Weeks 4-5

8088/8086 Microprocessor Programming
### Assemble, Link and Run a Program

#### Steps in creating an executable Assembly Language Program

<table>
<thead>
<tr>
<th>Step</th>
<th>Input</th>
<th>Program</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Editing</td>
<td>Usually Keyboard Editor (Text word editors etc.)</td>
<td>Myfile.asm</td>
<td></td>
</tr>
<tr>
<td>2. Assemble</td>
<td>Myfile.asm</td>
<td>MASM</td>
<td>Myfile.obj</td>
</tr>
<tr>
<td>3. Link</td>
<td>Myfile.obj</td>
<td>LINK</td>
<td>Myfile.exe</td>
</tr>
</tbody>
</table>

#### Example Assembly Code

```assembly
.model small
.stack 100h
main proc
    mov ah,01h
    int 21h
    MOV AH, 4Ch
    INT 21H
.end main
```

#### Debugging or Codeview

<table>
<thead>
<tr>
<th>Origin</th>
<th>Group</th>
<th>Class</th>
<th>Start</th>
<th>Stop</th>
<th>Length</th>
<th>Name</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000H</td>
<td>00007H</td>
<td>_TEXT</td>
<td>00008H</td>
<td>00008H</td>
<td>00000H</td>
<td>_DATA</td>
<td>DATA</td>
</tr>
<tr>
<td>00010H</td>
<td>0010FH</td>
<td>00100H</td>
<td>STACK</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Program entry point at 0000:0000
Instructions

[label:] mnemonic [operands] [: comment]

- Address identifier: Max 31 characters
- Indicates the opcode generating the instruction

Ex. START: MOV AX,BX ; copy BX into AX

Does not generate any machine code
Assembly Language Basics

• Character or String Constants
  – ‘ABC’
  – ‘X’
  – “This isn’t a test”
  – “4096”

• Numeric Literals
  – 26
  – 1Ah
  – 1101b
  – 36q
  – 2BH
  – 47d
Statements

- `longarrayDefinition dw 1000h,1020h,1030h \n  ,1040h, 1050h, 1060h, 1070h`
  
  Lines may break with "\" character

- Identifier name limit of max 247 characters

- Case insensitive

- Variable
  - `Count1 db 50 ;a variable (memory allocation)`

- Label:
  - If a name appears in the code area of the program it is a label.

```
LABEL1: mov ax,0
         mov bx,1
LABEL2: jmp Label1 ;jump to label1
```
Assembler Directives

.MODEL SMALL ; selects the size of the memory model usually sufficient max 64K code 64K data
.STACK ; size of the stack segment
.DATA ; beginning of the data segment
.CODE ; beginning of the code segment

Ex:

.CHAR
.DATA
DATAW DW 213FH
DATA1 DB 52H
SUM DB ? ; nothing stored but a storage is assigned

Ex:

.CODE
PROGRAMNAME PROC; Every program needs a name
.... ; program statements
PROGRAMNAME ENDP
END PROGRAMNAME
Sample Program

title Hello World Program (hello.asm)
; This program displays "Hello, world!"
.model small
.stack 100h
.data
message db "Hello, world!",0dh,0ah,'$' ;newline+eoc
.code
main proc
  mov  ax,@data ; address of data
  mov  ds,ax
  mov  ah,9
  mov  dx,offset message ;disp.msg.starting at location
  int  21h ;or LEA dx,message will do!
  mov  ax,4C00h ; halt the program and return
  int  21h
main endp
end main
# DataTypes and Data Definition

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATA1</td>
<td>DB</td>
<td>25</td>
</tr>
<tr>
<td>DATA2</td>
<td>DB</td>
<td>10001001b</td>
</tr>
<tr>
<td>DATA3</td>
<td>DB</td>
<td>12h</td>
</tr>
<tr>
<td></td>
<td>ORG</td>
<td>0010h</td>
</tr>
<tr>
<td>DATA4</td>
<td>DB</td>
<td>“2591”</td>
</tr>
<tr>
<td></td>
<td>ORG</td>
<td>0018h</td>
</tr>
<tr>
<td>DATA5</td>
<td>DB</td>
<td>?</td>
</tr>
</tbody>
</table>

