

# Chapter 4

## Introduction to IBM PC Assembly Languages

```
TITLE EXAMPLE: EXAMPLE1
```

← Program Title (Optional)

```
.MODEL SMALL
```

```
.STACK 100H
```

```
.DATA
```

```
;data definitions go here
```

```
.CODE
```

```
MAIN PROC
```

```
;instructions go here
```

```
MAIN ENDP
```

```
;other procedures go here
```

```
END MAIN
```

```

TITLE EXAMPLE: EXAMPLE1
.MODEL SMALL
.STACK 100H
.DATA
;data definitions go here
.CODE
MAIN PROC

;instructions go here

MAIN ENDP
;other procedures go here
END MAIN

```

## ❑ Details in 4.7.1 (Table 4.4)

SMALL-> code in one segment

data in one segment

MEDIUM -> code in more than one

data in one

COMPACT-> code in one

data in more than one

Similarly LARGE and HUGE

```
TITLE EXAMPLE: EXAMPLE1
.MODEL SMALL
.STACK 100H
.DATA
;data definitions go here
.CODE
MAIN PROC

;instructions go here

MAIN ENDP
;other procedures go here
END MAIN
```



### Details in 4.7.3

.STACK    size\_of\_the\_stack  
if not specified, then the default size  
is 1KB

```
TITLE EXAMPLE: EXAMPLE1
.MODEL SMALL
.STACK 100H
.DATA
;data definitions go here
.CODE
MAIN PROC

;instructions go here

MAIN ENDP
;other procedures go here
END MAIN
```

- ☐ **Details in 4.7.2**
- ☐ Data Segment
- ☐ Variable and Constant declarations are done here

```
TITLE EXAMPLE: EXAMPLE1
.MODEL SMALL
.STACK 100H
.DATA
;data definitions go here
.CODE
MAIN PROC

;instructions go here

MAIN ENDP
;other procedures go here
END MAIN
```

- ❑ **Details in 4.7.4**
- ❑ Inside a code segment, Instructions are organized as procedures.

```
TITLE EXAMPLE: EXAMPLE1
.MODEL SMALL
.STACK 100H
.DATA
;data definitions go here
.CODE
MAIN PROC
;instructions go here
MAIN ENDP
;other procedures go here
END MAIN
```

☐ **Details in 4.7.4**

- ☐ The program instructions are packed inside procedures.
- ☐ The definition of a single procedure is  
name PROC  
; body of the procedure  
name ENDP
- ☐ Here it is an example of main procedure

```
TITLE EXAMPLE: EXAMPLE1
.MODEL SMALL
.STACK 100H
.DATA
;data definitions go here
.CODE
MAIN PROC

;instructions go here

MAIN ENDP
;other procedures go here
END MAIN
```

- ☐ Details in 4.7.4
- ☐ Other procedure declarations are done here



```
TITLE EXAMPLE: EXAMPLE1
.MODEL SMALL
.STACK 100H
.DATA
;data definitions go here
.CODE
MAIN PROC

;instructions go here

MAIN ENDP
;other procedures go here
END MAIN
```

- ☐ Details in 4.7.4
- ☐ Write this at the end of all procedures (that means at the end of the code segment)

# Assembly Language Syntax

- Assembly language code is generally not case sensitive
- Program consists of statements, one per line.
- Each statement is of two type
- Type1: instruction

name operation operand(s) comment

An Example:

START:        MOV CX,5        ;initialize counter

☐ Details are in 4.1  
4.1.1 to 4.1.4

- Type2: Assembler directive

An Example:

MAIN        PROC

```
TITLE EXAMPLE: EXAMPLE1
.MODEL SMALL
.STACK 100H
.DATA
;data definitions go here
.CODE
MAIN PROC

;instructions go here

MAIN ENDP
;other procedures go here
END MAIN
```



☐ Let's now give focus on it !!

```

TITLE PGM4_1: ECHO PROGRAM
.MODEL SMALL
.STACK 100H
.CODE
MAIN PROC
    ;display prompt
    MOV AH,2
    MOV DL,'?'
    INT 21H
    ; input a character
    MOV AH,1
    INT 21H
    MOV BL,AL
    ; go to a new line
    MOV AH,2
    MOV DL,0DH
    INT 21H
    MOV DL,0AH
    INT 21H
    ; display character
    MOV DL,BL
    INT 21H
    ; return to DOS
    MOV AH,4CH
    INT 21H
MAIN ENDP
END MAIN

