CSE 230

Assembly for 8086

Lecture 6

80 x 86 Assembly Programming

Some slides are taken from the Mazidi book slides, Brey book and from Dr. Ali Ziya Alkar (Ph.D University of Colorado, Boulder)

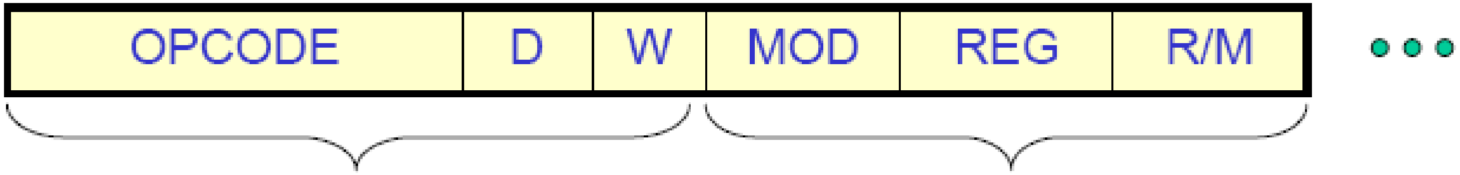
Assembly Language

- There is a one-to-one relationship between assembly and machine language instructions
- What is found is that a compiled machine code implementation of a program written in a high-level language results in inefficient code
  - More machine language instructions than an assembled version of an equivalent handwritten assembly language program
- Two key benefits of assembly language programming
  - It takes up less memory
  - It executes much faster

Languages in terms of applications

- One of the most beneficial uses of assembly language programming is **real-time applications**.
- Real time means the task required by the application must be completed before any other input to the program that will alter its operation can occur
- For example the device service routine which controls the operation of the floppy disk drive is a good example that is usually written in assembly language
- Assembly language not only good for controlling hardware devices but also **performing pure software operations**
  - searching through a large table of data for a special string of characters
  - Code translation from ASCII to EBCDIC
  - Table sort routines
  - Mathematical routines
- Assembly language: perform real-time operations
- High-level languages: Those operations mostly not critical in time.

Converting Assembly Language Instructions to Machine Code



OPCODE	D	W	MOD	REG	R/M	•••

- An instruction can be coded with 1 to 6 bytes
- **Byte 1 contains three kinds of information:**
  - Opcode field (6 bits) specifies the operation such as add, subtract, or move
  - Register Direction Bit (D bit)
    - Tells the register operand in REG field in byte 2 is source or destination operand
      - 1:Data flow to the REG field from R/M
      - 0: Data flow from the REG field to the R/M
  - Data Size Bit (W bit)
    - Specifies whether the operation will be performed on 8-bit or 16-bit data
      - 0: 8 bits
      - 1: 16 bits
- **Byte 2 has two fields:**
  - Mode field (MOD) – 2 bits
  - Register field (REG) - 3 bits
  - Register/memory field (R/M field) – 2 bits

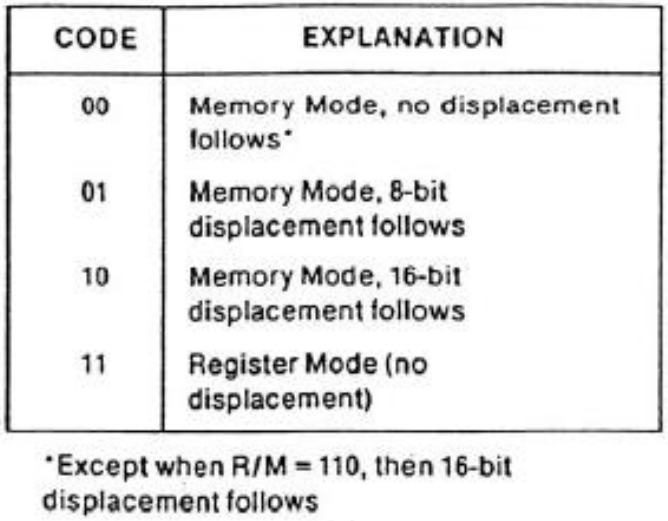
Continued

- REG field is used to identify the register for the first operand

REG	W = 0	W = 1
000	AL	AX
001	CL	CX
010	DL	DX
011	BL	BX
100	AH	SP
101	CH	BP
110	DH	SI
111	BH	DI

Continued

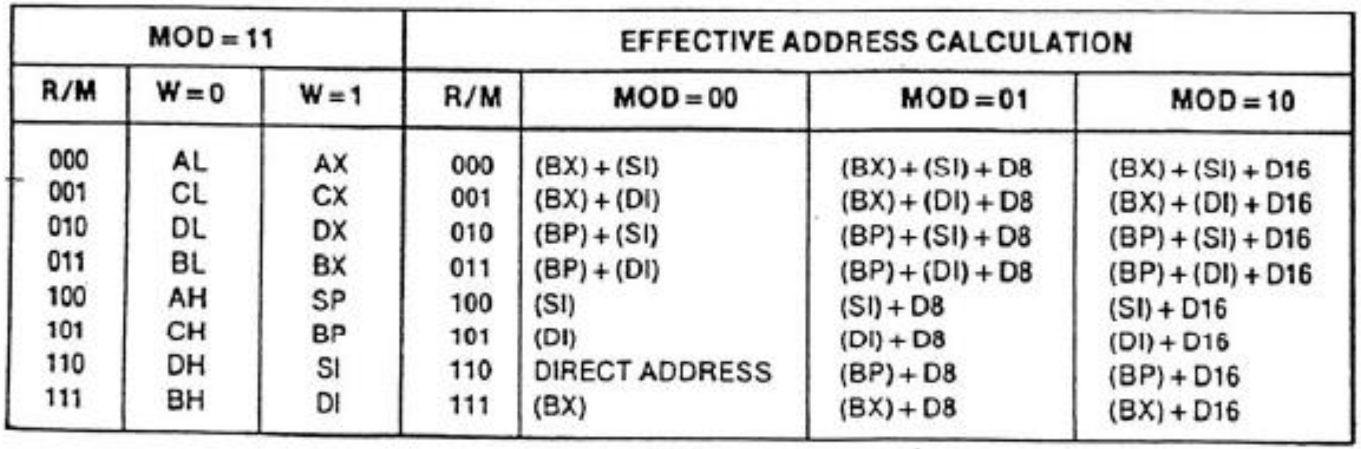
- 2-bit MOD field and 3-bit R/M field together specify the second operand



CODE	EXPLANATION
00	Memory Mode, no displacement follows*
01	Memory Mode, 8-bit displacement follows
10	Memory Mode, 16-bit displacement follows
11	Register Mode (no displacement)

*Except when R/M = 110, then 16-bit displacement follows

(a)



MOD = 11			EFFECTIVE ADDRESS CALCULATION			
R/M	W = 0	W = 1	R/M	MOD = 00	MOD = 01	MOD = 10
000	AL	AX	000	(BX) + (SI)	(BX) + (SI) + D8	(BX) + (SI) + D16
001	CL	CX	001	(BX) + (DI)	(BX) + (DI) + D8	(BX) + (DI) + D16
010	DL	DX	010	(BP) + (SI)	(BP) + (SI) + D8	(BP) + (SI) + D16
011	BL	BX	011	(BP) + (DI)	(BP) + (DI) + D8	(BP) + (DI) + D16
100	AH	SP	100	(SI)	(SI) + D8	(SI) + D16
101	CH	BP	101	(DI)	(DI) + D8	(DI) + D16
110	DH	SI	110	DIRECT ADDRESS	(BP) + D8	(BP) + D16
111	BH	DI	111	(BX)	(BX) + D8	(BX) + D16

(b)

Examples

- MOV BL,AL
- Opcode for MOV = 100010
- We'll encode AL so
  - D = 0 (AL source operand)
- W bit = 0 (8-bits)
- MOD = 11 (register mode)
- REG = 000 (code for AL)
- R/M = 011

OPCODE	D	W	MOD	REG	R/M
100010	0	0	11	000	011

MOV BL,AL => 10001000 11000011 = 88 C3h
ADD AX,[SI] => 00000011 00000100 = 03 04 h
ADD [BX][DI] + 1234h, AX => 00000001 10000001 __ __ h
=> 01 81 34 12 h