This is how data is initialized in the data segment

<table>
<thead>
<tr>
<th>Address</th>
<th>Hex</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>00</td>
<td>19</td>
</tr>
<tr>
<td>0001</td>
<td>00</td>
<td>89</td>
</tr>
<tr>
<td>0002</td>
<td>00</td>
<td>12</td>
</tr>
<tr>
<td>0010</td>
<td>08</td>
<td>32 35 39 31</td>
</tr>
<tr>
<td>0018</td>
<td>00</td>
<td>00</td>
</tr>
</tbody>
</table>
DB DW DD

.data

MESSAGE2 DB '1234567'
MESSAGE3 DW 6667H
data1 db 1,2,3
db 45h
db 'a'
db 11110000b
data2 dw 12,13
dw 2345h
dd 300h

; how it looks like in memory
31 32 33 34 35 36 37
67 66
1 2 3
45
61
F0
0C 00 0D 00
45 23
00 03 00 00
More Examples

DB  6 DUP(FFh); fill 6 bytes with ffh

DW  954
DW  253Fh ; allocates two bytes
DW  253Fh

DD  5C2A57F2h ; allocates four bytes
DQ  12h ; allocates eight bytes

COUNTER1  DB  COUNT
COUNTER2  DB  COUNT
More assembly

• **OFFSET**
  - The offset operator returns the distance of a label or variable from the beginning of its segment. The destination must be 16 bits
  - `mov bx, offset count`

• **SEG**
  - The segment operator returns the segment part of a label or variable’s address.
    ```
    Push ds
    Mov ax, seg array
    Mov ds, ax
    Mov bx, offset array
    .
    Pop ds
    ```

• **DUP operator only appears after a storage allocation directive.**
  - `db 20 dup(?)`

• **EQU directive assigns a symbolic name to a string or constant.**
  - `Maxint equ 0ffffh`
  - `COUNT EQU 2`
Memory Models

- **Tiny**
  - code and data combined must be less than 64K
- **Small Code**
  - Code <=64K and Data<= 64K (seperate)
- **Medium Data**
  - Code <=64K any size multiple code seg
- **Compact Code**
  - Data <=64K any size multiple data seg
- **Large Code**
  - Code >64K and Data>64K multiple code and data seg
- **Huge**
  - Same as the Large except that individual vars can be >64K
The PTR Operator - Byte or word or doubleword?

- INC [20h] ; is this byte/word/dword? or
- MOV [SI],5
  - Is this byte 05?
  - Is this word 0005?
  - Or is it double word 00000005?

- To clarify we use the PTR operator
  - INC BYTE PTR [20h]
  - INC WORD PTR [20h]
  - INC DWORD PTR [20h]

- or for the MOV example:
  - MOV byte ptr [SI],5
  - MOV word ptr [SI],5
The PTR Operator

- Would we need to use the PTR operator in each of the following?

```plaintext
MOV AL, BVAL
MOV DL, [BX]
SUB [BX], 2
MOV CL, WVAL
ADD AL, BVAL+1
```

.data
BVAL DB 10H, 20H
WVAL DW 1000H
Simple Assembly Language Program

```assembly
.MODEL SMALL
.STACK 64
.DATA
DATA1 DB 52h
DATA2 DB 29h
SUM DB ?
.CODE
MAIN PROC FAR
    MOV AX,@DATA; copy the data segment into the DS reg.
    MOV DS,AX
    MOV AL,DATA1
    MOV BL,DATA2; or DATA1+1
    ADD AL,BL
    MOV SUM,AL
    MOV AH,4Ch
    INT 21h
MAIN ENDP
END MAIN
```
MS-DOS Functions and BIOS Calls

- BIOS is hardware specific
- BIOS is supplied by the computer manufacturer
- Resident portion which resides in ROM and nonresident portion IO.SYS which provides a convenient way of adding new features to the BIOS
80x86 Interrupts