```

- ❑ A sample instruction  
The format of MOV instruction is  
MOV destination, source
- ❑ XCHG instruction is also like MOV
- ❑ Details of MOV and XCHG are in 4.5.1

Table 4.2 Legal Combinations of Operands for MOV and XCHG

#### MOV

Source Operand	Destination Operand			
	General register	Segment register	Memory location	Constant
General register	yes	yes	yes	no
Segment register	yes	no	yes	no
Memory location	yes	yes	no	no
Constant	yes	no	yes	no

#### XCHG

Source Operand	Destination Operand	
	General register	Memory location
General register	yes	yes
Memory location	yes	no

```

TITLE PGM4_1: ECHO PROGRAM
.MODEL SMALL
.STACK 100H
.CODE
MAIN PROC
    ;display prompt
    MOV AH,2
    MOV DL,'?'
    INT 21H
    ; input a character
    MOV AH,1
    INT 21H
    MOV BL,AL
    ; go to a new line
    MOV AH,2
    MOV DL,0DH
    INT 21H
    MOV DL,0AH
    INT 21H
    ; display character
    MOV DL,BL
    INT 21H
    ; return to DOS
    MOV AH,4CH
    INT 21H
MAIN ENDP
END MAIN

```

- ❑ There are other instructions like ADD, SUB, INC, DEC and NEG
- ❑ Details are in 4.5.2 and 4.5.3

```

TITLE PGM4_1: ECHO PROGRAM
.MODEL SMALL
.STACK 100H
.CODE
MAIN PROC
    ;display prompt
    MOV AH,2
    MOV DL,'?'
    INT 21H
    ; input a character
    MOV AH,1
    INT 21H
    MOV BL,AL
    ; go to a new line
    MOV AH,2
    MOV DL,0DH
    INT 21H
    MOV DL,0AH
    INT 21H
    ; display character
    MOV DL,BL
    INT 21H
    ; return to DOS
    MOV AH,4CH
    INT 21H
MAIN ENDP
END MAIN

```

- ❑ INT 21H
- ❑ Details are in 4.8 and 4.8.1

#### Function number

1

#### Routine

single-key input

2

single-character output ✓

9

character string output

#### Function 1: Single-Key Input

Input: AH = 1

Output: AL = ASCII code if character key is pressed  
= 0 if non-character key is pressed

#### Function 2: Display a character or execute a control function

Input: AH ≠ 2

DL = ASCII code of the display character or  
control character

Output: AL = ASCII code of the display character or  
control character

Time to run our first program on our  
own!!



TITLE PGM4\_1.5: SAMPLE INPUT

.MODEL SMALL

.STACK 100H

.DATA

VAR1 DB ?

.CODE

MAIN PROC

; initialize DS

MOV DX, @DATA

MOV DS, DX

; display message

MOV DL, 5

MOV AH, 1

INT 21H

; move to variable

MOV VAR1, AL

; add 2 with the value

ADD VAR1, 2

MOV DL, VAR1

MOV AH, 2

INT 21H

; return to DOS

MOV AH, 4CH

INT 21H

MAIN ENDP

END MAIN

☐ The format of variable declaration

variable\_name DB initial\_value

variable\_name DW initial\_value

☐ See the table 4.1 for more

Example:

var1 DB 4

var2 DW 'A'

☐ If we want keep the variable uninitialized  
then we use a question mark (?)

var1 DB ?

☐ For Details see Section 4.2 and 4.3



```
TITLE PGM4_1.5: SAMPLE INPUT
.MODEL SMALL
.STACK 100H
.DATA
VAR1 DB ?
.CODE
MAIN PROC
```

```
    ;initialize DS
    MOV DX, @DATA
    MOV DS, DX
```

```
    ;display message
```

```
    MOV DL, 5
```

```
    MOV AH, 1
```

```
    INT 21H
```

```
    ;move to variable
```

```
    MOV VAR1, AL
```

```
    ; add 2 with the value
```

```
    ADD VAR1, 2
```

```
    MOV DL, VAR1
```

```
    MOV AH, 2
```

```
    INT 21H
```

```
    ; return to DOS
```

```
    MOV AH, 4CH
```

```
    INT 21H
```

```
MAIN ENDP
```

```
END MAIN
```