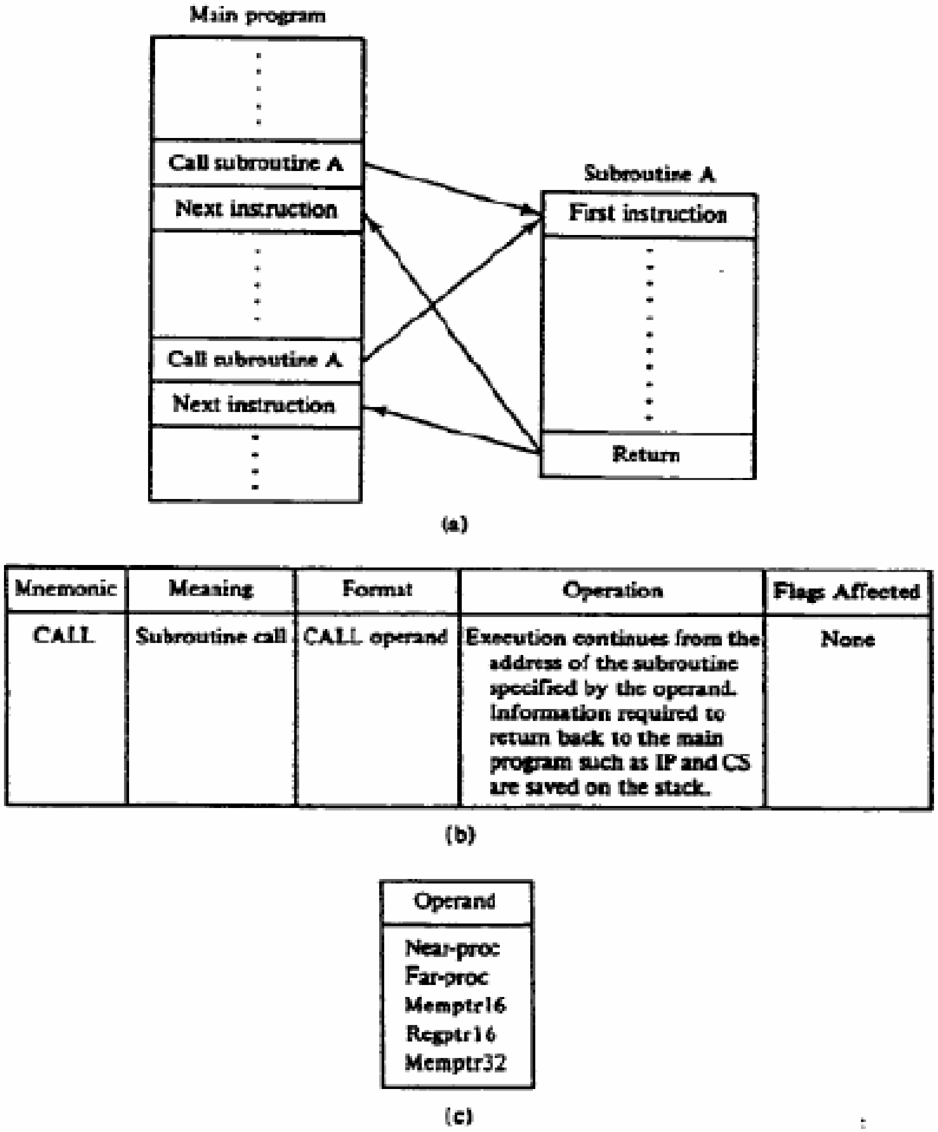
Subroutines and Subroutine Handling Functions

✓A subroutine is a special segment of a program that can be called for execution from any point in the program

✓A RET instruction must be included at the end of the subroutine to initiate the return sequence to the main program environment

Examples. **Call 1234h**
Call BX
Call [BX]

Two calls
•intrasegment
•intersegment



Mnemonic	Meaning	Format	Operation	Flags Affected
CALL	Subroutine call	CALL operand	Execution continues from the address of the subroutine specified by the operand. Information required to return back to the main program such as IP and CS are saved on the stack.	None

(b)

Operand
Near-proc Far-proc Memptr16 Regptr16 Memptr32

(c)

Figure 6–20 (a) Subroutine concept. (b) Subroutine call instruction. (c) Allowed operands.

Calling a NEAR proc

✓The CALL instruction and the subroutine it calls are in the same segment.

✓Save the current value of the IP on the stack.

✓load the subroutine's offset into IP (nextinst + offset)

Calling Program   Subroutine   Stack

```
Main proc          sub1 proc
001A: call sub1    0080: mov ax,1
001D: inc ax       …
.                  ret
Main endp          sub1 endp
```

1ffd	1D
1ffe	00
1fff	(not used)

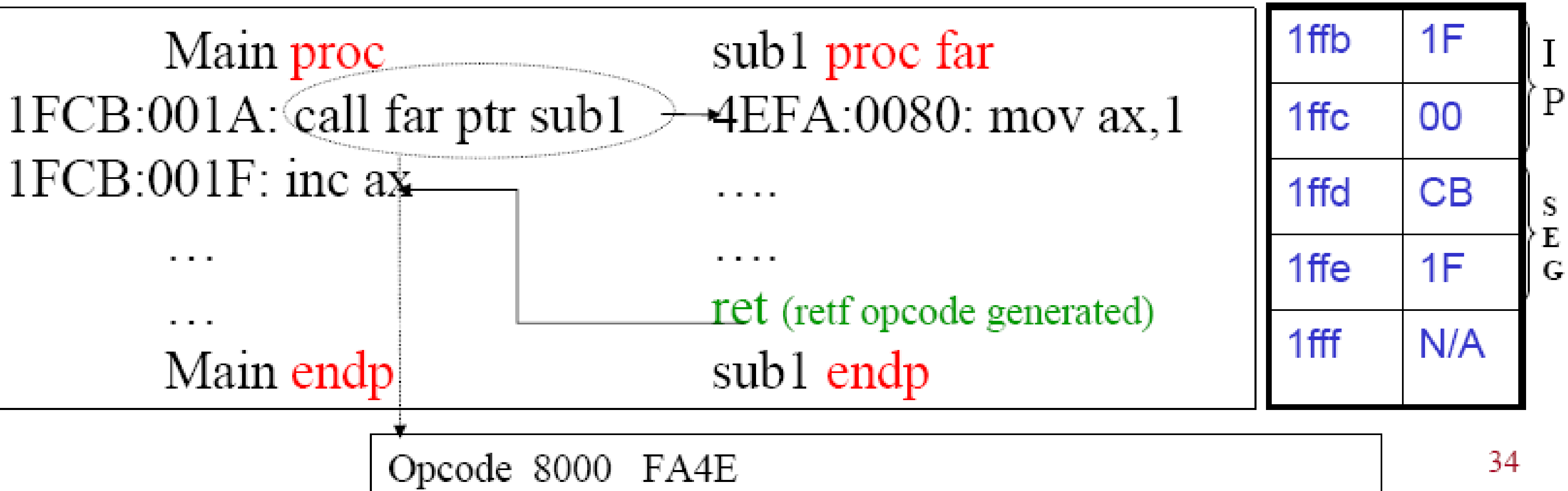
Calling a FAR proc

✓The CALL instruction and the subroutine it calls are in the “Different” segments.

✓Save the current value of the CS and IP on the stack.

✓Then load the subroutine’s CS and offset into IP.

Calling Program Subroutine Stack

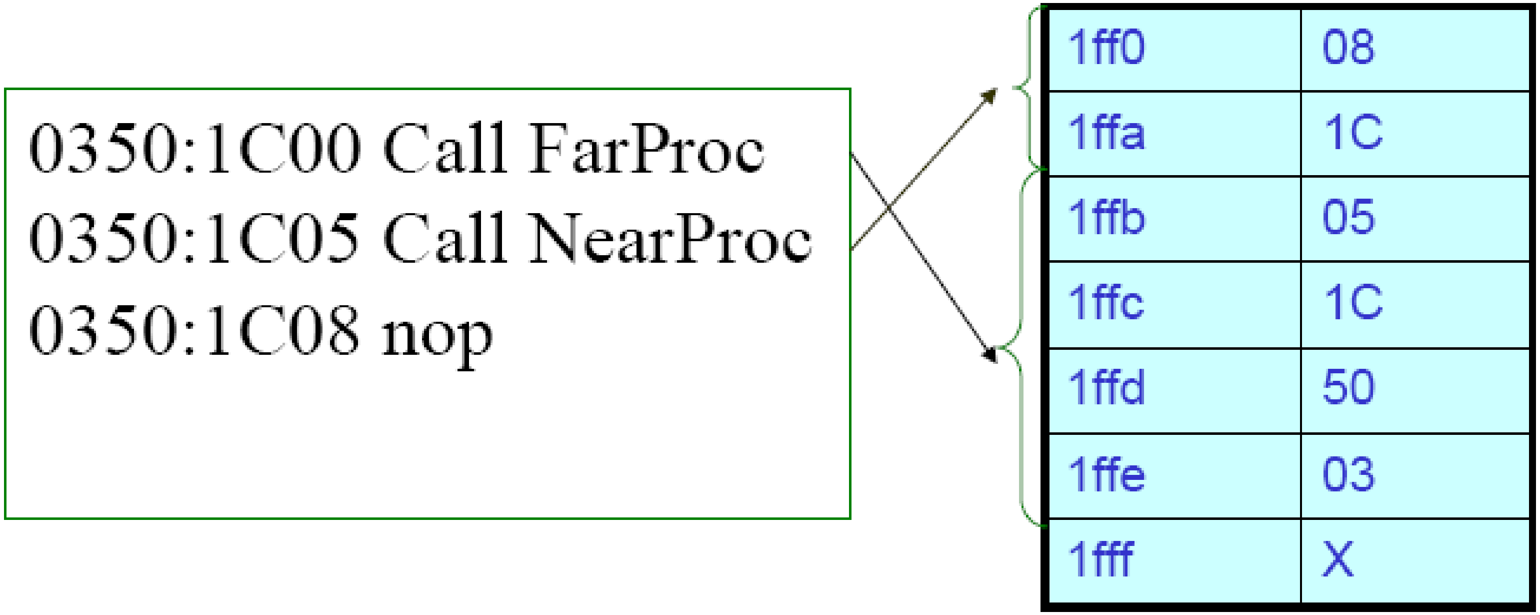


```
        Main proc                         sub1 proc far
1FCB:001A: call far ptr sub1         4EFA:0080: mov ax,1
1FCB:001F: inc ax                         ....
        ...                               ....
        ...                               ret (retf opcode generated)
        Main endp                         sub1 endp
```

