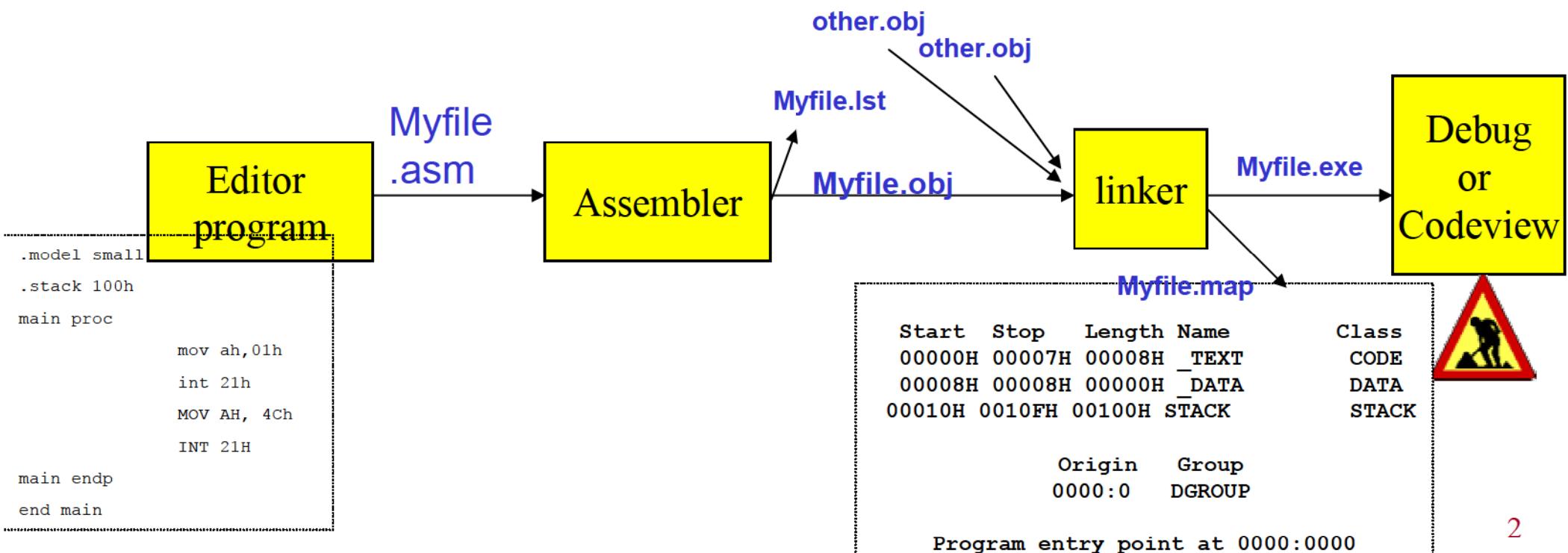


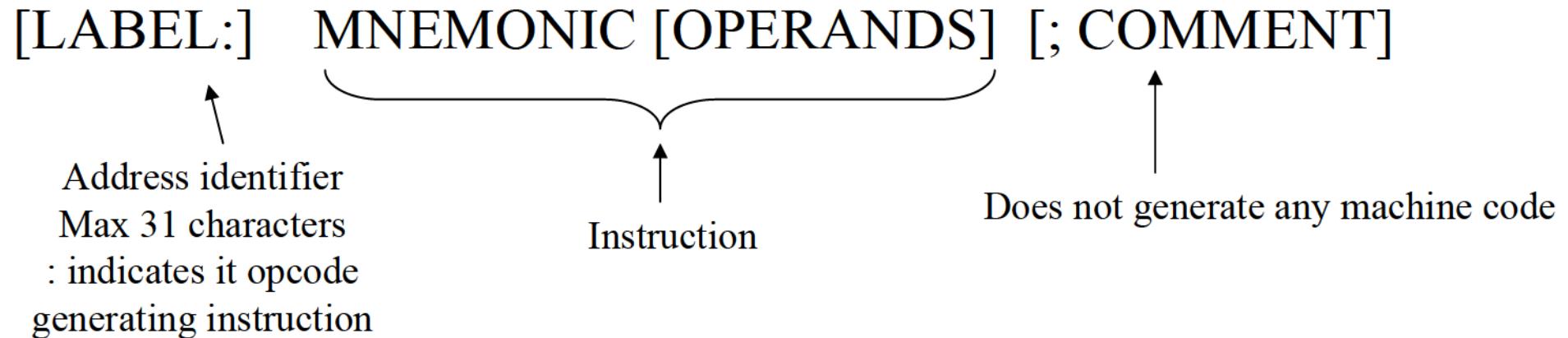
Assemble, Link and Run a Program

- Steps in creating an executable Assembly Language Program

Step	Input	Program	Output
1. Editing	Usually Keyboard	Editor (Text word editors etc.)	Myfile.asm
2. Assemble	Myfile.asm	MASM	Myfile.obj
3. Link	Myfile.obj	LINK	Myfile.exe



Instructions



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

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 \
 ,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:

.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

DATA1	DB	25
DATA2	DB	10001001b
DATA3	DB	12h
		ORG 0010h ;indicates distance ;from initial location
DATA4	DB	"2591"
		ORG 0018h
DATA5	DB	?

This is how data is initialized in the data segment

0000	19
0001	89
0002	12
0010	32 35 39 31
0018	00

DB DW DD

.data

```
MESSAGE2 DB '1234567'
```

; how it looks like in
memory

31	32	33	34	35	36	37
----	----	----	----	----	----	----

```
MESSAGE3 DW 6667H
```

67	66
----	----

```
data1 db 1,2,3
```

1	2	3
---	---	---

```
db 45h
```

45

```
db 'a'
```

61

```
db 11110000b
```

F0

```
data2 dw 12,13
```

0C	00	0D	00
----	----	----	----

```
dw 2345h
```

45	23
----	----

```
dd 300h
```

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?

MOV AL,BVAL

MOV DL,[BX]

SUB [BX],2

MOV CL,WVAL

ADD AL,BVAL+1

MOV AL,BVAL

MOV DL,[BX]