• An interrupt is an event that causes the processor to suspend its present task and transfer control to a new program called the interrupt service routine (ISR)
• There are three sources of interrupts
  – Processor interrupts
  – Hardware interrupts generated by a special chip, for ex: 8259 Interrupt Controller.
  – Software interrupts
• Software Interrupt is just similar to the way the hardware interrupt actually works!. The INT Instruction requests services from the OS, usually for I/O. These services are located in the OS.
• INT has a range 0 → FFh. Before INT is executed AH usually contains a function number that identifies the subroutine.
80x86 Interrupts

- Each interrupt must supply a type number which is used by the processor as a pointer to an interrupt vector table (IVT) to determine the address of that interrupt’s service routine.

- **Interrupt Vector Table:** CPU processes an interrupt instruction using the interrupt vector table (This table resides in the lowest 1K memory).

- Each entry in the IVT=segment+offset address in OS, points to the location of the corresponding ISR.

- Before transferring control to the ISR, the processor performs one very important task:
  - It saves the current program address and flags on the stack.
  - Control then transfers to the ISR.
  - When the ISR finishes, it uses the instruction IRET to recover the flags and old program address from the stack.

- Many of the vectors in the IVT are reserved for the processor itself and others have been reserved by MS-DOS for the BIOS and kernel.
  - 10 -- 1A are used by the BIOS.
  - 20 -- 3F are used by the MS-DOS kernel.
80x86 Interrupts

- The number after the mnemonic tells which entry to locate in the table. For example INT 10h requests a video service.

```
mov...
int 10h
add
```

Interrupt service routine IRET

F000:F065
Entry for INT 10
Interrupt Vector Table

Interrupt vector (type number)

<table>
<thead>
<tr>
<th>0</th>
<th>Pointer to ISR 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Pointer to ISR 1 Single-Step</td>
</tr>
<tr>
<td>2</td>
<td>Pointer to ISR 2 NMI</td>
</tr>
<tr>
<td>3</td>
<td>Pointer to ISR 3 Breakpoint</td>
</tr>
<tr>
<td>4</td>
<td>Pointer to ISR 4 Overflow</td>
</tr>
<tr>
<td>5</td>
<td>Pointer to ISR 5</td>
</tr>
<tr>
<td>6</td>
<td>Pointer to ISR 6</td>
</tr>
<tr>
<td>FF</td>
<td>Pointer to ISR 255</td>
</tr>
<tr>
<td>FE</td>
<td>Pointer to ISR 254</td>
</tr>
</tbody>
</table>

Interrupt vector table (IVT)
1K (real mode)
2K (protected mode)

4 bytes (real mode) or
8 bytes (protected mode)

<table>
<thead>
<tr>
<th>Processor</th>
<th>Pointer Size</th>
<th>IVT Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Real Mode</td>
<td>4 bytes</td>
<td>Address 00000000–000003FF</td>
</tr>
<tr>
<td>Protected Mode</td>
<td>8 bytes</td>
<td>Anywhere in Physical Memory</td>
</tr>
</tbody>
</table>
Interrupts

• There are some extremely useful subroutines within BIOS or DOS that are available to the user through the INT (Interrupt) instruction.

• The INT instruction is like a FAR call; when it is invoked
  – It saves CS:IP and flags on the stack and goes to the subroutine associated with that interrupt.
  – Format:
    • INT xx ; the interrupt number xx can be 00-FFH
  – This gives a total of 256 interrupts
  – Common Interrupts
    • INT 10h Video Services
    • INT 16h Keyboard Services
    • INT 17h Printer Services
    • INT 21h MS-DOS services
  – Before the services, certain registers must have specific values in them, depending on the function being requested.
**Int 10 AH=02H SET CURSOR POSITION**

- **INT 10H function 02;** setting the cursor to a specific location
  - Function AH = 02 will change the position of the cursor to any location.
  - The desired cursor location is in DH = row, DL = column

![Image of a text editor window showing code and directory structure]
Int 10 03 GET CURSOR POSITION

- **INT 10H function 03**: get current cursor position
  MOV AH, 03
  MOV BH, 00
  INT 10H

- Registers DH and DL will have the current row and column positions and CX provides info about the shape of the cursor.