- ☐ DS must be initialized to use the data segment
- ☐ For Details Section 4.11(Page 74)

Time for the second one

```

TITLE PGM4_2: PRINT
.MODEL SMALL
.STACK 100H
.DATA
MSG DB 'HELLO!$'
.CODE
MAIN PROC
    ;initialize DS
    MOV AX,@DATA
    MOV DS,AX
    ;display message
    LEA DX,MSG
    MOV AH,9
    INT 21H
    ; return to DOS
    MOV AH,4CH
    INT 21H
MAIN ENDP
END MAIN

```

### ☐ Working with Array:

MSG DB 'HELLO!\$'

☐ If the address of variable MSG is 100h then

Symbol	Address	Contents
MSG	100h	48h
MSG+1	101h	45h
MSG+2	102h	4Ch
MSG+3	103h	4Ch
MSG+4	104h	4Fh
MSG+5	105h	24h

☐ The dollar character(\$) is used to indicate the end of a string

☐ The alternate representation of

MSG DB 'HELLO!\$'

is

MSG DB 48h, 45h, 4Ch, 4Ch, 4Fh, 24h

☐ For details see Section 4.3.3

```

TITLE PGM4_2: PRINT
.MODEL SMALL
.STACK 100H
.DATA
MSG DB 'HELLO!$'
.CODE
MAIN PROC
    ;initialize DS
    MOV AX,@DATA
    MOV DS,AX
    ;display message
    LEA DX,MSG
    MOV AH,9
    INT 21H
    ; return to DOS
    MOV AH,4CH
    INT 21H
MAIN ENDP
END MAIN

```

### ☐ Displaying a String

INT 21h, Function 9:

Display a String

Input: DX = offset address of string.

The string must end with a '\$' character.

☐ To load the offset address into DX we need to use

LEA(Load Effective address)

☐ For details see Section 4.11

# Problem 1

- ❑ Write an assembly program that will take a lower case letter and convert it to an upper case letter

TITLE PGM4\_3: CASE CONVERSION PROGRAM

.MODEL SMALL

.STACK 100H

.DATA

CR EQU 0DH,

LF EQU 0AH

MSG1 DB 'ENTER A LOWER CASE LETTER: \$'

MSG2 DB 0DH,0AH,'IN UPPER CASE IT IS: '

CHAR DB ?, '\$'

.CODE

MAIN PROC

;initialize DS

MOV AX,@DATA

MOV DS,AX

;print user prompt

LEA DX,MSG1

MOV AH,9

INT 21H

;input a character and convert to upper case

MOV AH,1

INT 21H

SUB AL,20H

MOV CHAR,AL

;display on the next line

LEA DX,MSG2

MOV AH,9

INT 21H

; return to DOS

MOV AH,4CH

INT 21H

MAIN ENDP

END MAIN

# Problem 2

- ❑ Write an assembly program that will take two small digits (one is less than 5 and another is less than 6), add these and display the output.

```

TITLE PGM4_4: SMALL ADDITION PROGRAM
.MODEL SMALL
.STACK 100H
.DATA
VAR1      DB  ?
VAR2      DB  ?
MSG1      DB  'PLEASE ENTER THE FIRST DIGIT< <5 >: $'
MSG2      DB  'PLEASE ENTER THE SECOND DIGIT< <6 >: $'
R_MSG     DB  'THE RESULT IS: $'
.CODE
MAIN PROC
;initialize DS
MOV AX,@DATA
MOV DS,AX
;print the first message
MOV AH,9
LEA DX,MSG1
INT 21H
; input first value
MOV AH,1
INT 21H
SUB AL,'0'
MOV BL,AL
; go to a new line
MOV AH,2
MOV DL,0DH
INT 21H
MOV DL,0AH
INT 21H