SUB [BX],byte ptr 2

MOV CL,byte ptr WVAL

ADD AL,BVAL+1

.data

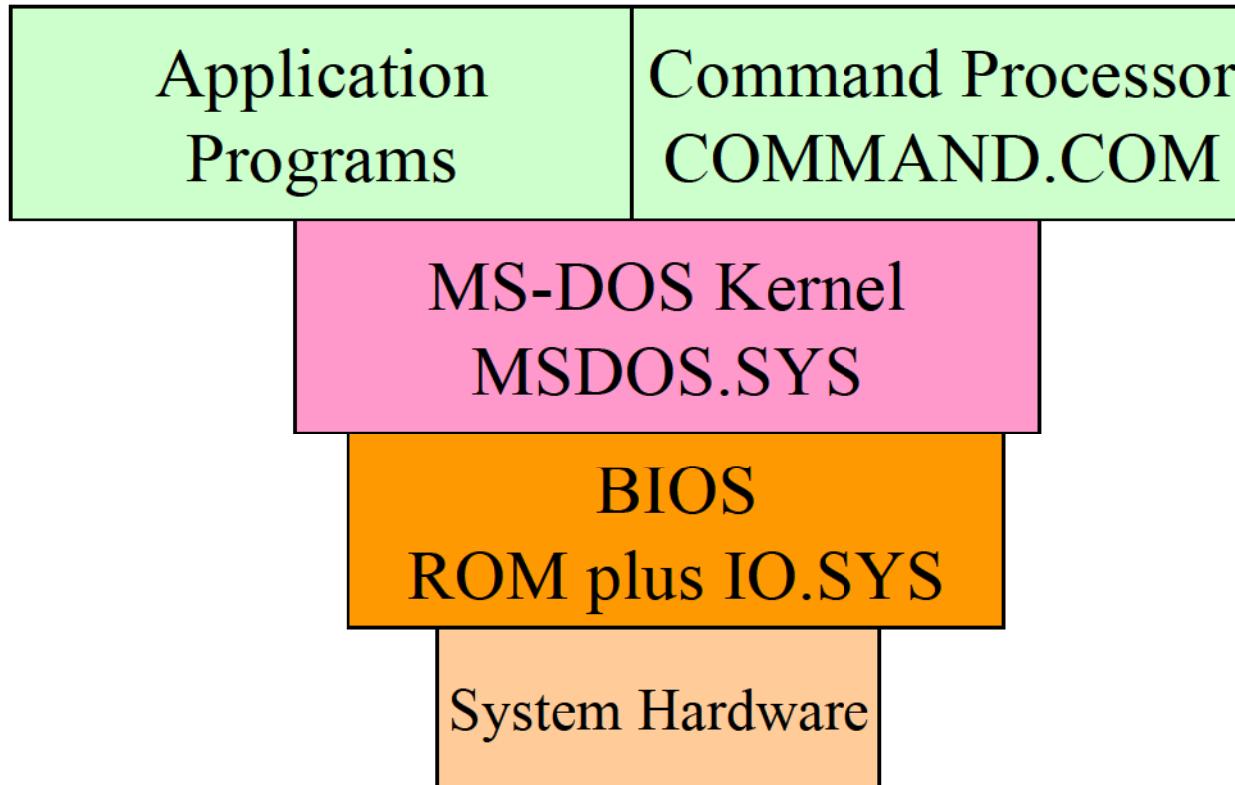
BVAL DB 10H,20H

WVAL DW 1000H

Simple Assembly Language Program

```
.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

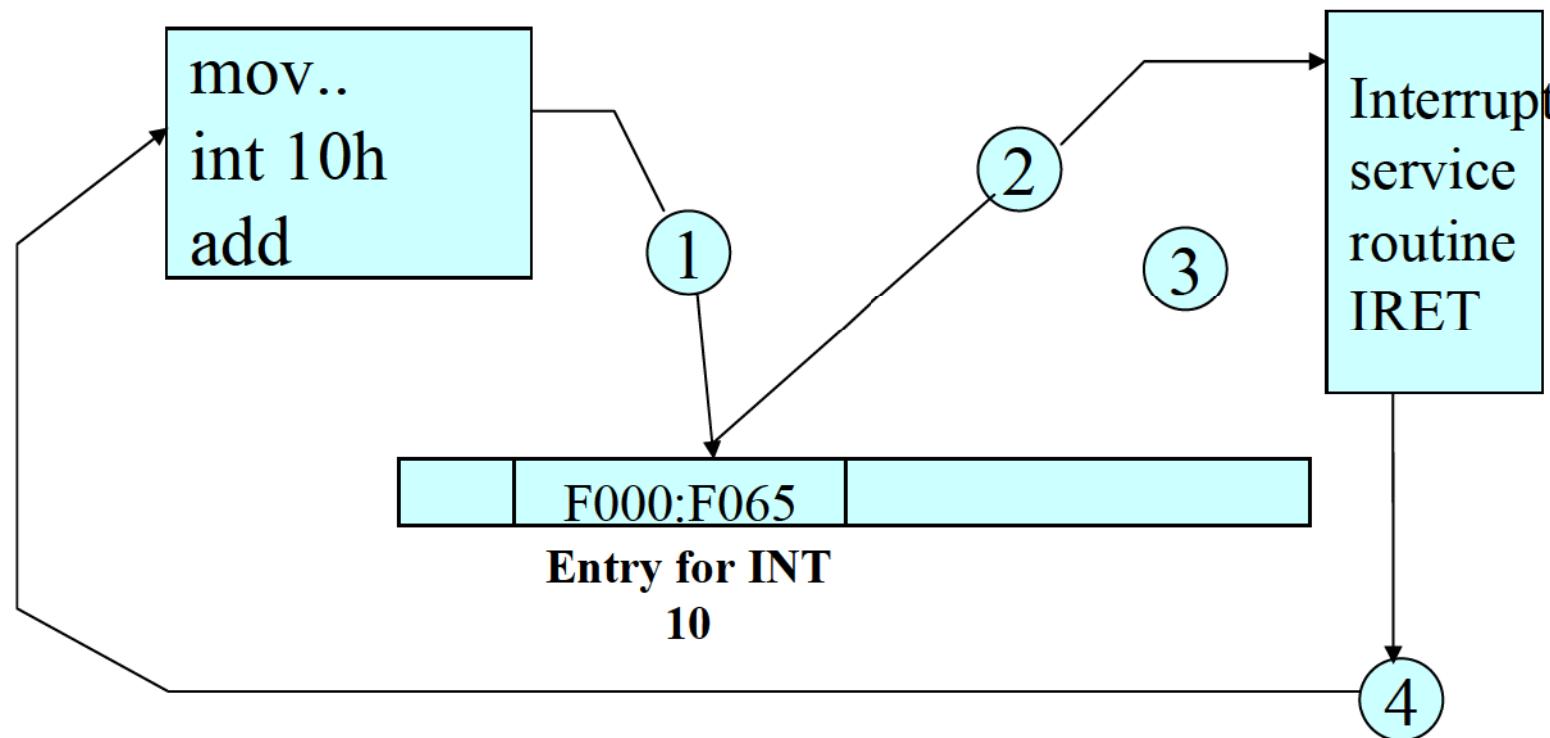
- An interrupt is an event that causes the processor to suspend its present task and transfer control to a new program called the interrupt service routine (**ISR**)
- There are three sources of interrupts
 - Processor interrupts
 - Hardware interrupts generated by a special chip, for ex: 8259 Interrupt 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 $0 \rightarrow FFh$. Before INT is executed **AH** usually contains a function number that identifies the subroutine.

80x86 Interrupts

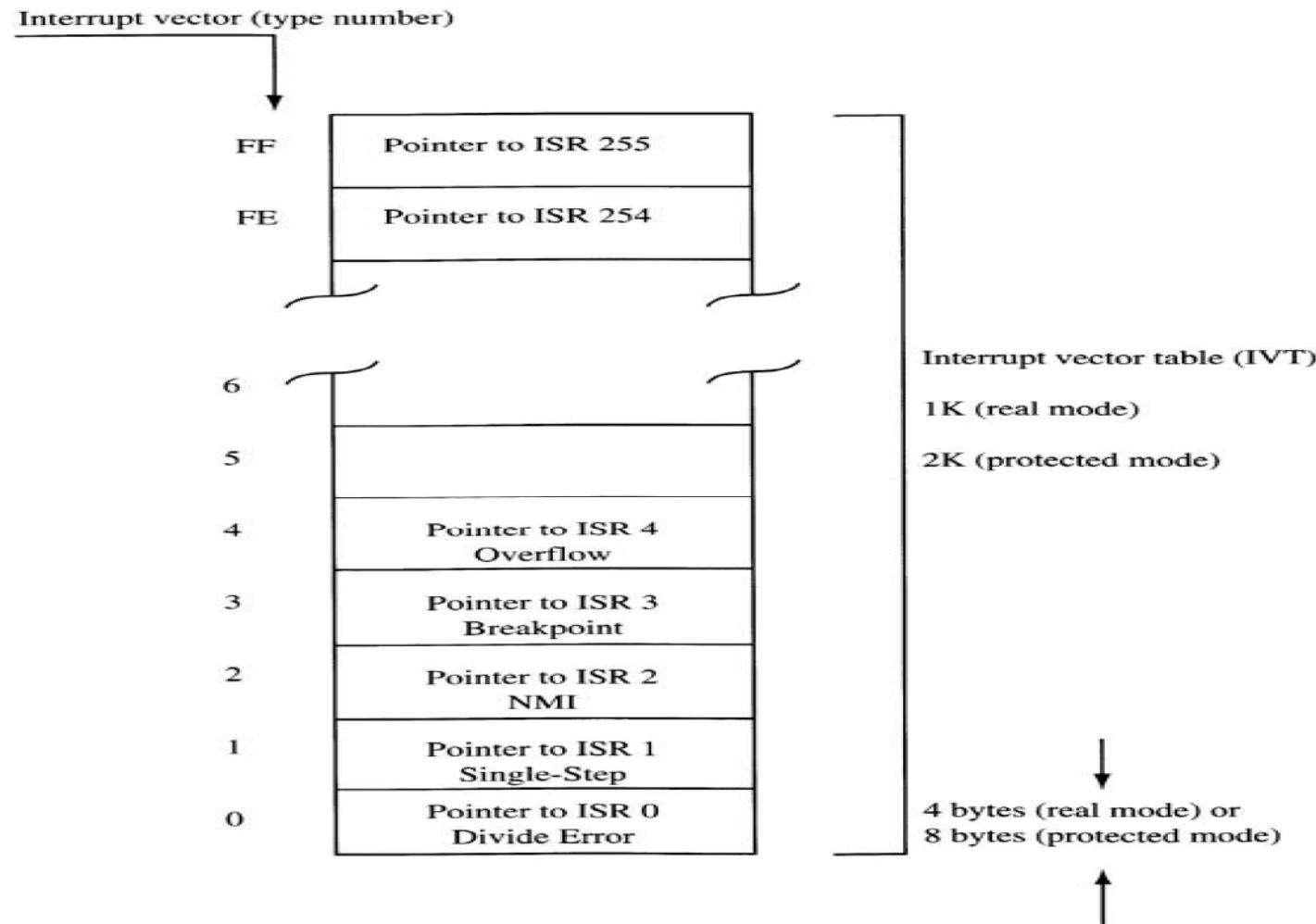
- Each interrupt must supply a **type number** which is used by the processor as a pointer to an interrupt vector table (**IVT**) to determine the address of that interrupt's service routine
- **Interrupt Vector Table:** CPU processes an interrupt instruction using the interrupt vector table (This table 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.



Interrupt Vector Table



Processor	Pointer Size	IVT Location
Real Mode	4 bytes	Address 00000000–000003FF
Protected Mode	8 bytes	Anywhere in Physical Memory

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

The screenshot shows a DOS terminal window with a yellow border. On the left, assembly code is displayed:

```
.model small
.stack 100h
.data
    ; ORG 0010H;
    ; DATA1
.code
main proc
    mov ah,02h
    ; 
    mov al,05h
    mov dl,39h
    mov dh,02h
    mov bh,0h ; r
    int 10h
    MOU AH, 4Ch
    INT 21H
main endp
end main
```

On the right, a file listing is shown with a yellow circle highlighting the directory path "C:\Irvine>". A yellow box labeled "New Cursor Location" is overlaid on the listing area.

C:\Irvine>	<DIR>	5-15-02	2:24a	ch01		
CH08	<DIR>	05-15-02	2:24a	ch08		
CH09	<DIR>	05-15-02	2:24a	ch09		
C:\Irvine>	<DIR>					
CH11	<DIR>					
CH12	<DIR>					
CH13	<DIR>	05-15-02	2:24a	ch13		
CH14	<DIR>	05-15-02	2:24a	ch14		
CH15	<DIR>	05-15-02	2:24a	ch15		
HELLO	OBJ	467	02-23-03	7:54p	HELLO.obj	
HELLO	MAP	281	02-23-03	7:54p	HELLO.MAP	
EARTH	OBJ	1,192	02-23-03	7:54p	EARTH.EXE	
EARTH	MAP	427	03-02-03	3:21p	EARTH.obj	
EARTH	EXE	281	03-02-03	3:21p	EARTH.MAP	
CURRENT	STS	1,176	03-02-03	3:21p	EARTH.EXE	
CLRFILE	CV4	737	03-02-03	1:16p	CURRENT.STS	
EARTH100	OBJ	203	03-02-03	1:16p	CLRFILE.CV4	
EARTH100	MAP	415	03-02-03	3:59p	EARTH100.obj	
EARTH100	EXE	281	03-02-03	3:59p	EARTH100.MAP	
		1,164	03-02-03	3:59p	EARTH100.EXE	
		24 file(s)		187,814 bytes		
		16 dir(s)		4,469.53 MB free		

At the bottom, the command "C:\Irvine>earth100" is entered, along with F1 Help, F2 Save, and F3.

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 page1  
INT 10H  
; below will switch to video page 0  
MOV AH, 05h  
MOV AL, 00H; switch to video page0  
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

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

Cursor Locations

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

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

Example Int10 06

Created with HyperSnap-DX 5
To avoid this stamp, buy a license at
<http://www.hyperionics.com>

A

Windows

C:\WINDOWS\DESKTOP\EARTH.ASM

```
.model small
.stack 100h
.data
    ; ORG 0010H; offset address
    ; DATA1          DB 6,?,6 DUP(00)
.code
main proc
    mov ah,06h
    mov al,05h
    mov ch,0h
    mov cl,0h
    mov dh,24h
    mov dl,01h
    mov bh,7h
    int 10h
    MOU AH, 4Ch
    INT 21H
main endp
end main
```

10:18

F1 Help F2 Save F3 Open Alt+F3 Close F5 Zoom F6 Next F10 Menu

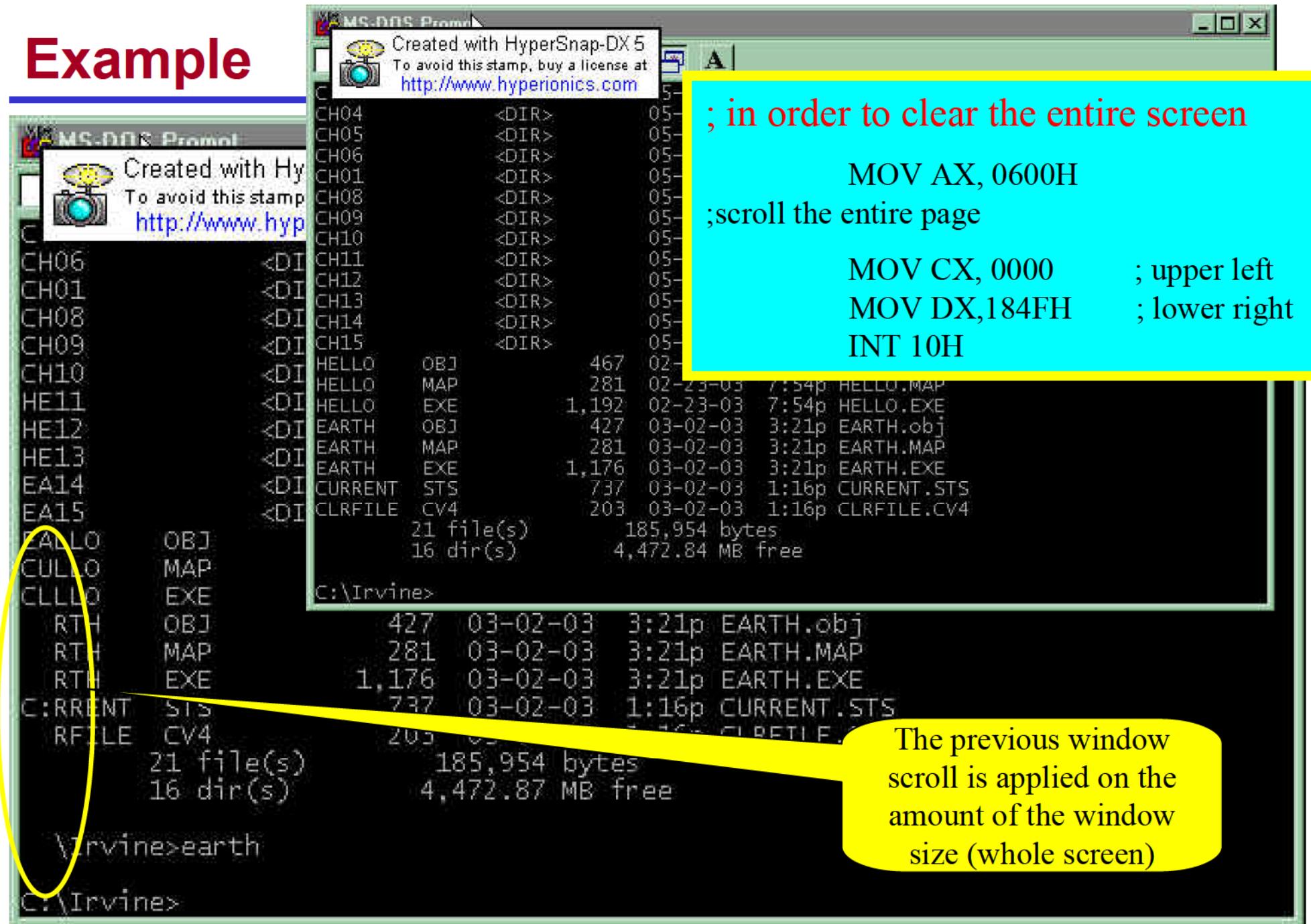
Init reg AH for the program

Define the line of the “window” size to scroll

Define the “the window”

Halt the program

Example



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

The screenshot shows a DOS terminal window with a blue background. On the left, assembly code is displayed:

```
.model small
.stack 100h
.data
    ; ORG 0010H; offset address
    ; DATA1           DB 6,?,6 DUP(00)
.code
main proc
    mov ah,09h
    mov al,0Ah ;interpreted as white circle on black background.
    mov bh,0
    mov bl,87h; blinking attribute
    mov cx,10h
    int 10h
    MOU AH, 4Ch
    INT 21H
main endp
end main
```

At the bottom of the terminal window, there is a status bar with the text "10:69" and some icons. Below the terminal window, a file list is shown in a separate window:

	EART1050.OBJ	403	03-02-03	4:15p	E
E	Created with HyperSnap-DX 5		3-02-03	4:16p	E
E	To avoid this stamp, buy a license at		3-02-03	4:16p	E
E	http://www.hyperionics.com		3-02-03	4:33p	E
E			3-02-03	4:33p	E
E	EART1090.EXE	1,172	03-02-03	4:33p	E
	36 file(s)			195,185 bytes	
	16 dir(s)			4,445.30 MB free	

The command "C:\Irvine>eart1090" is entered at the prompt, and the output "oooooooooooo" is displayed below it.

Int 10 – 0E PRINT SINGLE CHARACTER

The screenshot shows a Microsoft Windows desktop with several windows open:

- A Notepad window titled "C:\WINDOWS\Desktop\EART10E0.ASM" containing assembly language code for printing a character.
- A terminal window at the bottom showing the command prompt and file navigation.
- A taskbar at the bottom with icons for various applications.
- A vertical list of files on the right side of the desktop.

The assembly code in the Notepad window is:

```
.model small
.stack 100h
.data
    ; ORG 0010H; offset adress
    ; DATA1           DB 6,?,6 DUP(00)
.code
main proc
    mov ah,0Eh
    mov al,10h_
    mov bh,0h
    int 10h
    MOU AH, 4Ch
    INT 21H
main endp
end main
```

A yellow callout box highlights the instruction `INT 10h` with the text:

Write out a single character
(Also stored in AL)

The terminal window at the bottom shows the following output:

```
C:\Irvine>eart10e0
C:\Irvine>
```

A yellow circle highlights the command prompt in the terminal window.

The taskbar icons include:

- File Explorer
- Recycle Bin
- Task View
- Start button
- Calculator
- File Manager
- Notepad
- Terminal (highlighted with a yellow circle)
- PowerShell
- Windows Terminal
- FileZilla
- Visual Studio Code
- GitHub Desktop
- OneDrive
- Microsoft Edge
- Windows File Explorer
- Windows Terminal
- FileZilla
- Visual Studio Code
- GitHub Desktop
- OneDrive
- Microsoft Edge

The vertical list of files on the right side of the desktop includes:

- ART1090.MAP
- ART1090.EXE
- ART10E0.obj
- ART10E0.MAP
- ART10E0.EXE

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.

The screenshot shows a DOS terminal window with a blue background. On the left, assembly code is displayed:

```
[1] C:\>
.model small
.stack 100h
.data
    ; ORG 0010H; offset
    ; DATA1           DB
.code
main proc
    mov ah,01h
    int 21h
    MOU AH, 4Ch
    INT 21H
main endp
end main
```

On the right, the file list and the output of the program execution are shown:

File	Type	Size	Date	Time
EART21	MAP	281	03-02-03	5:08
EART21	EXE	1,128	03-02-03	5:08
		42 file(s)		198,829 bytes
		16 dir(s)		4,429.55 MB free

C:\Irvine>eart21
A
C:\Irvine>

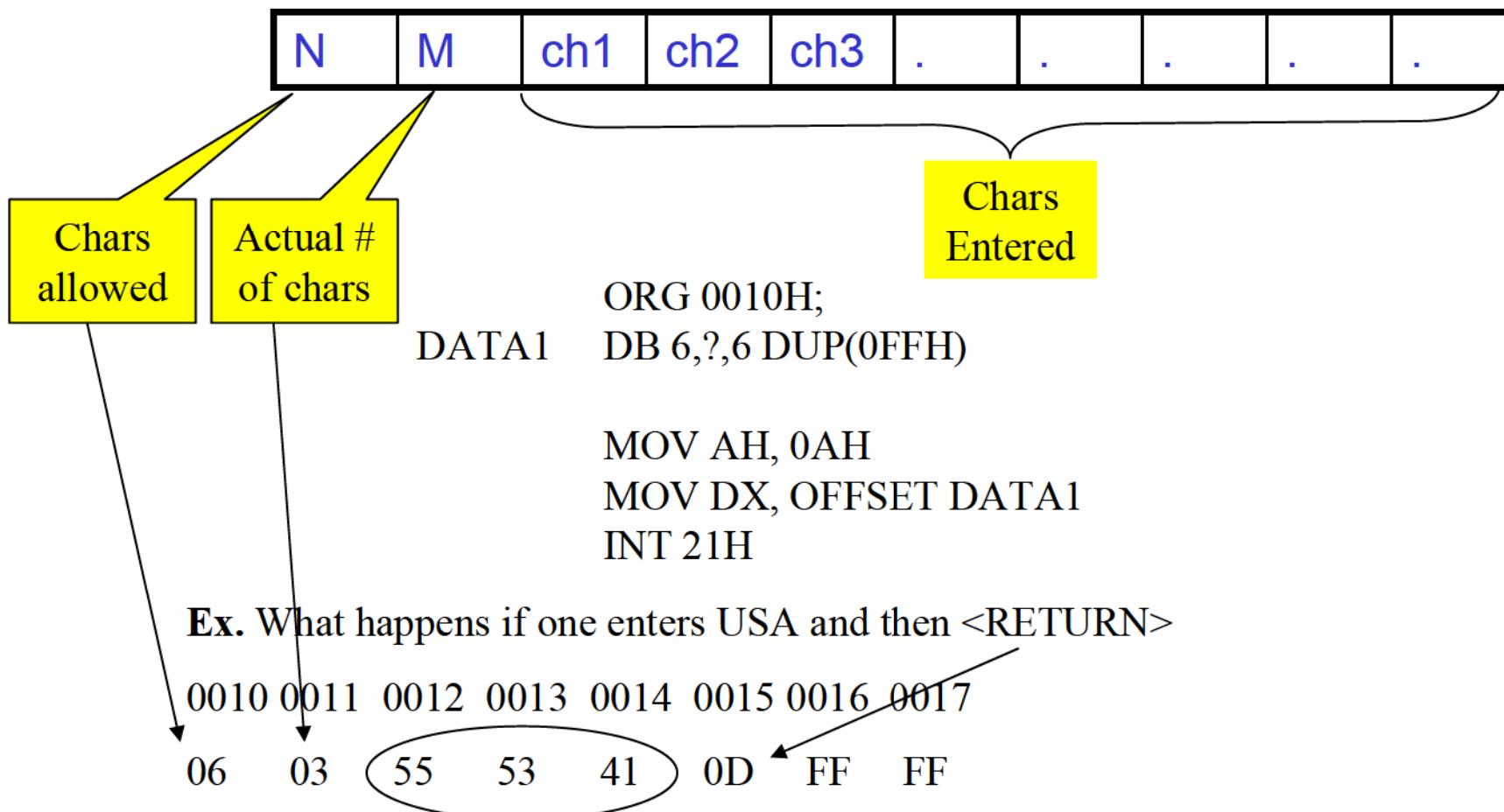
At the bottom, the time is 10:15 and the menu bar includes F1 Help, F2 Save, F3 Open, Alt-F3 Close, F5 Zoom, F6 Next, and F10 Menu.

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



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).

The screenshot shows a debugger window with the following details:

- Registers:** AX = 3B00, BX = 0000, CX = 0000, DX = 0000, SP = 0100, BP = 0000, SI = 0000, DI = 0000, DS = 1D4B, ES = 1D4B, SS = 1D5C, CS = 1D5B, IP = 0004, FL = 3206.
- Stack:** NV UP EI PL NZ NA PE NC
- Registers:** [7] reg
- Memory Dump:** [5] memory1 b DS:0
- Assembly Code:** source1 CS:IP EART1610.asm
10: mov ah, 10h
1D5B:0000 B410 MOV AH, 10
11: int 16h
1D5B:0002 CD16 INT 16
12: MOV AH, 4Ch
1D5B:0004 B44C MOV AH, 4C
13: INT 21H
1D5B:0006 CD21 INT 21
14: main endp
15:
16: end main
- Command Line:** [9] command
CV1053 Warning: TOOLS.INI not found
>
- Keyboard Shortcuts:** <F8=Trace> <F10=Step> <F5=Go> <F3=S1 Fmt> <Sh+F3=M1 Fmt>
- Status Bar:** DEC

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

Mnemonic	Meaning	Format	Operation	Flags Affected
MOV	Move	MOV D, S	(S) →(D)	None

Destination	Source
Memory	Accumulator
Accumulator	Memory
Register	Register
Register	Memory
Memory	Register
Register	Immediate
Memory	Immediate
Seg reg	Reg16
Seg reg	Mem16
Reg 16	Seg reg
Memory	Seg reg

Seg immediate
& Memory to
memory
are not allowed

Data Transfer Instructions - XCHG

Mnemonic	Meaning	Format	Operation	Flags Affected
XCHG	Exchange	XCHG D,S	(Dest) \leftrightarrow (Source)	None

Destination	Source
Reg16	Reg16
Memory	Register
Register	Register
Register	Memory

Example: XCHG [1234h], BX

Data Transfer Instructions – LEA, LDS, LES

Mnemonic	Meaning	Format	Operation	Flags Affected
LEA	Load Effective Address	LEA Reg16,EA	EA →(Reg16)	None
LDS	Load Register and DS	LDS Reg16, MEM32	(Mem32) → (Reg16) (Mem32 + 2) → (DS)	None
LES	Load Register and ES	LES Reg16, MEM32	(Mem32) → (Reg16) (Mem32 + 2) → (ES)	None

Examples for LEA, LDS, LES