1ffb	1F	IP
1ffc	00	
1ffd	CB	SEG
1ffe	1F	
1fff	N/A	

Opcode 8000 FA4E

Example on Far/Near Procedure Calls



```
0350:1C00 Call FarProc
0350:1C05 Call NearProc
0350:1C08 nop
```

1ff0	08
1ffa	1C
1ffb	05
1ffc	1C
1ffd	50
1ffe	03
1fff	X

Nested Procedure Calls

A subroutine may itself call other subroutines.

Example:

```
            main proc
000A        call subr1
000C        mov ax,…
…

            main endp
```

```
            subr1  proc
 0030       nop
            …
            call subr2
0040        ret …

            subr1 endp
```

```
            subr2  proc
 0050       nop
            …
            call subr3
0060        ret …

            subr2 endp
```

```
            subr3  proc
 0070       nop
            …
0079        nop
007A        ret
            subr3 endp
```

Q: show the stack contents at 0079?

1ff0	60
1ffa	00
1ffb	40
1ffc	00
1ffd	0c
1ffe	00
1fff	X

Do NOT overlap Procedure Declarations

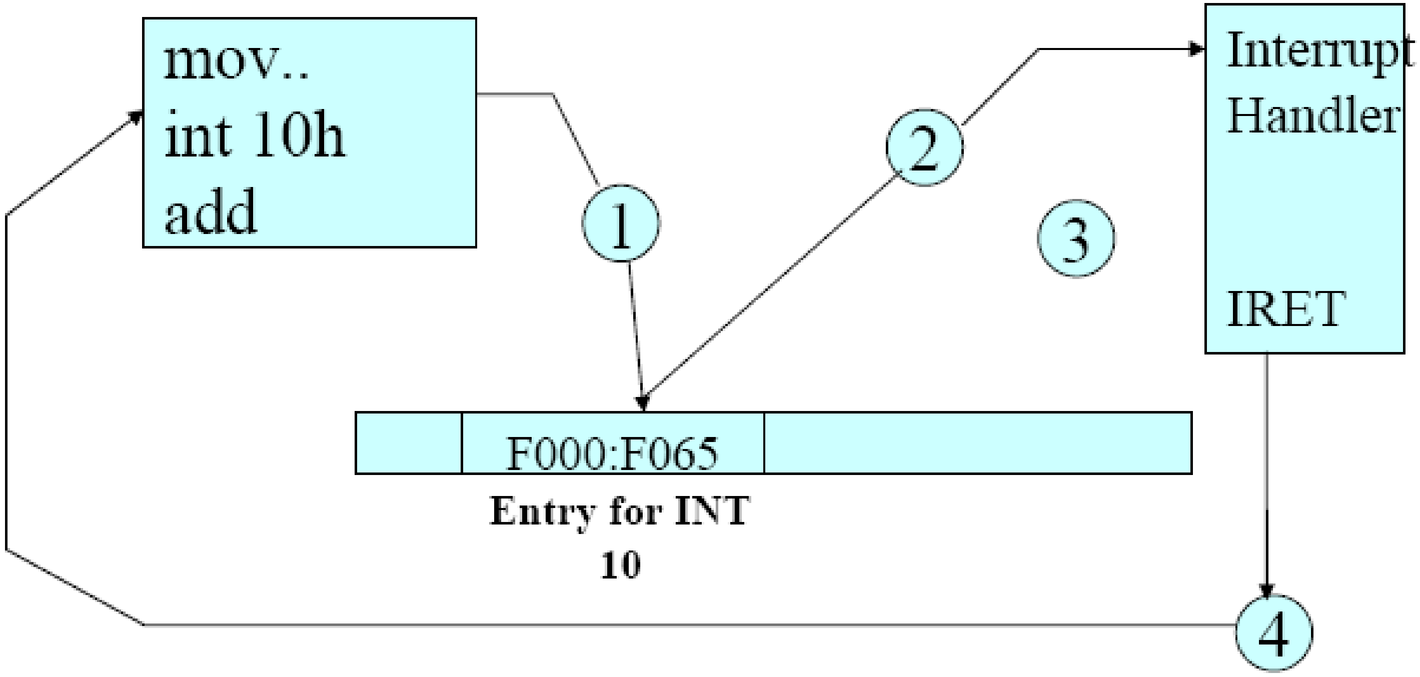
80x86 Interrupts

- 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.

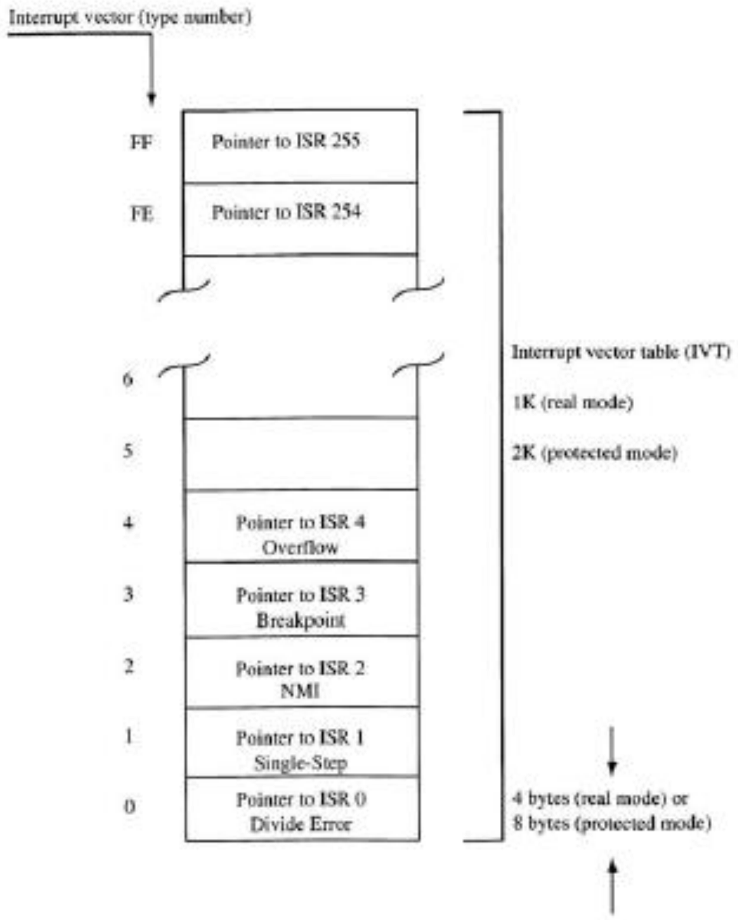
- Each interrupt must supply a type number which is used by the processor as a pointer to an interrupt vector table (IVT) to determine the address of that interrupt's service routine
- Interrupt Vector Table: CPU processes an interrupt instruction using the interrupt vector table (This table resides in the lowest 1K memory)
- Each entry in the IVT=32 bit segment+offset adress 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



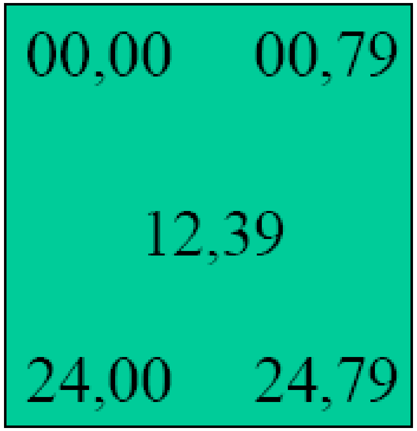
Processor	Pointer Size	IVT Location
Real Mode	4 bytes	Address 00000000-000003FF
Protected Mode	8 bytes	Anywhere in Physical Memory

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.