- Useful in applications where the user is moving the cursor around the screen for menu selection

Int 10 05 SWITCH VIDEO MODES

- **INT 10H function 05**: switch between video modes by adjusting AL
  MOV AH, 05h
  MOV AL, 01H; switch to video page 1
  INT 10H

  ; below will switch to video page 0
  MOV AH, 05h
  MOV AL, 00H; switch to video page 0
  INT 10H

  Extremely useful in text modes that support multiple pages!
  This is what we had before Windows™
INT 10 – AH=06 SCROLL

- INT 10H Function 06 (AH = 06) Scroll a screen windows.
  - Moves the data on the video display up or down. As screen is rolled the bottom is replaced by a blank line. Rows:0-24 from top, bottom: 0-79 from the left. (0,0) to (24,79). Lines scrolled can not be recovered!
  - AL = number of lines to scroll (with AL=00, window will be cleared)
  - BH = Video attribute of blank rows
  - CH, CL = Row,Column of upper left corner
  - DH, DL = Row,Column of lower right corner

```
mov ah,6h
mov al,0h
mov ch,0h
mov cl,0h
mov dh,24h
mov dl,01h
mov bh,7h
int 10h
```

Example: Clear the screen by scrolling it upward with a normal attribute.

```
00,00  00,79
12,39
24,00  24,79
```

Cursor Locations
Example Int10 06

- Init reg AH for the program
- Define the line of the “window” size to scroll
- Define the “the window”
- Halt the program
Example

The previous window scroll is applied on the amount of the window size (whole screen);
in order to clear the entire screen

```
MOV AX, 0600H
```
; scroll the entire page

```
MOV CX, 0000 ; upper left
MOV DX, 184FH ; lower right
INT 10H
```
INT 10 - 0A PRINT CHARACTERS

- Write *one or more* characters at the current cursor position
- This function can display any ASCII character.
- AH function code
- AL character to be written
- BH video page
- CX repetition factor; how many times the char will be printed
Int 10 – 0E PRINT SINGLE CHARACTER

Write out a single character
(Also stored in AL)
INT 21h

**INT 21H Option 01**: Inputs a single character with echo

– This function waits until a character is input from the keyboard, then echoes it to the monitor. After the interrupt, the input character will be in AL.

```
.model small
.stack 100h
.data
; ORG 0010h; offset
; DATA1 DB
.code
main proc
    mov ah, 01h
    int 21h
    mov ah, 4Ch
    int 21h
main endp
end main
```
**INT 21h**

- **INT 21H Option 0AH/09H**: Inputs/outputs a string of data stored at DS:DX
  - AH = 0AH, DX = offset address at which the data is located
  - AH = 09, DX = offset address at which the data located

<table>
<thead>
<tr>
<th>N</th>
<th>M</th>
<th>ch1</th>
<th>ch2</th>
<th>ch3</th>
<th>.</th>
<th>.</th>
<th>.</th>
<th>.</th>
</tr>
</thead>
</table>

Chars allowed

Actual # of chars

Chars Entered

**Ex.** What happens if one enters USA and then <RETURN>

```
0010 0011 0012 0013 0014 0015 0016 0017
```

```
06 03 55 53 41 0D FF FF
```

**ORG 0010H;**

```
DATA1 DB 6,?,6 DUP(0FFH)
```

**MOV AH, 0AH**

**MOV DX, OFFSET DATA1**

**INT 21H**
INT 16h Keyboard Services

- Checking a key press, we use INT 16h function AH = 01
  
  ```
  MOV AH, 01
  INT 16h
  ```

- Upon return, ZF = 0 if there is a key press; ZF = 1 if there is no key press
- Which key is pressed?
- To do that, INT 16h function can be used immediately after the call to INT 16h function AH=01
  
  ```
  MOV AH, 0
  INT 16h
  ```

- Upon return, AL contains the ASCII character of the pressed key
Example INT 16 – 00

- BIOS Level Keyboard Input (more direct)
- Suppose F1 pressed (Scan Code 3BH). AH contains the scan code and AL contains the ASCII code (0).
Example. The PC Typewriter

- Write an 80x86 program to input keystrokes from the PC’s keyboard and display the characters on the system monitor. Pressing any of the function keys F1-F10 should cause the program to end.