```
DATAX DW 1000H
```

```
DATAY DW 5000H
```

```
.CODE
```

```
LEA SI, DATAX
```

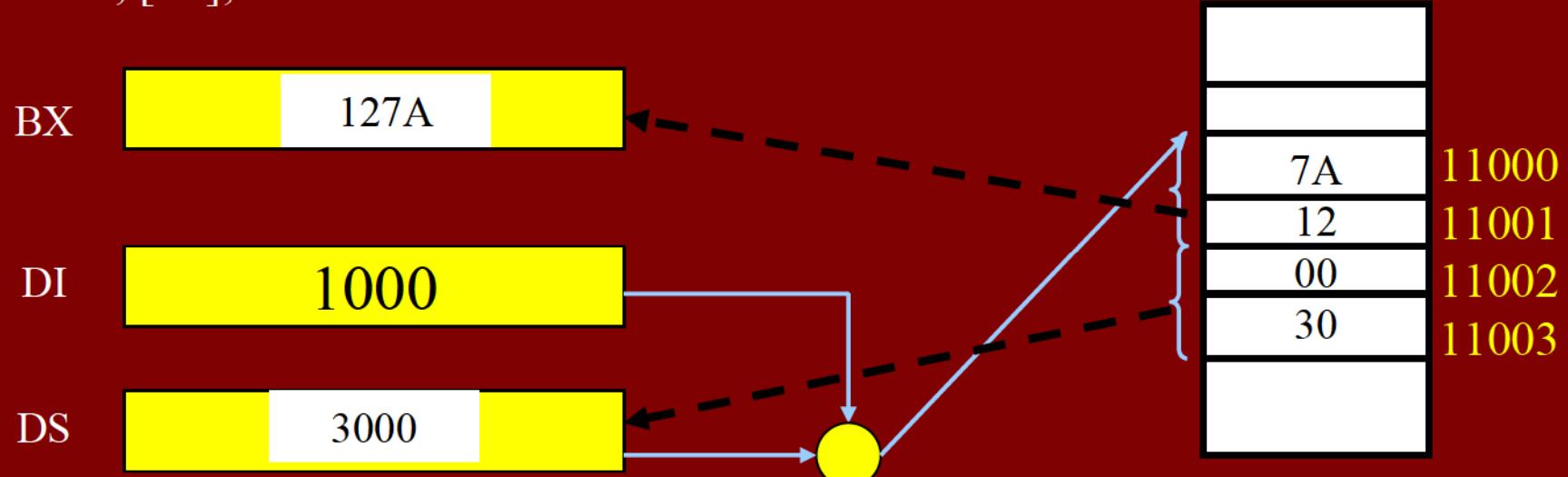
```
MOV DI, OFFSET DATAY; THIS IS MORE EFFICIENT
```

```
LEA BX,[DI]; IS THE SAME AS...
```

```
MOV BX,DI; THIS JUST TAKES LESS CYCLES.
```

```
LEA BX,DI; INVALID!
```

```
LDS BX, [DI];
```



Arithmetic Instructions – ADD, ADC, INC, AAA, DAA

Mnemonic	Meaning	Format	Operation	Flags Affected
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)

25	
56	
+ -----	
7B → 81	

ADD AL, BL
DAA

Example

Write a program that adds two multiword numbers:

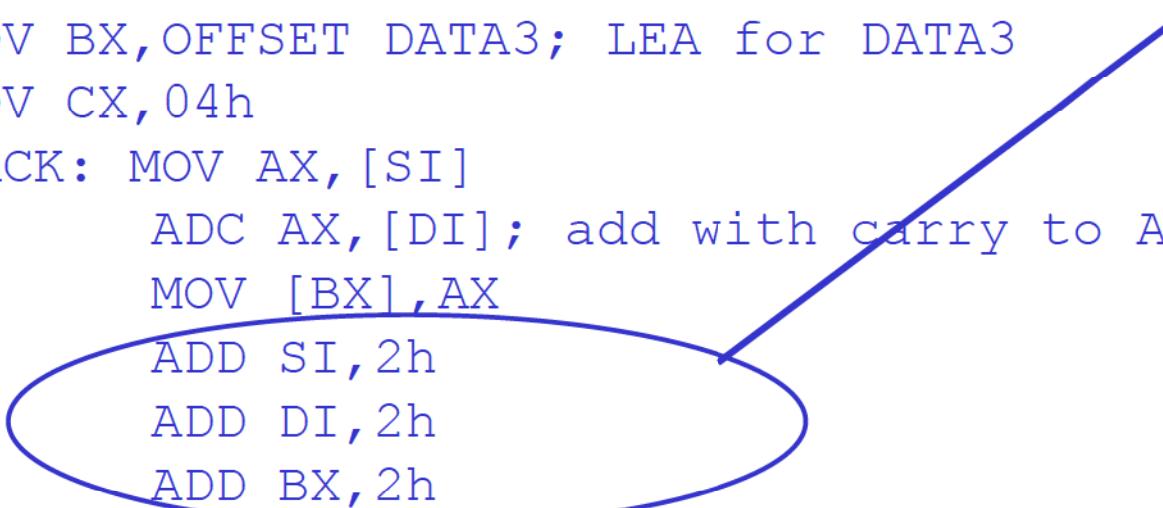
```
.MODEL SMALL  
.STACK 64  
.DATA  
    DATA1 DQ 548F9963CE7h; allocate 8 bytes  
ORG 0010h  
    DATA2 DQ 3FCD4FA23B8Dh; allocate 8 bytes  
ORG 0020h  
    DATA3 DQ ?
```

Example Cont'd

```
.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
```

```
INC SI
INC SI
INC DI
INC DI
INC BX
INC BX
```



Example Cont'd

Created with HyperSnap-DX 5
To avoid this stamp, buy a license at
<http://www.hyperionics.com>

A

Run Data Options Calls Windows Help

[3] source1 CS:IP HELLO4.asm

```
31:             BACK: MOV AX, [SI]
1D5B:001B 8B04      MOV      AX,WORD PTR [SI]
32:             ADC AX, [DI]; add with carry to AX
1D5B:001D 1305      ADC      AX,WORD PTR [DI]
33:             MOV [BX],AX
1D5B:001F 8907      MOV      WORD PTR [BX],AX
34:             ADD SI,2h
1D5B:0021 83C602      ADD      SI,02
35:             ADD DI,2h
1D5B:0024 83C702      ADD      DI,02
36:             ADD BX,2h
```

[4] source2 HELLO4.asm

[5] memory1 b 0x1D5E:0x0010

```
1D5E:0010 E7 3C 96 F9 48 05 00 00 00 00 00 00 00 00 00 00 T<û•H+.....
1D5E:001D 00 00 00 8D 3B A2 4F CD 3F 00 00 00 00 00 00 00 00 ...1;00=?....
```

[6] memory2 b 0x1D5E:0x0020