```

```

;print the second message
MOV AH,9
LEA DX,MSG2
INT 21H
;input second value
MOV AH,1
INT 21H
SUB AL,'0'
MOV BH,AL
;addition operation
ADD BH,BL
ADD BH,'0'
;go to a new line
MOV AH,2
MOV DL,0DH
INT 21H
MOV DL,0AH
INT 21H
;display result message
LEA DX,R_MSG
MOV AH,9
INT 21H
; display character
MOV AH,2
MOV DL,BH
INT 21H
; return to DOS
MOV AH,4CH
INT 21H
MAIN ENDP
END MAIN

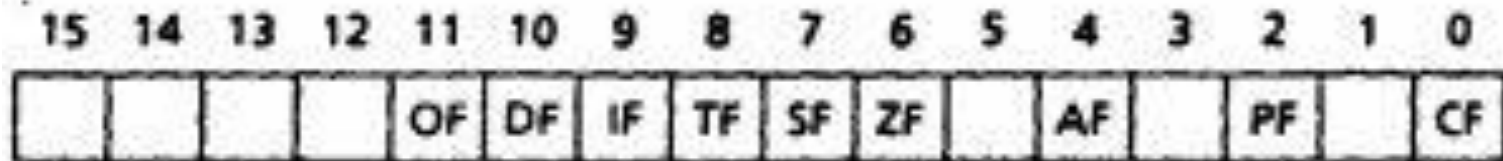
```



# Chapter 5

## The Processor Status and the FLAGS Register

# The FLAGS Register



**Table 5.1** *Flag Names and Symbols*

**Status Flags**

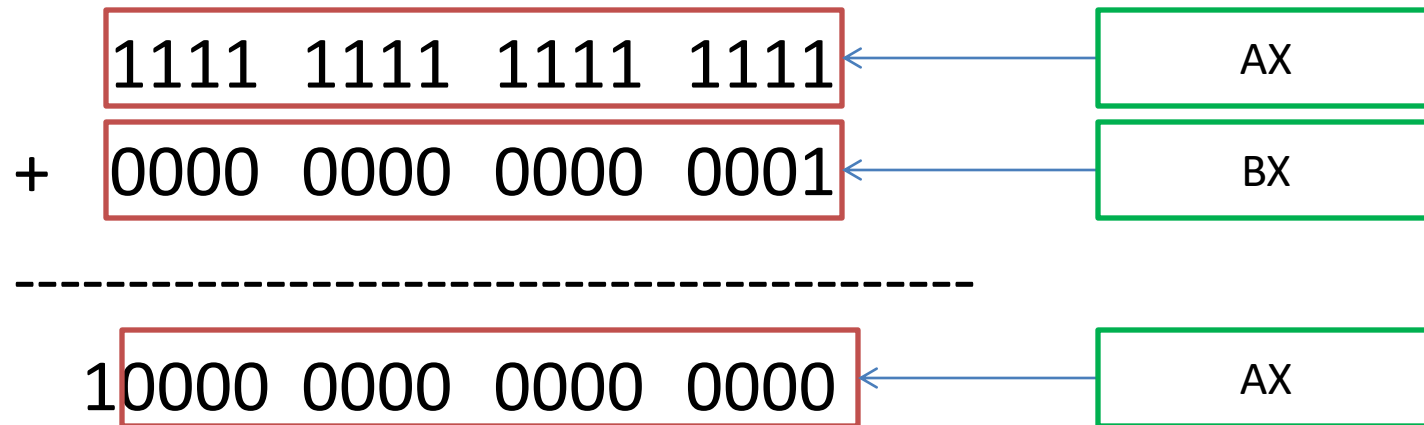
<b>Bit</b>	<b>Name</b>	<b>Symbol</b>
0	Carry flag	CF
2	Parity flag	PF
4	Auxiliary carry flag	AF
6	Zero flag	ZF
7	Sign flag	SF
11	Overflow flag	OF

**Control Flags**

<b>Bit</b>	<b>Name</b>	<b>Symbol</b>
8	Trap flag	TF
9	Interrupt flag	IF
10	Direction flag	DF

