 29887, 329

***If W_ARRAY is assigned offset address 0300h by assembler**

Symbol	Address	Contents
W_ARRAY	0300h	1000d
W_ARRAY+ 2	0302h	40d
W_ARRAY+ 4	0304h	29887d
W_ARRAY+ 6	0306h	329d

- **High & Low Bytes of a Word**

  WORD1 DW 1234h
- Low Byte = 34h, symbolic address is WORD1
- High Byte = 12h, symbolic address is WORD1+1

Character String

LETTERS DB ‘ABC’

Is equivalent to

LETTERS DB 41h, 42h, 43h

- Assembler differentiates between upper case and lower case.
- Possible to combine characters and numbers.

MSG DB ‘HELLO’, 0Ah, 0Dh, ‘$’

Is equivalent to

MSG DB 48h, 45h, 4Ch, 4Ch, 4Fh, 0Ah, 0Dh, 24h

Example 3

- Show how character string “RG 2z” is stored in memory starting at address 0.
- Solution:

Address	Character	ASCII Code (HEX)	ASCII Code (Binary) [Memory Contents]
0	R	52	0101 0010
1	G	47	0100 0111
2	Space	20	0010 0000
3	2	32	0011 0010
4	z	7A	0111 1010

Named Constants

- Use symbolic name for a constant quantity
- **Syntax**:

  name **EQU** constant
- **Example**:

  LF **EQU** 0Ah

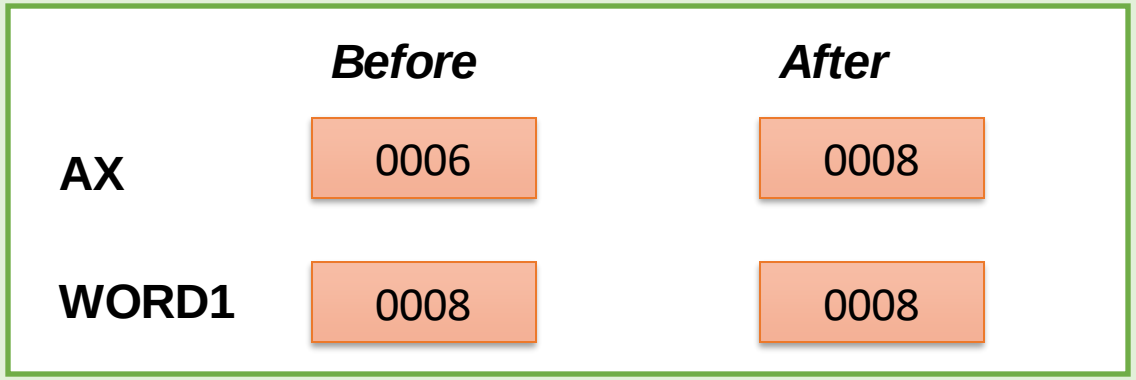
- No memory allocated

MOV

- Transfer data
  - Between registers
  - Between register and a memory location
  - Move a no. directly to a register or a memory location
- Syntax

  MOV *destination, source*
- Example

  MOV *AX, WORD1*



- **Difference?**
  - MOV AH, 'A'
  - MOV AX, 'A'

Legal Combinations of Operands for MOV

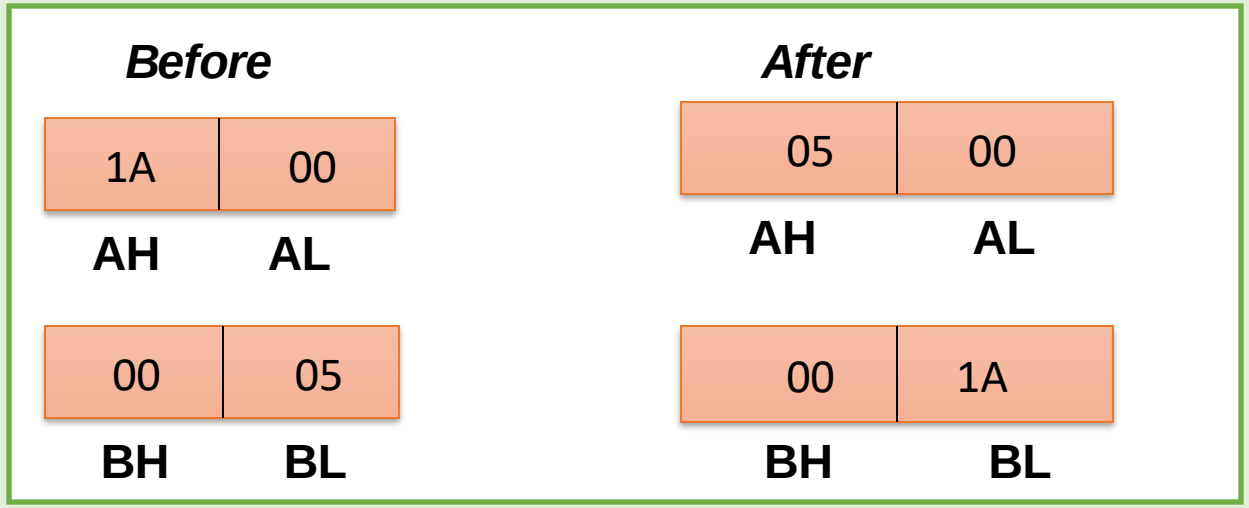
Destination Operand	Source Operand	Legal
General Register	General Register	YES
General Register	Memory Location	YES
General Register	Segment Register	YES
General Register	Constant	YES
Memory Location	General Register	YES
Memory Location	Memory Location	NO
Memory Location	Segment Register	YES
Memory Location	Constant	YES