```
1D5E:0020 8D 3B A2 4F CD 3F 00 00 00 00 00 00 00 00 00 00 00 i;00=?.....
1D5E:002D 00 00 00 00 00 00 00 00 00 00 00 00 00 00 4E 42 .....NB
```

[7] reg

AX	= 7874
BX	= 0030
CX	= 0004
DX	= 0000
SP	= 0100
BP	= 0000
SI	= 0010
DI	= 0020
DS	= 1D5E
ES	= 1D4B
SS	= 1D62
CS	= 1D5B
IP	= 001F
FL	= B216

NV UP EI PL
NZ AC PE NC

ds:0030 0000

<F8=Trace> <F10=Step> <F5=Go> <F3=S1 Fmt> <Sh+F3=M2 Fmt>

DEC

After 1st word addition

Arithmetic Instructions – SUB, SBB, DEC, AAS, DAS, NEG

Mnemonic	Meaning	Format	Operation	Flags Affected
SUB	Subtract	SUB D, S	$(D) - (S) \rightarrow (D)$ Borrow $\rightarrow (CF)$	All
SBB	Subtract with borrow	SBB D, S	$(D) - (S) - (CF) \rightarrow (D)$	All
DEC	Decrement by one	DEC D	$(D) - 1 \rightarrow (D)$	All but CY
NEG	Negate	NEG D	2's complement operation	All
DAS	Decimal adjust for subtraction	DAS	(convert the result in AL to packed decimal format)	All
AAS	ASCII adjust after subtraction	AAS	(convert the result in AX to packed decimal format) 37-38 -> 09	CY, AC

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
 - $X = (\text{DS}:203h)(\text{DS}:202h)(\text{DS}:201h)(\text{DS}:200h)$
 - $Y = (\text{DS}:103h)(\text{DS}:102h)(\text{DS}:101h)(\text{DS}:100h)$
- 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

Multiplication (MUL or IMUL)	Multiplicand	Operand (Multiplier)	Result
Byte * Byte	AL	Register or memory	AX
Word * Word	AX	Register or memory	DX :AX
Dword * Dword	EAX	Register or Memory	EDX :EAX

Division (DIV or IDIV)	Dividend	Operand (Divisor)	Quotient : Remainder
Word / Byte	AX	Register or memory	AL : AH
Dword / Word	DX:AX	Register or memory	AX : DX
Qword / Dword	EDX: EAX	Register or Memory	EAX : EDX

Unsigned Multiplication Exercise

DATA ^X	DB	4EH
DATA ^Y	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

Ex. MOV BL,9
MOV AX,0702H
;convert to binary first
AAD; 00-99
DIV BL

- 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 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

DATA	DB	+13,-10,+19,+14,-18	;0d,f6,13,0e,ee
	MOV	CX,5	;LOAD COUNTER
	SUB	BX, BX	;CLEAR BX, USED AS ACCUMULATOR
	MOV	SI, OFFSET DATA	;SET UP POINTER
BACK:	MOV	AL,[SI]	;MOVE BYTE INTO AL
	CBW		;SIGN EXTEND INTO AX
	ADD	BX, AX	;ADD TO BX
	INC	SI	;INCREMENT POINTER
	DEC	CX	;DECREMENT COUNTER
	JNZ	BACK	;LOOP IF NOT FINISHED
	MOV	CL,5	;MOVE COUNT TO AL
	DIV	CL	;FIND THE AVERAGE

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)
 - xxxx xxxx **AND** 0000 1111 = 0000 xxxx (clear the first four bits)
 - Examples: AND BL, 0FH; AND AL, [345H]
- OR
 - Used in setting certain bits
 - xxxx xxxx **OR** 0000 1111 = xxxx 1111
- XOR
 - Used in inverting bits
 - xxxx xxxx **XOR** 0000 1111 = xxxx yyyy
- **Ex.** Clear bits 0 and 1, set bits 6 and 7, invert bit 5

AND CX, OFCH	1111 1100
OR CX, 0C0H	1100 0000
XOR CX, 020H	0010 0000
XOR AX.,AX	

Turn the CAPS LOCK on



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

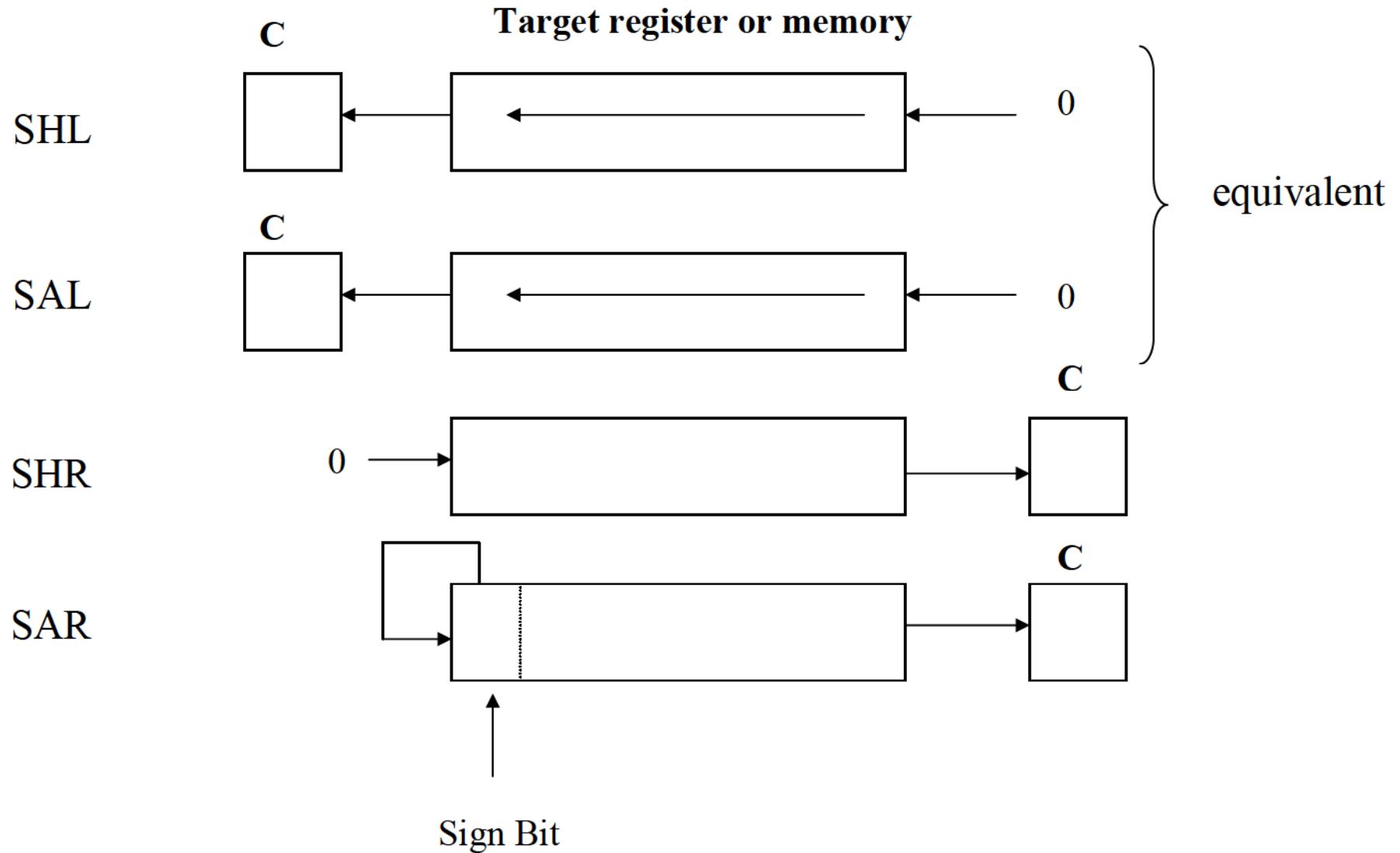
TEST AL, 1 ; test right bit

JNZ RIGHT ; if set

TEST AL, 128 ; test left bit

JNZ LEFT ; if set

Shift



Examples

Examples SHL AX,1

SAL DATA1, CL ; shift count is a modulo-32 count

Ex. ; Multiply AX by 10

 SHL AX, 1

 MOV BX, AX

 MOV CL,2

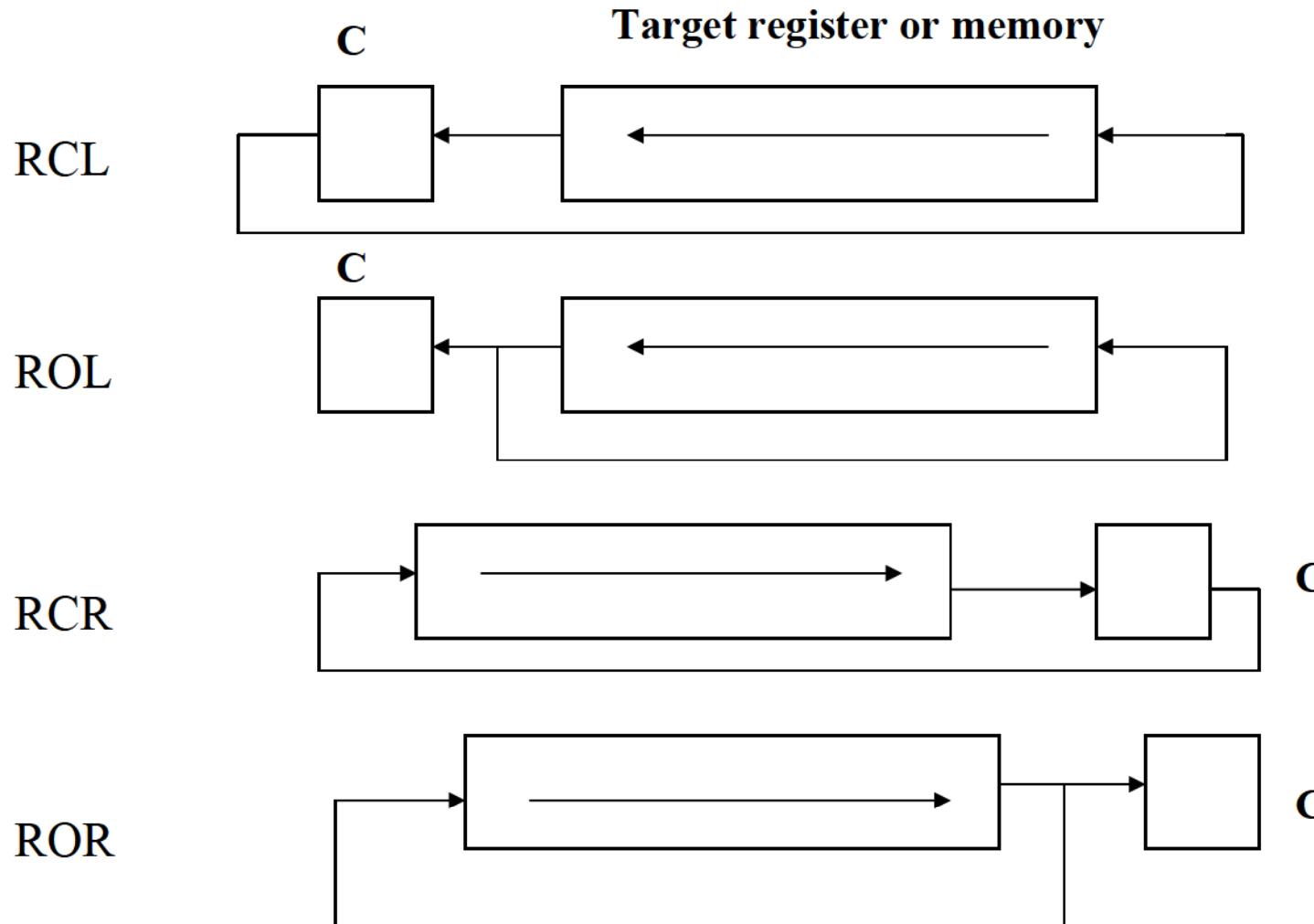
 SHL AX,CL

 ADD AX, BX

Ex. What are the results of SAR CL, 1 if CL initially contains B6H?

Ex. What are the results of SHL AL, CL if AL contains 75H
and CL contains 3?

Rotate



Ex.

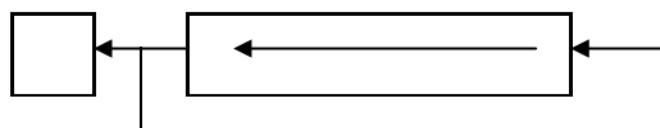
What is the result of ROL byte ptr [SI], 1 if this memory location 3C020 contains 41H?

What is the result of ROL word ptr [SI], 8 if this memory location 3C020 contains 4125H?

Example

Write a program that counts the number of 1's in a byte and writes it into BL