Some Software Interrupts

- 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

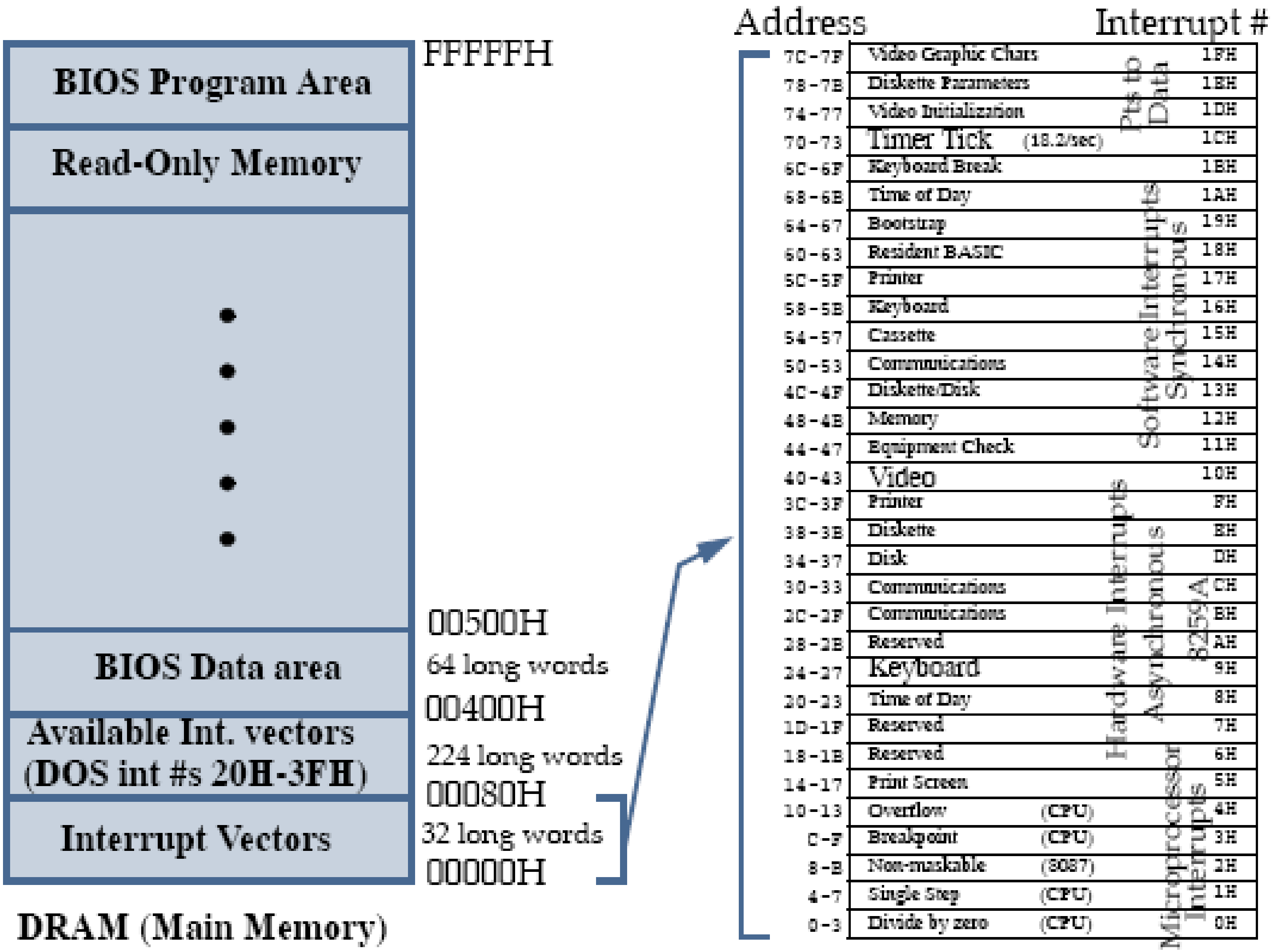


Cursor Locations

<u>Example:</u> 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
```

Interrupt Vectors (DOS PC)

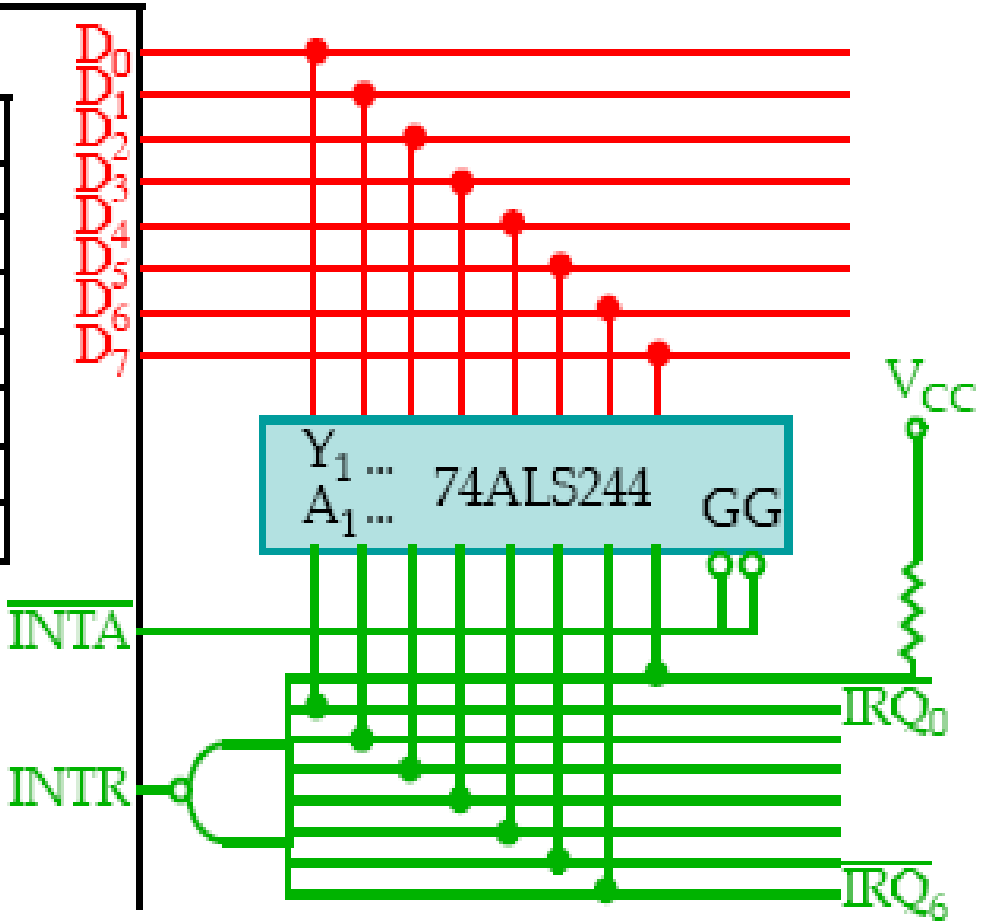


DRAM (Main Memory)

Address				Interrupt #
7C-7F	Video Graphic Chars		Pts to Data	1FH
78-7B	Diskette Parameters			1EH
74-77	Video Initialization			1DH
70-73	Timer Tick	(18.2/sec)		1CH
6C-6F	Keyboard Break		Software Interrupts Synchronous	1BH
68-6B	Time of Day			1AH
64-67	Bootstrap			19H
60-63	Resident BASIC			18H
5C-5F	Printer			17H
58-5B	Keyboard			16H
54-57	Cassette			15H
50-53	Communications			14H
4C-4F	Diskette/Disk			13H
48-4B	Memory			12H
44-47	Equipment Check			11H
40-43	Video			10H
3C-3F	Printer		Hardware Interrupts Asynchronous 8259A	FH
38-3B	Diskette			EH
34-37	Disk			DH
30-33	Communications			CH
2C-2F	Communications			BH
28-2B	Reserved			AH
24-27	Keyboard			9H
20-23	Time of Day			8H
1D-1F	Reserved			7H
18-1B	Reserved			6H
14-17	Print Screen			5H
10-13	Overflow	(CPU)	Microprocessor Interrupts	4H
C-F	Breakpoint	(CPU)		3H
8-B	Non-maskable	(8087)		2H
4-7	Single Step	(CPU)		1H
0-3	Divide by zero	(CPU)		0H

Handling more than 1 IRQ

IRQs							
6	5	4	3	2	1	0	Vect
1	1	1	1	1	1	0	FEH
1	1	1	1	1	0	1	FDH
1	1	1	1	0	1	1	FBH
1	1	1	0	1	1	1	F7H
1	1	0	1	1	1	1	EFH
1	0	1	1	1	1	1	DFH
0	1	1	1	1	1	1	BFH

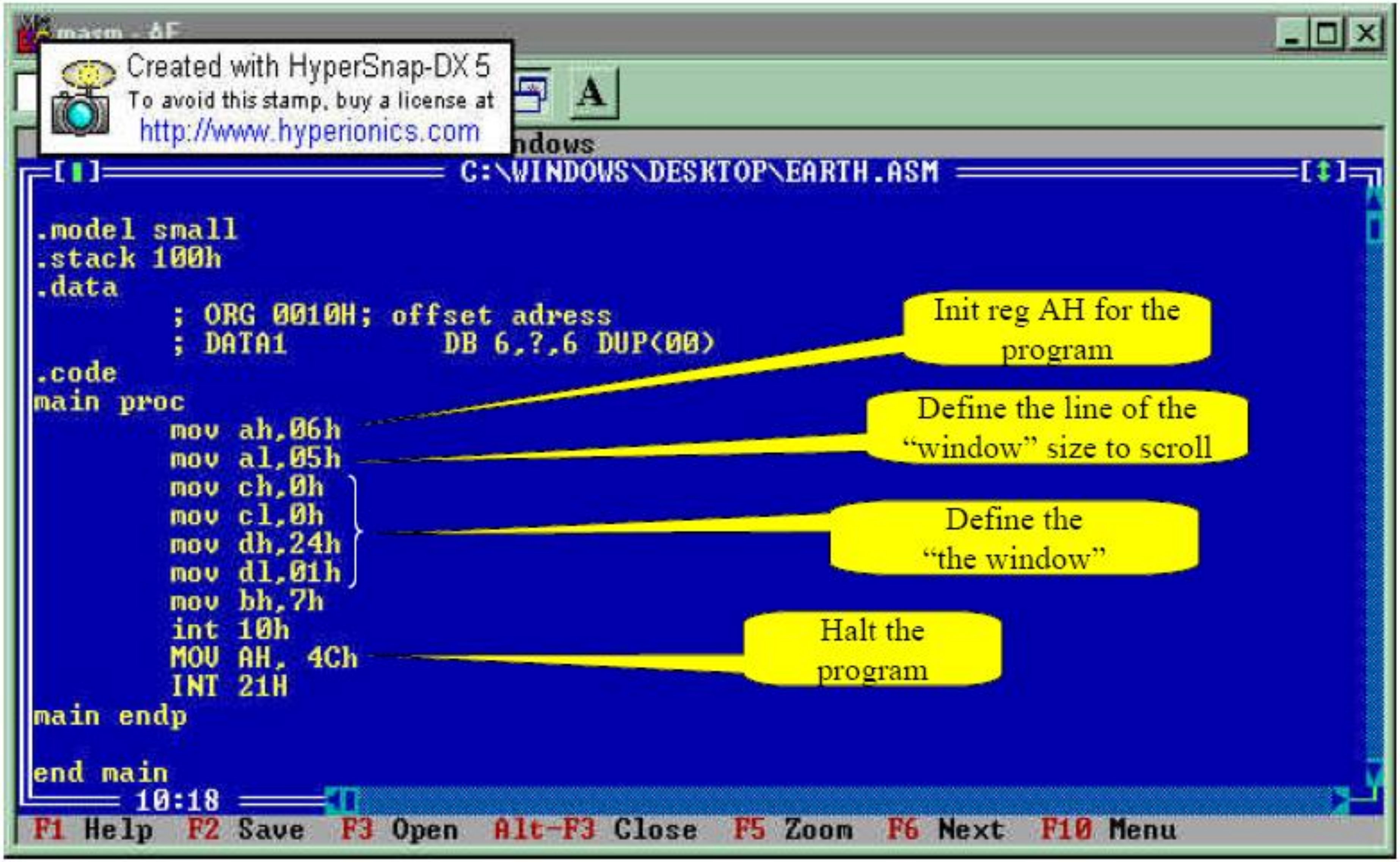


If any of $\overline{IRQ}_x$ goes low, the NAND goes low requesting an interrupt.

Note that if more than one IRQ goes low, a unique interrupt vector is generated and an interrupt priority needs to be defined.

The Interrupt Vector table must be expanded to accommodate this.

Example Int10 06


masm - AE
ndows
C:\WINDOWS\DESKTOP\EARTH.ASM
10:18
F1 Help F2 Save F3 Open Alt-F3 Close F5 Zoom F6 Next F10 Menu