- Algorithm:
  1. Get the code for the key pressed
  2. If this code is ASCII, display the key pressed on the monitor and continue
  3. Quit when a non-ASCII key is pressed

- INT 16, BIOS service 0 – Read next keyboard character
  - Returns 0 in AL for non-ASCII characters or the character is simply stored in AL

- To display the character, we use INT 10, BIOS service 0E- write character in teletype mode. AL should hold the character to be displayed.

- INT 20 for program termination
Example

MOV DX, OFFSET MES
MOV AH,09h
INT 21h ; to output the characters starting from the offset

AGAIN: MOV AH,0h
INT 16h; to check the keyboard
CMP AL,00h
JZ QUIT ; check the value of the input data
MOV AH, 0Eh
INT 10h; echo the character to output
JMP AGAIN

QUIT: INT 20h

MES DB ‘type any letter, number or punctuation key’
DB ‘any F1 to F10 to end the program”
DB 0d,0a,0a,’$’
Data Transfer Instructions - MOV

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Meaning</th>
<th>Format</th>
<th>Operation</th>
<th>Flags Affected</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOV</td>
<td>Move</td>
<td>MOV D, S</td>
<td>(S) → (D)</td>
<td>None</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Destination</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory</td>
<td>Accumulator</td>
</tr>
<tr>
<td>Accumulator</td>
<td>Memory</td>
</tr>
<tr>
<td>Register</td>
<td>Register</td>
</tr>
<tr>
<td>Register</td>
<td>Memory</td>
</tr>
<tr>
<td>Memory</td>
<td>Register</td>
</tr>
<tr>
<td>Register</td>
<td>Immediate</td>
</tr>
<tr>
<td>Memory</td>
<td>Immediate</td>
</tr>
<tr>
<td>Seg reg</td>
<td>Reg16</td>
</tr>
<tr>
<td>Seg reg</td>
<td>Mem16</td>
</tr>
<tr>
<td>Reg 16</td>
<td>Seg reg</td>
</tr>
<tr>
<td>Memory</td>
<td>Seg reg</td>
</tr>
</tbody>
</table>

Seg immediate & Memory to memory are not allowed
# Data Transfer Instructions - XCHG

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Meaning</th>
<th>Format</th>
<th>Operation</th>
<th>Flags Affected</th>
</tr>
</thead>
<tbody>
<tr>
<td>XCHG</td>
<td>Exchange</td>
<td>XCHG D,S</td>
<td>(Dest) ↔ (Source)</td>
<td>None</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Destination</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reg16</td>
<td>Reg16</td>
</tr>
<tr>
<td>Memory</td>
<td>Register</td>
</tr>
<tr>
<td>Register</td>
<td>Register</td>
</tr>
<tr>
<td>Register</td>
<td>Memory</td>
</tr>
</tbody>
</table>

Example: XCHG [1234h], BX
### Data Transfer Instructions – LEA, LDS, LES

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Meaning</th>
<th>Format</th>
<th>Operation</th>
<th>Flags Affected</th>
</tr>
</thead>
<tbody>
<tr>
<td>LEA</td>
<td>Load Effective Address</td>
<td>LEA Reg16,EA</td>
<td>EA →(Reg16)</td>
<td>None</td>
</tr>
<tr>
<td>LDS</td>
<td>Load Register and DS</td>
<td>LDS Reg16, MEM32</td>
<td>(Mem32) → (Reg16)</td>
<td>None</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(Mem32 + 2) → (DS)</td>
<td></td>
</tr>
<tr>
<td>LES</td>
<td>Load Register and ES</td>
<td>LES Reg16, MEM32</td>
<td>(Mem32) → (Reg16)</td>
<td>None</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(Mem32 + 2) → (ES)</td>
<td></td>
</tr>
</tbody>
</table>
Examples for LEA, LDS, LES