```
DATA1 DB 97          ; 61h
      SUB BL,BL      ;clear BL to keep the number of 1s
      MOV DL,8        ;rotate total of 8 times
      MOV AL,DATA1
AGAIN: ROL AL,1      ;rotate it once
      JNC NEXT       ;check for 1
      INC BL         ;if CF=1 then add one to count
NEXT:  DEC DL        ;go through this 8 times
      JNZ AGAIN      ;if not finished go back
      NOP
```



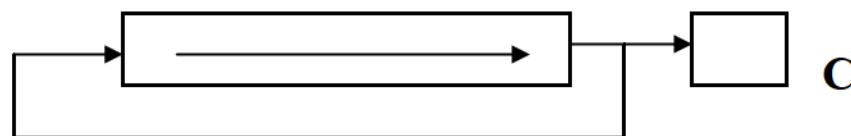
BCD and ASCII Numbers

- BCD (Binary Coded Decimal)
 - Unpacked BCD: One byte per digit
 - Packed BCD: 4 bits per digit (more efficient in storing data)
- ASCII to unpacked BCD conversion
 - Keyboards, printers, and monitors all use ASCII.
 - Digits 0 to 9 are represented by ASCII codes 30 – 39.
- **Example.** Write an 8086 program that displays the packed BCD number in register AL on the system video monitor
 - The first number to be displayed should be the MS Nibble
 - It is found by masking the LS Nibble and then rotating the MS Nibble into the LSD position
 - The result is then converted to ASCII by adding 30h
 - The BIOS video service is then called to display this result.

ASCII Numbers Example

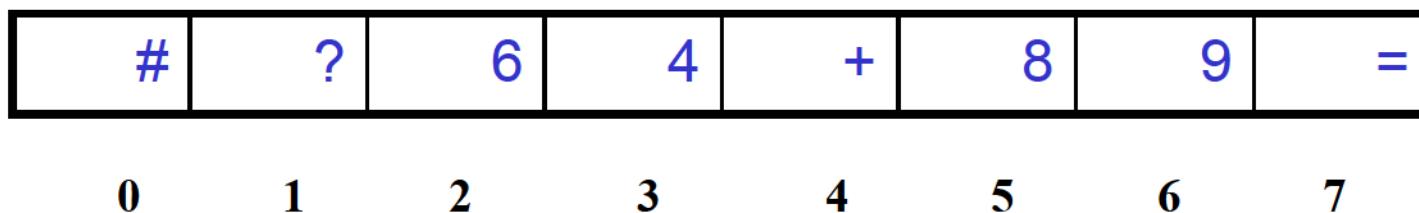
```
MOV BL,AL; save  
AND AL,F0H  
MOV CL,4  
ROR AL,CL  
ADD AL,30H  
MOV AH,0EH  
INT 10H ;display single character
```

```
MOV AL,BL; use again  
AND AL,0FH  
ADD AL,30H  
INT 10H  
INT 20H      ; RETURN TO DOS
```



Example

- Write an 8086 program that adds two packed BCD numbers input from the keyboard and computes and displays the result on the system video monitor
- Data should be in the form $64+89=$ The answer 153 should appear in the next line.



Example Continued

```
Mov dx, offset bufferaddress  
Mov ah,0a  
Mov si,dx  
Mov byte ptr [si], 8  
Int 21  
Mov ah,0eh  
Mov al,0ah  
Int 10  
; BIOS service 0e line feed position cursor
```

```
sub byte ptr[si+2], 30h  
sub byte ptr[si+3], 30h  
sub byte ptr[si+5], 30h  
sub byte ptr[si+6], 30h
```

```
Mov cl,4  
Rol byte ptr [si+3],cl  
Rol byte ptr [si+6],cl  
Ror word ptr [si+5], cl  
Ror word ptr [si+2], cl
```

```
Mov al, [si+3]  
Add al, [si+6]  
Daa  
Mov bh,al  
Jnc display  
Mov al,1  
Call display  
Mov al,bh  
Call display  
Int 20
```

8	?	6	4	+	8	9	=
---	---	---	---	---	---	---	---

Flag Control Instructions



- LAHF Load AH from flags $(AH) \leftarrow (\text{Flags})$
- SAHF Store AH into flags $(\text{Flags}) \leftarrow (AH)$
 - Flags affected: SF, ZF, AF, PF, CF
- CLC Clear Carry Flag $(CF) \leftarrow 0$
- STC Set Carry Flag $(CF) \leftarrow 1$
- CLI Clear Interrupt Flag $(IF) \leftarrow 0$
- STI Set interrupt flag $(IF) \leftarrow 1$
- Example (try with debug)

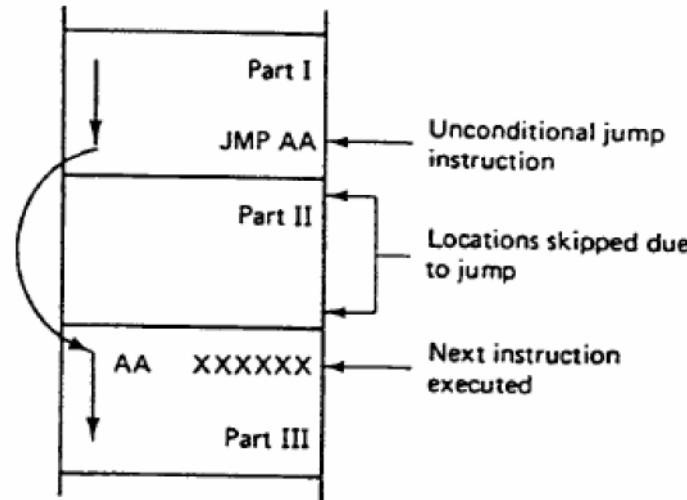
```
MOV AX,0000  
ADD AX,00  
LAHF  
MOV AX,0000  
SAHF  
– Check the flag changes!
```

Bulk manipulation
of the flags

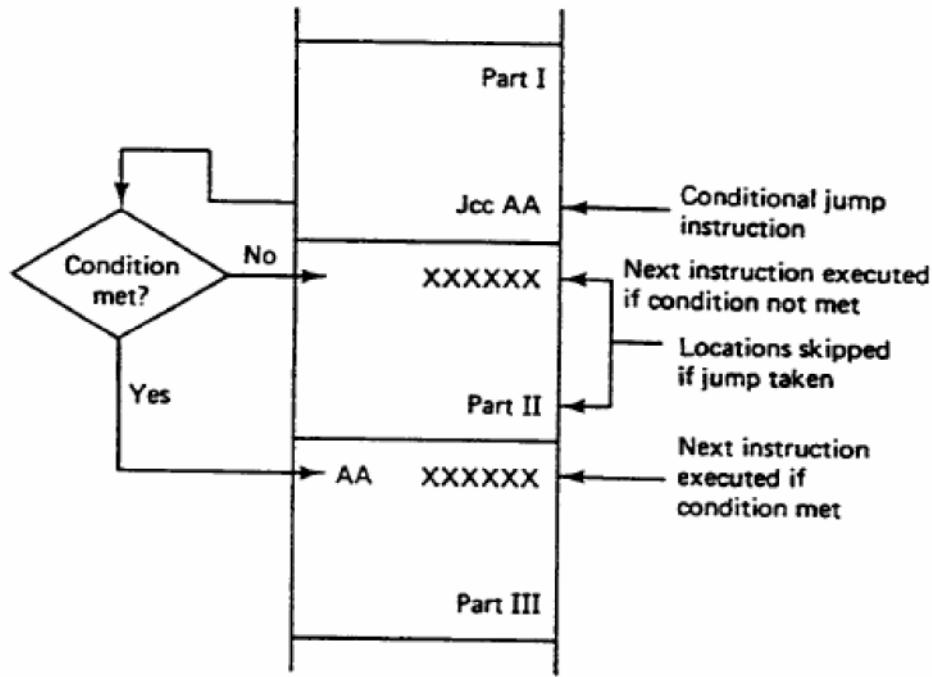
Individual
manipulation of
the flags

Jump Instructions

- Unconditional vs conditional jump



(a)



(b)

Conditional Jump

These flags are based on general comparison

Mnemonic	Description	Flags/Registers
JZ	Jump if ZERO	ZF = 1
JE	Jump if EQUAL	ZF = 1
JNZ	Jump if NOT ZERO	ZF = 0
JNE	Jump if NOT EQUAL	ZF = 0
JC	Jump if CARRY	CF = 1
JNC	Jump if NO CARRY	CF = 0
JCXZ	Jump if CX = 0	CX = 0
JECXZ	Jump if ECX = 0	ECX = 0
JP	Jump if PARITY EVEN	PF = 1
JNP	Jump if PARITY ODD	PF = 0

Jump Based on Unsigned Comparison

These flags are based on unsigned comparison

Mnemonic	Description	Flags/Registers
JA	Jump if above op1>op2	CF = 0 and ZF = 0
JNBE	Jump if not below or equal op1 not <= op2	CF = 0 and ZF = 0
JAE	Jump if above or equal op1>=op2	CF = 0
JNB	Jump if not below op1 not <op2	CF = 0
JB	Jump if below op1<op2	CF = 1
JNAE	Jump if not above nor equal op1< op2	CF = 1
JBE	Jump if below or equal op1 <= op2	CF = 1 or ZF = 1
JNA	Jump if not above op1 <= op2	CF = 1 or ZF = 1

Jump Based on Signed Comparison

These flags are based on signed comparison

Mnemonic	Description	Flags/Registers
JG	Jump if GREATER op1>op2	SF = OF AND ZF = 0
JNLE	Jump if not LESS THAN or equal op1>op2	SF = OF AND ZF = 0
JGE	Jump if GREATER THAN or equal op1>=op2	SF = OF
JNL	Jump if not LESS THAN op1>=op2	SF = OF
JL	Jump if LESS THAN op1<op2	SF <> OF
JNGE	Jump if not GREATER THAN nor equal op1<op2	SF <> OF
JLE	Jump if LESS THAN or equal op1 <= op2	ZF = 1 OR SF <> OF
JNG	Jump if NOT GREATER THAN op1 <= op2	ZF = 1 OR SF <> OF
JS	JUMP IF SIGN (NEGATIVE)	SF = 1
JNS	JUMP IF NOT SIGN (POSITIVE)	SF = 0
JO	JUMP IF OVERFLOW	OF = 1
JNO	JUMP IF NO OVERFLOW	OF = 0

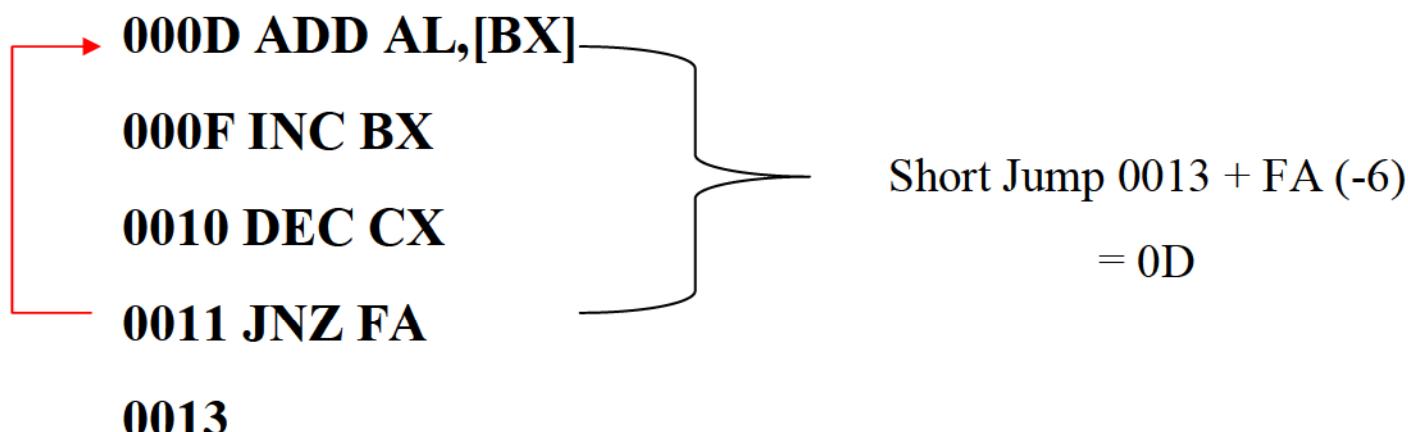
Control Transfer Instructions

- It is often necessary to transfer the program execution.
 - NEAR
 - If the control is transferred to a memory location within the current code segment (intrasegment), it is NEAR. IP is updated and CS remains the same
 - FAR
 - If the control is transferred to a memory location outside the current segment.
 - Control is passing outside the current segment both CS and IP have to be updated to the new values. ex: JMP FAR PTR label = EA 00 10 00 20
 - Short
 - A special form of the direct jump: “short jump”
 - All conditional jumps are short jumps
 - Used whenever target address is in range +127 or -128 (single byte)
 - Instead of specifying the address a relative offset is used.

Short Jumps

- Conditional Jump is a two byte instruction.
- In a jump backward the second byte is the 2's complement of the displacement value.
- To calculate the target the second byte is added to the IP of the instruction after the jump.

Ex:





Hello2.exe

SJ Example

MS-DOS Prompt - DERIK

Created with HyperSnap-DX5
To avoid this stamp, buy a license at
<http://www.hyperionics.com>

A