```
.model small
.stack 100h
.data
        ; ORG 0010H; offset adress
        ; 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
        MOV AH, 4Ch
        INT 21H
main endp

end main
```

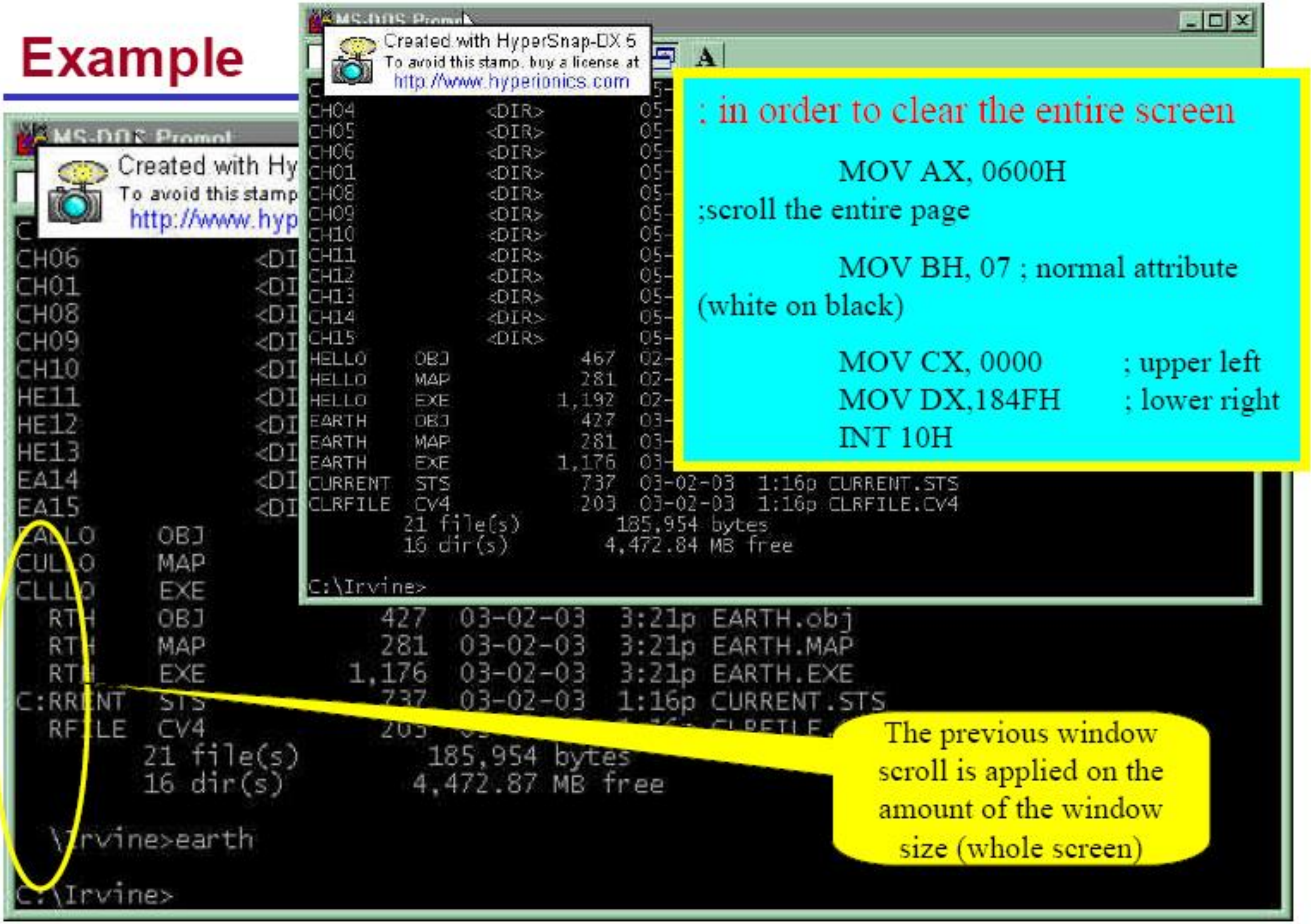
Init reg AH for the program

Define the line of the “window” size to scroll

Define the “the window”

Halt the program

Example



```
C
CH04          <DIR>        05-
CH05          <DIR>        05-
CH06          <DIR>        05-
CH01          <DIR>        05-
CH08          <DIR>        05-
CH09          <DIR>        05-
CH10          <DIR>        05-
CH11          <DIR>        05-
CH12          <DIR>        05-
CH13          <DIR>        05-
CH14          <DIR>        05-
CH15          <DIR>        05-
HELLO    OBJ          467  02-
HELLO    MAP          281  02-
HELLO    EXE        1,192  02-
EARTH    OBJ          427  03-
EARTH    MAP          281  03-
EARTH    EXE        1,176  03-
CURRENT  STS          737  03-02-03  1:16p CURRENT.STS
CLRFILE  CV4          203  03-02-03  1:16p CLRFILE.CV4
        21 file(s)        185,954 bytes
        16 dir(s)       4,472.84 MB free