DATA X DW 1000H
DATA Y DW 5000H

.CODE
LEA SI, DATA X
MOV DI, OFFSET DATA Y; THIS IS MORE EFFICIENT
LEA BX,[DI]; IS THE SAME AS…
MOV BX, DI; THIS JUST TAKES LESS CYCLES.
LEA BX, DI; INVALID!

LDS BX, [DI];

BX  127A
DI  1000
DS  3000

7A
12
00
30
11000
11001
11002
11003
## Arithmetic Instructions – ADD, ADC, INC, AAA, DAA

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Meaning</th>
<th>Format</th>
<th>Operation</th>
<th>Flags Affected</th>
</tr>
</thead>
</table>
| ADD | Addition | ADD D, S | (S) + (D) \(\rightarrow\) (D)  
Carry \(\rightarrow\) (CF) | All |
| ADC | Add with carry | ADC D, S | (S) + (D) + (CF) \(\rightarrow\) (D)  
Carry \(\rightarrow\) (CF) | All |
| INC | Increment by one | INC D | (D) + 1 \(\rightarrow\) (D) | All but CY |
| AAA | ASCII adjust after addition of two ASCII numbers | AAA | Operate on AL (value in ASCII number) for the source & adjust for BCD to AX | AF,CY |
| DAA | Decimal adjust after addition | DAA | Adjusts AL for decimal | All |
Examples

Ex. 1 ADD AX, 2
    ADC AX, 2

Ex. 2 INC BX
    INC word ptr [BX]

Ex. 3 ASCII CODE 0-9 = 30h → 39h
    MOV AX, 38H ;(ASCII code for number 8)
    ADD AL, 39H ;(ASCII code for number 9)
    AAA; used for addition  AX has → 0107
    ADD AX, 3030H; change answer to ASCII if you needed

Ex. 4 AL contains 25 (packed BCD)
    BL contains 56 (packed BCD)
    ADD AL, BL
    DAA

                  25
    + ----------

                  56


                  7B → 81
Write a program that adds two multiword numbers:

.MOdel SMALL

.STACK 64

.DATA

DATA1 DQ 548F9963CE7h; allocate 8 bytes

ORG 0010h

DATA2 DQ 3FCD4FA23B8Dh; allocate 8 bytes

ORG 0020h

DATA3 DQ ?
.CODE

MAIN PROC FAR
MOV AX,@DATA; receive the starting address for DATA
MOV DS,AX
CLC; clears carry
MOV SI,OFFSET DATA1; LEA for DATA1
MOV DI,OFFSET DATA2; LEA for DATA2
MOV BX,OFFSET DATA3; LEA for DATA3
MOV CX,04h
BACK: MOV AX,[SI]
    ADC AX,[DI]; add with carry to AX
    MOV [BX],AX
    ADD SI,2h
    ADD DI,2h
    ADD BX,2h
LOOP BACK; decrement CX automatically until zero
MOV AH,4Ch
INT 21h; halt