```
-q
C:\>cd irvine
C:\Irvine>debug hello2.exe
-u 0 25
16EF:0000 B8F116      MOV    AX,16F1
16EF:0003 8ED8        MOV    DS,AX
16EF:0005 B400        MOV    AH,00
16EF:0007 CD16        INT    16
16EF:0009 3C61        CMP    AL,61
16EF:000B 720F        JB     001C
16EF:000D 3C7A        CMP    AL,7A
16EF:000F 770B        JA    001C
16EF:0011 B409        MOV    AH,09
16EF:0013 BA1200      MOV    DX,0012
16EF:0016 B409        MOV    AH,09
16EF:0018 CD21        INT    21
16EF:001A CD20        INT    20
16EF:001C BA3A00      MOV    DX,003A
16EF:001F B409        MOV    AH,09
16EF:0021 CD21        INT    21
16EF:0023 B8004C      MOV    AX,4C00
```

```
.model small
.stack 100h
.data
.org 0010
message1 db "You now have a small letter
entered !",0dh,0ah,'$'
.org 50
message2 db "You have NON small letters
",0dh,0ah,'$'
.code
main proc
    mov ax,@data
    mov ds,ax
    mov ah,00h
    int 16h
    cmp al,61h
    jb next
    Cmp al,7Ah
    ja next
    mov ah,09h
    mov dx,offset message1
    mov ah,09h
    int 21h
    int 20h
next: mov dx,offset message2
    mov ah,09h
    int 21h
    mov ax,4C00h
    int 21h
main endp
end main
```

A Simple Example Program finds the sum

- Write a program that adds 5 bytes of data and saves the result. The data should be the following numbers: 25,12,15,10,11

```
.model small
.stack 100h
.data
    Data_in DB 25,12,15,10,11
    Sum DB ?
.code
main proc far
    mov ax, @Data
    mov ds,ax
    mov cx,05h
    mov bx,offset data_in
    mov al,0
```

```
Again: add al,[bx]
        inc bx
        dec cx
        jnz Again
        mov sum,al
        mov ah,4Ch
        INT 21H
Main    endp
end main
```



Example Output

Created with HyperSnap-DX5
To avoid this stamp, buy a license at
<http://www.hyperionics.com>

A

Run Data Options Calls Windows Help

[3] source1 CS:IP EX1.asm

```
1: .model small
2: .stack 100h
3: .data
4:     Data_in DB      25,12,15,10,11
5:     Sum DB ?
6: .code
7: main proc far
8:     mov ax, @Data
1D5B:0000 B85C1D      MOV      AX,1D5C
9:     mov ds,ax
1D5B:0003 8ED8        MOV      DS,AX
```

[4] source2 EX1.asm

[5] memory1 b 0x1D5C:0x0000

1D5C:0000	49 75 EA A2 0F 00 B4 4C CD 21 19 0C 0F	Iu·óø.-! øø
1D5C:000D	0A 0B 49 4E 42 30 38 34 02 00 00 00 00	øTNB084ø....
1D5C:001A	00 00 01 00 43 56 01 00 00 00 00 00 00	..ø.CVø.....

> Process 0x1D4B terminated normally (2)

<F8=Trace> <F10=Step> <F5=Go> <F3=S1 Fmt> <Sh+F3=M1 Fmt>

[7] reg

AX	= 1D49
BX	= 000F
CX	= 0000
DX	= 0000
SP	= 0100
BP	= 0000
SI	= 0000
DI	= 0000
DS	= 1D5C
ES	= 1D4B
SS	= 1D5D
CS	= 1D5B
IP	= 0016
FL	= 3246

NV UP EI PL
ZR NA PE NC

DEC

Unconditional Jump

- ❖ Short Jump: jmp short L1 (8 bit)
- ❖ Near Jump: jmp near ptr Label
 - This is the **default** jump: JMP Label
 - The displacement (16 bit) is added to the IP of the instruction following jump instruction.
 - The displacement can be in the range of -32,768 to 32,768.
 - The target address can be register indirect, or assigned by the label.
 - **Register indirect JMP:** the target address is the contents of two memory locations pointed at by the register.
 - Ex: JMP [SI] will replace the IP with the contents of the memory locations pointed by DS:DI and DI+1 or JMP [BP + SI + 1000] in SS
- ❖ Far Jump: jmp far ptr Label
 - this is a jump out of the current segment.

Compare

Mnemonic	Meaning	Format	Operation	Flags Affected
CMP	Compare	CMP D,S	(D) – (S) is used in setting or resetting the flags	CF, AF, OF, PF, SF, ZF

(a)

Unsigned Comparison		
Comp Operands	CF	ZF
Dest > source	0	0
Dest = source	0	1
Dest < source	1	0

Destination	Source
Register	Register
Register	Memory
Memory	Register
Register	Immediate
Memory	Immediate
Accumulator	Immediate

(b)

Signed Comparison		
Comp Operands	ZF	SF,OF
Dest > source	0	SF=OF
Dest = source	1	?
Dest < source	?	SF<>OF

Compare Example

```
DATA1      DW 235Fh
```

...

```
MOV AX, CCCCCH
```

```
CMP AX, DATA1
```

```
JNC OVER
```

```
SUB AX,AX
```

```
OVER: INC DATA1
```

CCCC – 235F = A96D => Z=0, CF=0 =>

CCCC > DATA1

Compare (CMP)

For ex: CMP CL,BL ; CL-BL; no modification on neither operands

Write a program to find the highest among 5 grades and write it in DL

```
DATA    DB      51, 44, 99, 88, 80          ;13h,2ch,63h,58h,50h
       MOV     CX,5
       MOV     BX, OFFSET DATA
       SUB     AL,AL
AGAIN: CMP    AL,[BX]
       JA     NEXT
       MOV    AL,[BX]
NEXT:  INC    BX
       LOOP AGAIN
       MOV    DL, AL
       ;set up loop counter
       ;BX points to GRADE data
       ;AL holds highest grade found so far
       ;compare next grade to highest
       ;jump if AL still highest
       ;else AL holds new highest
       ;point to next grade
       ;continue search
```