C:\Irvine>
```

```
C
CH06                   <DI
CH01                   <DI
CH08                   <DI
CH09                   <DI
CH10                   <DI
HE11                   <DI
HE12                   <DI
HE13                   <DI
EA14                   <DI
EA15                   <DI
EALLO    OBJ
CULLO    MAP
CLLLO    EXE
  RTH    OBJ               427  03-02-03  3:21p EARTH.obj
  RTH    MAP               281  03-02-03  3:21p EARTH.MAP
  RTH    EXE             1,176  03-02-03  3:21p EARTH.EXE
C:RRENT  STS               737  03-02-03  1:16p CURRENT.STS
  RFILE  CV4               203
       21 file(s)        185,954 bytes
       16 dir(s)       4,472.87 MB free

  \Irvine>earth

C:\Irvine>
```

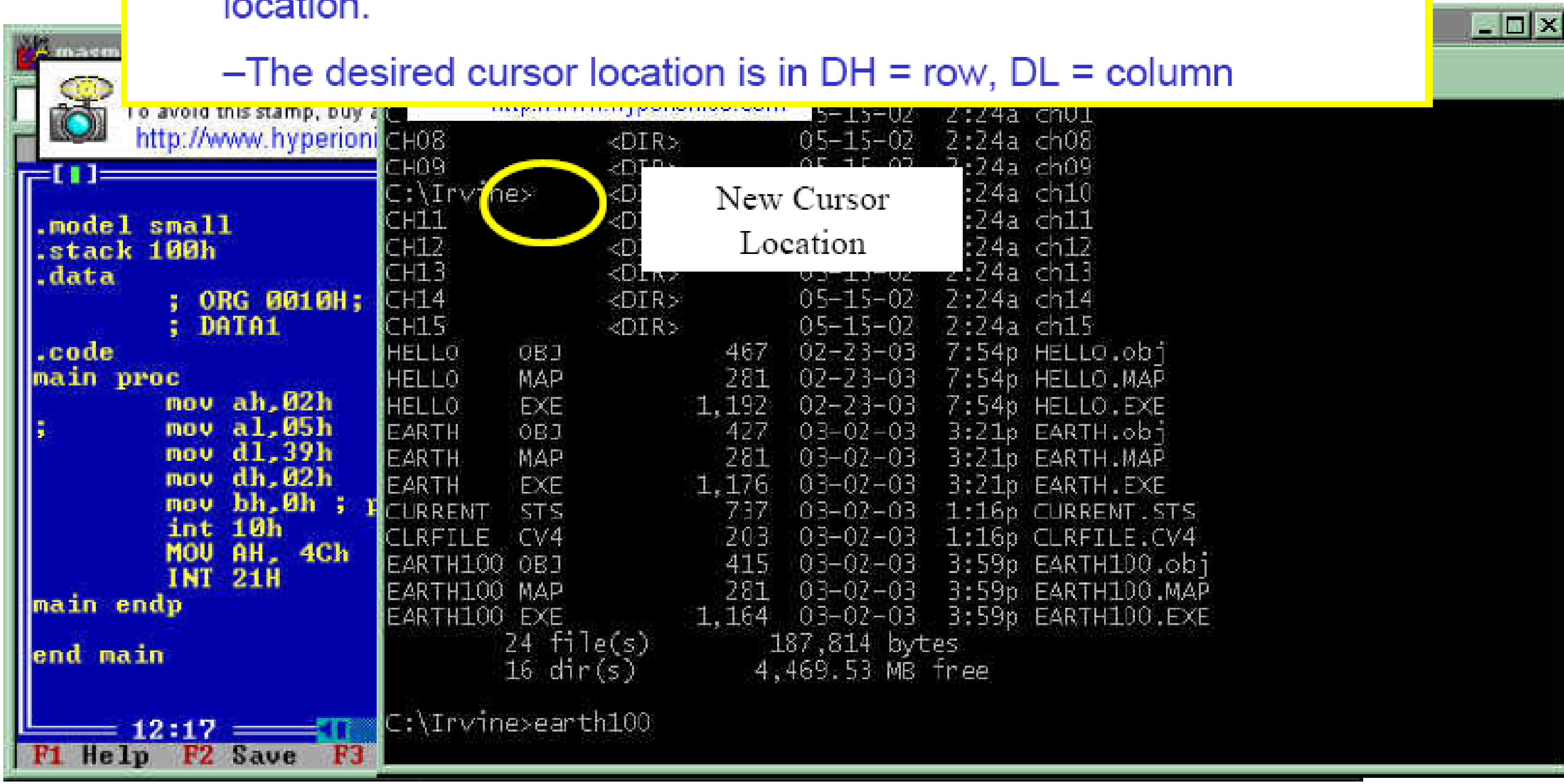
: in order to clear the entire screen

```
        MOV AX, 0600H
;scroll the entire page
        MOV BH, 07 ; normal attribute
(white on black)
        MOV CX, 0000        ; upper left
        MOV DX,184FH        ; lower right
        INT 10H
```

The previous window scroll is applied on the amount of the window size (whole screen)

Int 10 02H

- **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



```
.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 ; p
        int 10h
        MOU AH, 4Ch
        INT 21H
main endp

end main
```

```
C                           5-15-02   2:24a ch01
CH08           <DIR>        05-15-02  2:24a ch08
CH09           <DI                    2:24a ch09
C:\Irvine>     <DI                    :24a ch10
CH11           <DI                    :24a ch11
CH12           <DI                    :24a ch12
CH13           <DIR>                  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
HELLO    EXE         1,192  02-23-03  7:54p HELLO.EXE
EARTH    OBJ           427  03-02-03  3:21p EARTH.obj
EARTH    MAP           281  03-02-03  3:21p EARTH.MAP
EARTH    EXE         1,176  03-02-03  3:21p EARTH.EXE
CURRENT  STS           737  03-02-03  1:16p CURRENT.STS
CLRFILE  CV4           203  03-02-03  1:16p CLRFILE.CV4
EARTH100 OBJ           415  03-02-03  3:59p EARTH100.obj
EARTH100 MAP           281  03-02-03  3:59p EARTH100.MAP
EARTH100 EXE         1,164  03-02-03  3:59p EARTH100.EXE
        24 file(s)         187,814 bytes
        16 dir(s)     4,469.53 MB free

C:\Irvine>earth100
```

New Cursor Location

Int 10 03

- **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

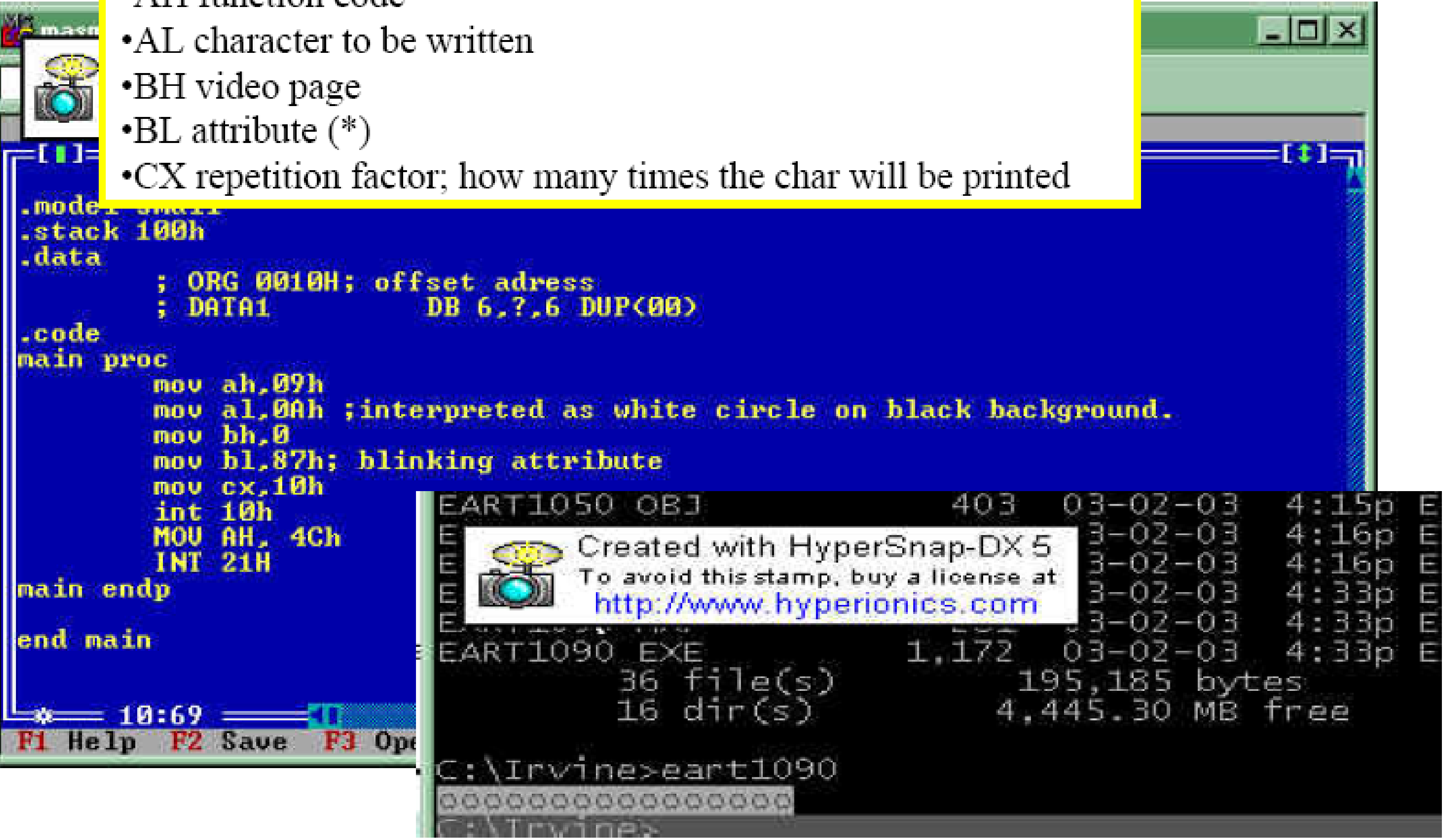
- **INT 10H function 05**; switch between video modes by adjusting AL