MAIN ENDP
END MAIN
Example Cont’d

After 1st word addition
# Arithmetic Instructions – SUB, SBB, DEC, AAS, DAS, NEG

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Meaning</th>
<th>Format</th>
<th>Operation</th>
<th>Flags Affected</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SUB</strong></td>
<td>Subtract</td>
<td>SUB D, S</td>
<td>(D) - (S) \rightarrow (D) &lt;br&gt;Borrow \rightarrow (CF)</td>
<td>All</td>
</tr>
<tr>
<td><strong>SBB</strong></td>
<td>Subtract with borrow</td>
<td>SBB D, S</td>
<td>(D) - (S) - (CF) \rightarrow (D)</td>
<td>All</td>
</tr>
<tr>
<td><strong>DEC</strong></td>
<td>Decrement by one</td>
<td>DEC D</td>
<td>(D) - 1 \rightarrow (D)</td>
<td>All but CY</td>
</tr>
<tr>
<td><strong>NEG</strong></td>
<td>Negate</td>
<td>NEG D</td>
<td>2’s complement operation</td>
<td>All</td>
</tr>
<tr>
<td><strong>DAS</strong></td>
<td>Decimal adjust for subtraction</td>
<td>DAS</td>
<td>(convert the result in AL to packed decimal format)</td>
<td>All</td>
</tr>
<tr>
<td><strong>AAS</strong></td>
<td>ASCII adjust after subtraction</td>
<td>AAS</td>
<td>(convert the result in AX to packed decimal format) 37-38 -&gt; 09</td>
<td>CY, AC</td>
</tr>
</tbody>
</table>
Examples with DAS and AAS

MOV BL, 28H
MOV AL, 83H
SUB AL, BL;    AL=5BH
DAS;    adjusted as AL=55H

MOV AX, 38H
SUB AL, 39H; AX=00FF
AAS; AX=FF09 ten’s complement of –1
OR AL, 30H; AL = 39
Example on SBB

• 32-bit subtraction of two 32 bit numbers X and Y that are stored in the memory as
  – Y = (DS:103h)(DS:102h)(DS:101h)(DS:100h)
• The result X - Y to be stored where X is saved in the memory

MOV SI, 200h
MOV DI, 100h
MOV AX, [SI]
SUB AX, [DI]
MOV [SI], AX    ;save the LS word of result
MOV AX, [SI] +2 ; carry is generated from the first sub?
SBB AX, [DI] +2 ; then subtract CY this time!
MOV [SI] +2, AX

Ex. 12 34 56 78 – 23 45 67 89 = EE EE EE EF
## Multiplication and Division

<table>
<thead>
<tr>
<th>Multiplication (MUL or IMUL)</th>
<th>Multiplicand</th>
<th>Operand (Multiplier)</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Byte * Byte</td>
<td>AL</td>
<td>Register or memory</td>
<td>AX</td>
</tr>
<tr>
<td>Word * Word</td>
<td>AX</td>
<td>Register or memory</td>
<td>DX :AX</td>
</tr>
<tr>
<td>Dword * Dword</td>
<td>EAX</td>
<td>Register or Memory</td>
<td>EDX :EAX</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Division (DIV or IDIV)</th>
<th>Dividend</th>
<th>Operand (Divisor)</th>
<th>Quotient : Remainder</th>
</tr>
</thead>
<tbody>
<tr>
<td>Word / Byte</td>
<td>AX</td>
<td>Register or memory</td>
<td>AL : AH</td>
</tr>
<tr>
<td>Dword / Word</td>
<td>DX:AX</td>
<td>Register or memory</td>
<td>AX : DX</td>
</tr>
<tr>
<td>Qword / Dword</td>
<td>EDX: EAX</td>
<td>Register or Memory</td>
<td>EAX : EDX</td>
</tr>
</tbody>
</table>
Unsigned Multiplication Exercise

DATAX DB 4EH
DATAY DW 12C3H
RESULT DQ DUP (?)

Find: Result = Datax * Datay

; one possible solution
XOR AX,AX ; or MOV AX, 0000H
LEA SI, DATAX
MOV AL,[SI]
MUL DATAY
LEA DI, RESULT
MOV [DI],AX
MOV [DI+2],DX
AAM, AAD, CBW, CWD

- **AAM**: Adjust AX after multiply
  - MOV AL,07 ; MOV CL,09; unpacked numbers
  - MUL CL ; second unpacked number multiplied with AL
  - AAM ; AX unpacked decimal representation: 06 03

- **AAD**: Adjust AX *(before)* for divide
  - AX converted from two unpacked BCD into Binary before division
  - For ex: MOV AX,0208h; dividend AAD forms: AX=001C

- **CBW** instruction. Division instructions can also be used to divide an 8 bit dividend in AL by an 8 bit divisor.
  - In order to do so, the sign of the dividend must be extended to to fill the AX register
  - AH is filled with zeros if AL is positive
  - AH is filled with ones if the number in AL is negative
  - Automatically done by executing the CBW (convert byte to word) instruction. Simply extends the sign bit into higher byte.