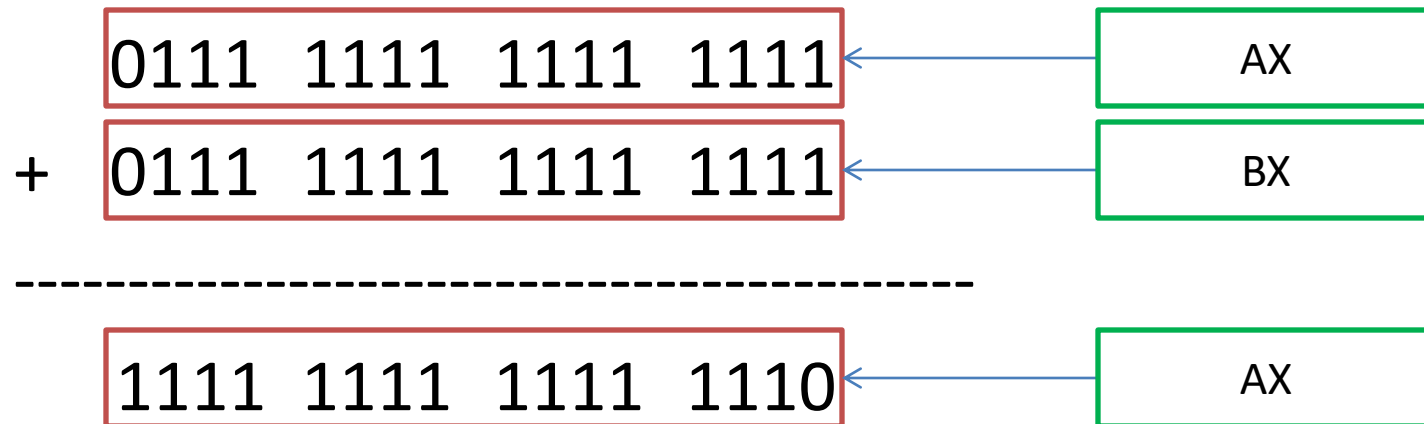
# Overflow

- Example of only unsigned overflow
- IF AX=FFFFh and BX=0001h
- ADD AX,BX



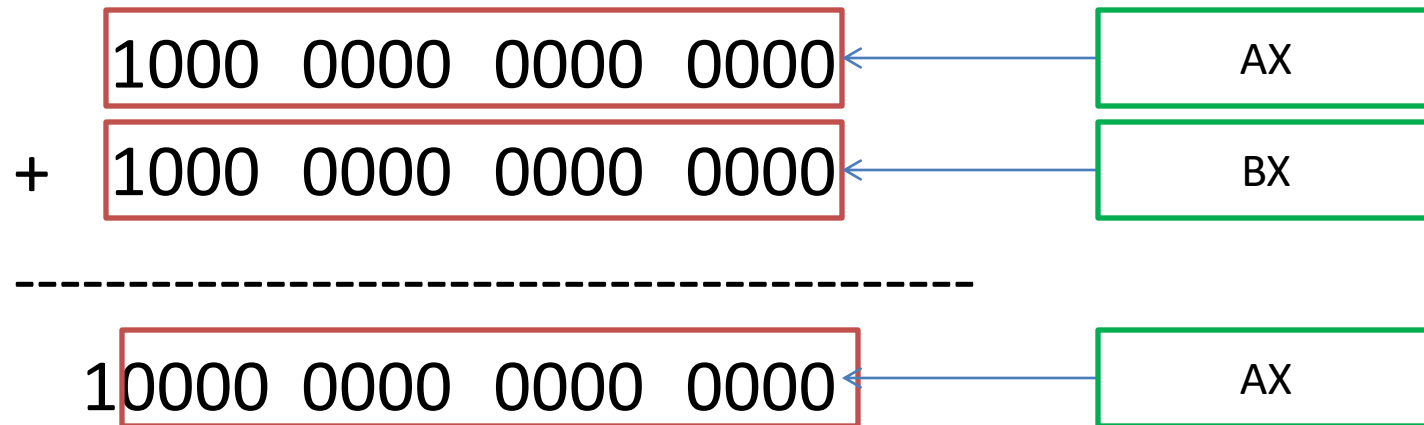
# Overflow

- Example of only signed overflow
- IF AX=7FFFh and BX=7FFFh
- ADD AX,BX



# Overflow

- Example of both signed and unsigned overflow
- IF AX=8000h and BX=8000h
- ADD AX,BX



# Unsigned Overflow

- Causes Carry Flag to become 1
- When occurs?
  - If the result of addition is more than the limit
  - If a big number is subtracted from a small number

# Signed Overflow

- Causes Overflow flag(OF) to become 1
- When occurs?
  - The result of addition has two different signs
  - The result of subtraction has two different signs

# How Instructions Affect the Flags

Instructions	Affects Flags
MOV/XCHG	none
ADD/SUB	all
INC/DEC	all except CF
NEG	All(CF=1 unless result is 0, OF=1 if word operand is 8000h Or byte operand is 80h)



Go through 5.1 to 5.3 for all the  
details of chapter 5

**YOU DESERVE**



**"A HUGE THANKS"**