```
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 - 09h or 0A (* no attribute)

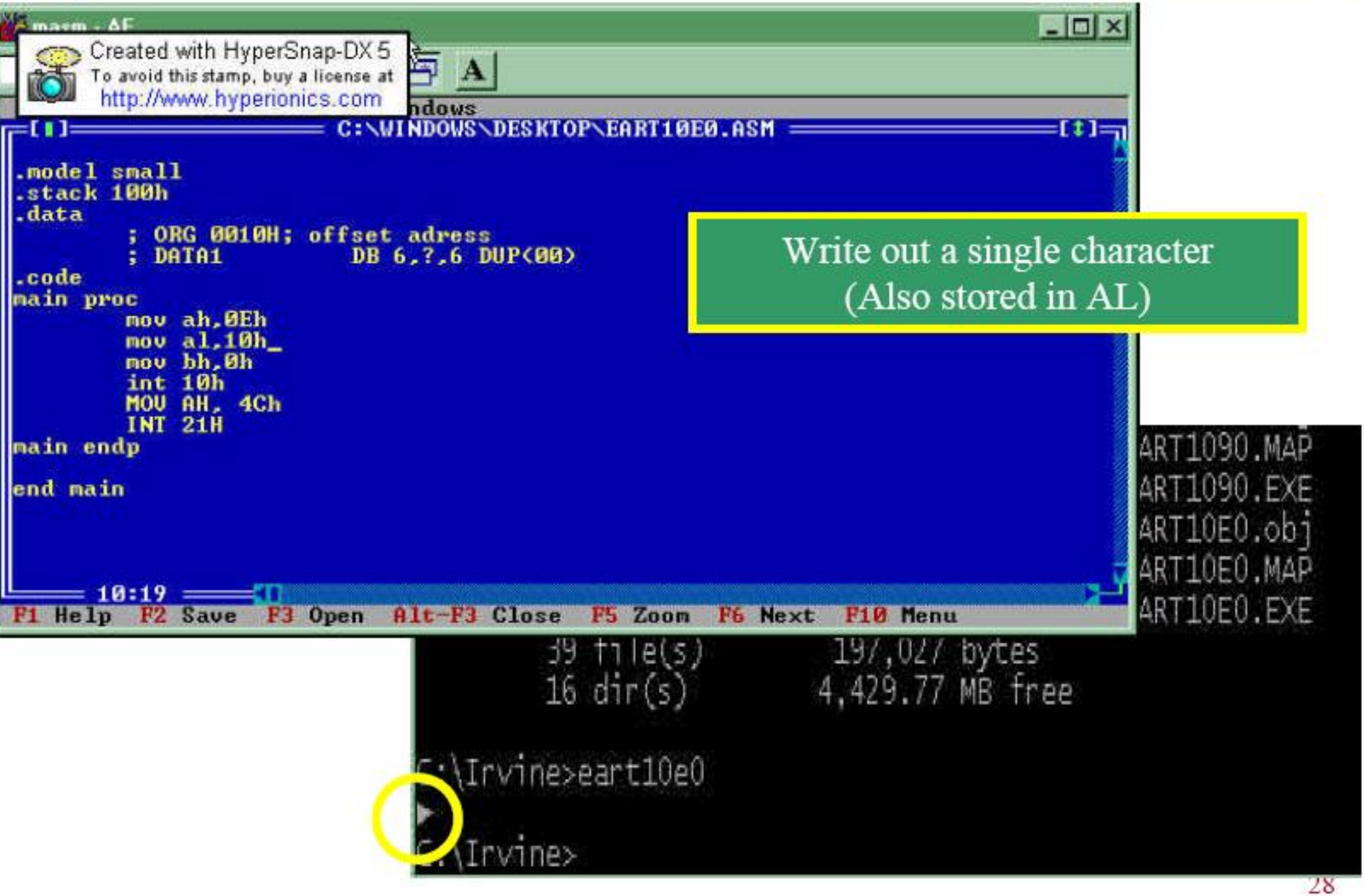
- Write *one or more* characters at the current cursor position
- This function can display any ASCII character.
- AH function code
- AL character to be written
- BH video page
- BL attribute (*)
- CX repetition factor; how many times the char will be printed



```
.model small
.stack 100h
.data
        ; ORG 0010H; offset adress
        ; 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
        MOV AH, 4Ch
        INT 21H
main endp

end main
```

```
EART1050 OBJ           403  03-02-03   4:15p E
E                           3-02-03   4:16p E
E                           3-02-03   4:16p E
E                           3-02-03   4:33p E
E                           3-02-03   4:33p 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

C:\Irvine>eart1090
oooooooooooooooo
C:\Irvine>
```

Int 10 - 0e


masm - AE
ndows
C:\WINDOWS\DESKTOP\EART10E0.ASM
10:19
F1 Help F2 Save F3 Open Alt-F3 Close F5 Zoom F6 Next F10 Menu

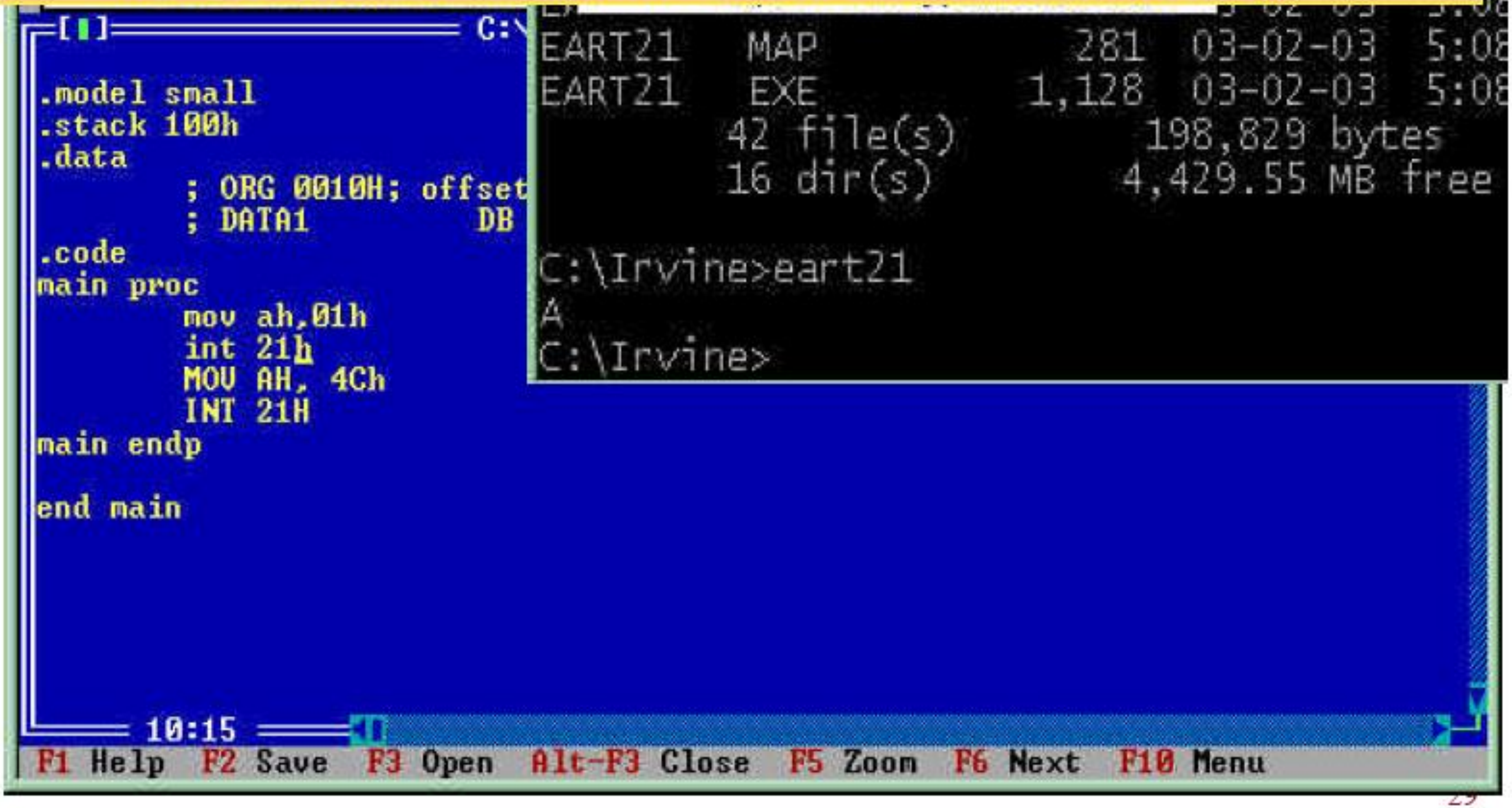
```
.model small
.stack 100h
.data
        ; ORG 0010H; offset adress
        ; DATA1         DB 6,?,6 DUP(00)
.code
main proc
        mov ah,0Eh
        mov al,10h_
        mov bh,0h
        int 10h
        MOV AH, 4Ch
        INT 21H
main endp

end main
```