- **CWD** *(convert word to double word)*
  - Ex. MOV AL, 0A1h
  - CBW; convert byte to word
  - CWD; convert word to double word (push sign into DX)
Example

• Write a program that calculates the average of five temperatures and writes the result in AX

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATA</td>
<td>DB +13,-10,+19,+14,-18 ;0d,f6,13,0e,ee</td>
</tr>
<tr>
<td>MOV</td>
<td>CX,5 ;LOAD COUNTER</td>
</tr>
<tr>
<td>SUB</td>
<td>BX, BX ;CLEAR BX, USED AS ACCUMULATOR</td>
</tr>
<tr>
<td>MOV</td>
<td>SI, OFFSET DATA ;SET UP POINTER</td>
</tr>
<tr>
<td>BACK: MOV</td>
<td>AL,[SI] ;MOVE BYTE INTO AL</td>
</tr>
<tr>
<td>CBW</td>
<td>;SIGN EXTEND INTO AX</td>
</tr>
<tr>
<td>ADD</td>
<td>BX, AX ;ADD TO BX</td>
</tr>
<tr>
<td>INC</td>
<td>SI ;INCREMENT POINTER</td>
</tr>
<tr>
<td>DEC</td>
<td>CX ;DECREMENT COUNTER</td>
</tr>
<tr>
<td>JNZ</td>
<td>BACK</td>
</tr>
<tr>
<td>mov ax,bx</td>
<td>;LOOP IF NOT FINISHED</td>
</tr>
<tr>
<td>MOV</td>
<td>CL,5 ;MOVE COUNT TO AL</td>
</tr>
<tr>
<td>DIV</td>
<td>CL ;FIND THE AVERAGE</td>
</tr>
</tbody>
</table>
Logical Instructions [reset CY and reset OF]

- **AND**
  - Uses any addressing mode except memory-to-memory and segment registers. Places the result in the first operator.
  - Especially used in clearing certain bits (masking)
    - \( \text{xxxx xxxx AND 0000 1111} = 0000 \text{ xxxx} \) (clear the first four bits)
  - Examples: AND BL, 0FH; AND AL, [345H]

- **OR**
  - Used in setting certain bits
    - \( \text{xxxx xxxx OR 0000 1111} = \text{xxxx 1111} \)

- **XOR**
  - Used in inverting bits
    - \( \text{xxxx xxxx XOR 0000 1111} = \text{xxxx yyyy} \)

- **Ex.** Clear bits 0 and 1, set bits 6 and 7, invert bit 5
  
  \[
  \begin{array}{l}
  \text{AND CX, OFCH} & 1111 \text{ 1100} \\
  \text{OR CX, 0C0H} & 1100 \text{ 0000} \\
  \text{XOR CX, 020H} & 0010 \text{ 0000} \\
  \text{XOR AX,.AX} \\
  \end{array}
  \]
Turn the CAPS LOCK on

```assembly
CAUTION

push ds ; save the current ds
mov ax,40h ; new ds at BIOS
mov ds,ax
mov bx,17h ; keyboard flag byte
xor byte ptr[ bx ], 01000000b ; now you altered CAPS
pop ds
MOV Ah, 4CH
INT 21H
```
TEST

- TEST instruction performs the AND operation but it does not change
  the destination operand as in AND but only the flags register.
- Similar to CMP bit it tests a single bit or occasionally multiple bits.
- **Ex.** TEST DL, DH ; TEST AX, 56

  TEST AL, 1 ; test right bit
  JNZ RIGHT ; if set
  TEST AL, 128 ; test left bit
  JNZ LEFT ; if set