XCHG

- Exchange the contents of
  - Two registers
  - Register and a memory location
- Syntax

  XCHG *destination, source*
- Example

  XCHG *AH, BL*



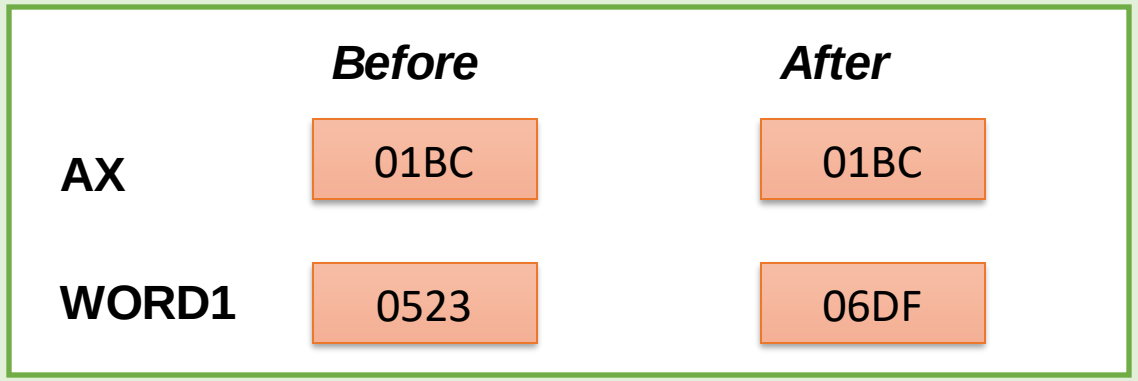
Legal Combinations of Operands for XCHG

Destination Operand	Source Operand	Legal
General Register	General Register	YES
General Register	Memory Location	YES
Memory Location	General Register	YES
Memory Location	Memory Location	NO

ADD Instruction

- To add contents of:
  - Two registers
  - A register and a memory location
  - A number to a register
  - A number to a memory location
- Example

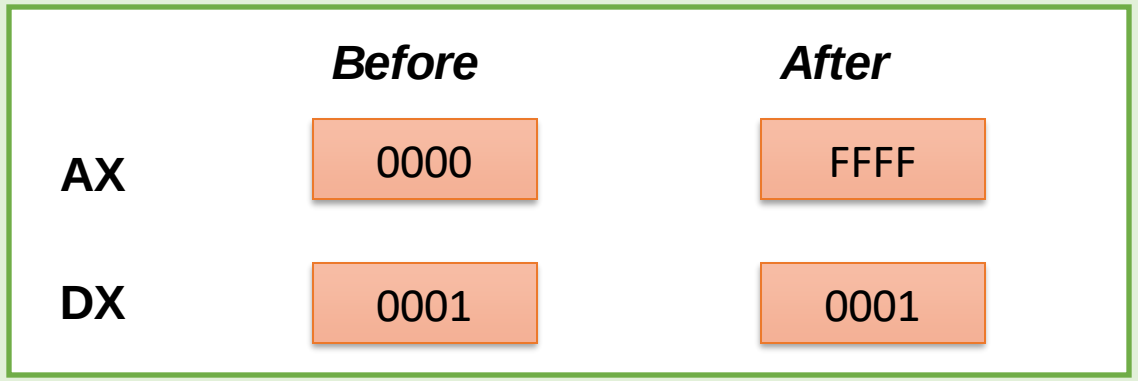
  ADD WORD1, AX



SUB Instruction

- To subtract the contents of:
  - Two registers
  - A register and a memory location
  - A number from a register
  - A number from a memory location
- Example

  SUB AX, DX



Legal Combinations of Operands for ADD & SUB instructions

Destination Operand	Source Operand	Legal
General Register	General Register	YES
General Register	Memory Location	YES
General Register	Constant	YES
Memory Location	General Register	YES
Memory Location	Memory Location	NO
Memory Location	Constant	YES

Contd..

ADD BYTE1, BYTE2 ILLEGAL instruction

- Solution?

  MOV AL, BYTE2

  ADD BYTE1, AL

- **What can be other possible solutions?**

-

INC & DEC

- **INC** (increment) instruction is used to add 1 to the contents of a register or memory location.
  - Syntax: INC *destination*
  - Example: INC WORD1

- **DEC** (decrement) instruction is used to subtract 1 from the contents of a register or memory location.
  - Syntax: DEC *destination*
  - Example: DEC BYTE1

- Destination can be 8-bit or 16-bits wide.
- Destination can be a register or a memory location.

Contd..

INC WORD1

DEC BYTE1

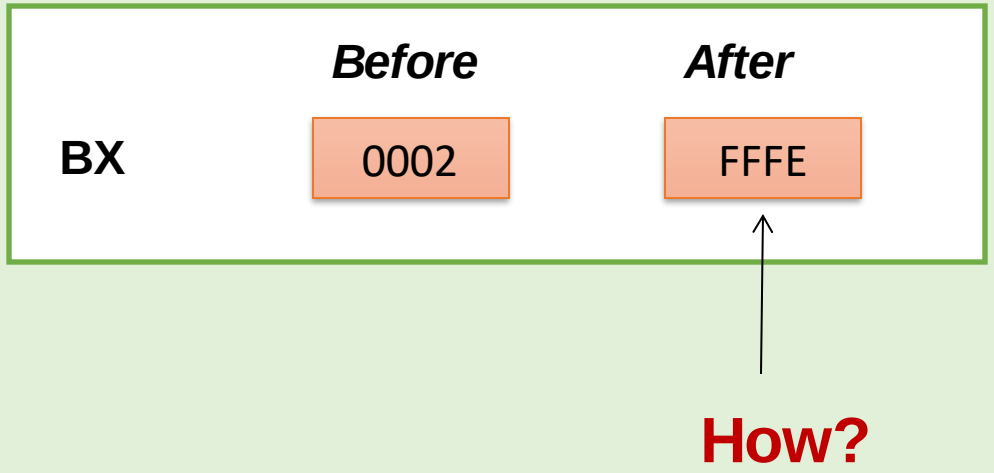
NEG

- Used to negate the contents of destination.
- Replace the contents by its 2's complement.
- Syntax

  NEG *destination*

- Example

  NEG BX



Examples

- Consider instructions: MOV, ADD, SUB, INC, DEC, NEG
- **A** and **B** are two word variables
- Translate statements into assembly language:

Statement	Translation
B = A	MOV AX, A MOV B, AX

A = 5 - A	MOV AX, 5
	SUB AX, A
	MOV A, AX
	OR
	NEG A
	ADD A, 5

Contd..

Statement	Translation

A = B – 2 x A	MOV AX, B
	SUB AX, A
	SUB AX, A
	MOV A, AX

- **Remember:** Solution not unique!
- **Be careful!** Word variable or byte variable?

Program Segments

- Machine Programs consists of
  - Code
  - Data
  - Stack
- Each part occupies a memory segment.
- Same organization is reflected in an assembly language program as **Program Segments**.
- Each program segment is translated into a memory segment by the assembler.

Memory Models

- Determines the size of data and code a program can have.
- Syntax:

  .MODEL memory_model

Model	Description
SMALL	code in one segment, data in one segment
MEDIUM	code in more than one segment, data in one segment
COMPACT	code in one segment, data in more than one segment
LARGE	Both code and data in more than one segments No array larger than 64KB
HUGE	Both code and data in more than one segments array may be larger than 64KB

Data Segment

- All variable definitions
- Use **.DATA** directive
- For Example:

```
.DATA
WORD1 DW 2
BYTE1 DB 10h
```

Stack Segment

- A block of memory to store stack
- Syntax

  .STACK size

  - Where size is optional and specifies the stack area size in bytes
  - If size is omitted, 1 KB set aside for stack area

- For example:

  .STACK 100h

Code Segment

- Contains a program’s instructions
- Syntax

.CODE name

- Where name is optional
- Do not write name when using SMALL as a memory model

Putting it Together!

```
.MODEL SMALL
.STACK 100h

.DATA
   ;data definition go here
.CODE
   ;instructions go here
```