Write out a single character
(Also stored in AL)

```
ART1090.MAP
ART1090.EXE
ART10E0.obj
ART10E0.MAP
ART10E0.EXE
      39 file(s)        197,027 bytes
      16 dir(s)      4,429.77 MB free

C:\Irvine>eart10e0
►
C:\Irvine>
```

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.

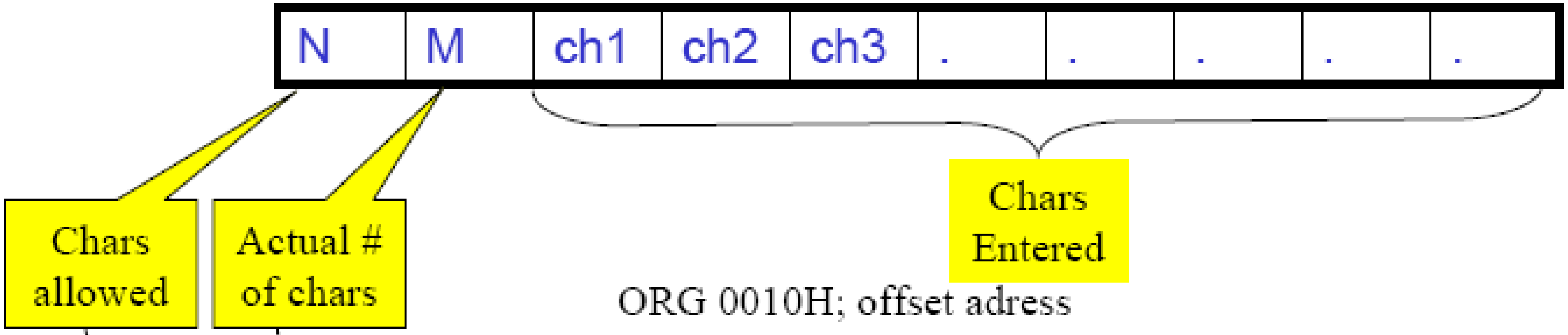

[■]
C:\
10:15
F1 Help F2 Save F3 Open Alt-F3 Close F5 Zoom F6 Next F10 Menu

```
.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
```

```
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>
```

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



N	M	ch1	ch2	ch3	.	.	.	.	.

Chars allowed

Actual # of chars

Chars Entered

```
        ORG 0010H; offset adress
DATA1   DB 6,?,6 DUP(FF)

        MOV AH, 0AH
        MOV DX, OFFSET DATA1
        INT 21H
```

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

INT 16h Keyboard Services

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

```
MOV AH, 01
INT 16h
```

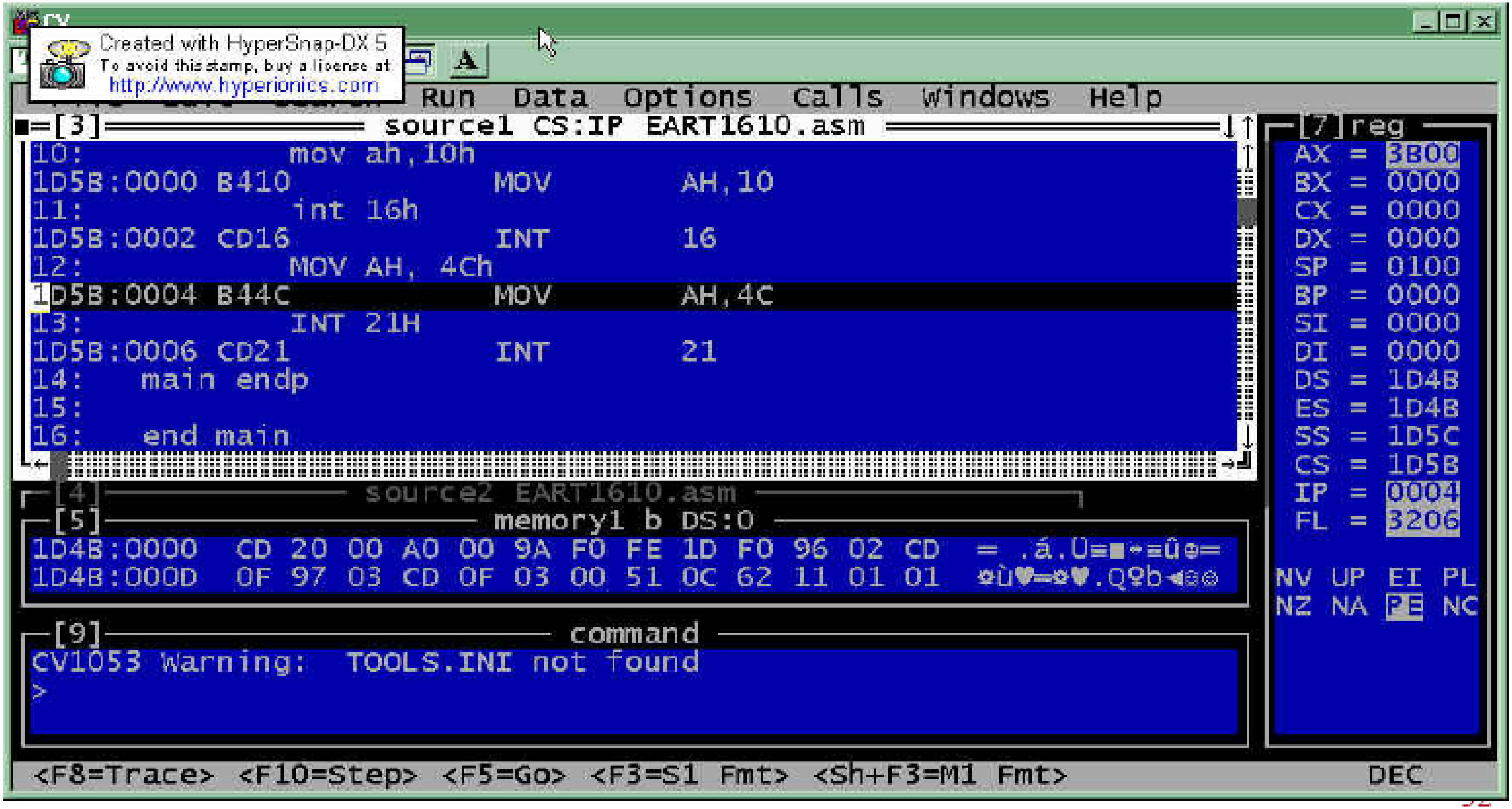
- Upon return, ZF = 0 if there is a key press; ZF = 1 if there is no key press
- Whick key is pressed?
- To do that, INT 16h function can be used immediately after the call to INT 16h function AH=01

```
MOV AH,0
INT 16h
```

- Upon return, AL contains the ASCII character of the pressed key

INT 16 – option 10 or 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).



```
[3]  source1 CS:IP EART1610.asm                        [7] reg
10:              mov ah,10h                            AX = 3B00
1D5B:0000 B410             MOV        AH,10            BX = 0000
11:              int 16h                               CX = 0000
1D5B:0002 CD16             INT        16               DX = 0000
12:              MOV AH, 4Ch                           SP = 0100
1D5B:0004 B44C             MOV        AH,4C            BP = 0000
13:              INT 21H                               SI = 0000
1D5B:0006 CD21             INT        21               DI = 0000
14:     main endp                                      DS = 1D4B
15:                                                    ES = 1D4B
16:     end main                                       SS = 1D5C
                                                       CS = 1D5B
[4]  source2 EART1610.asm                              IP = 0004
[5]  memory1 b DS:0                                    FL = 3206
1D4B:0000  CD 20 00 A0 00 9A F0 FE 1D F0 96 02 CD  = .á.Û=■•=û☻=
1D4B:000D  0F 97 03 CD 0F 03 00 51 0C 62 11 01 01  ☼ù♥=☼♥.Q♀b◄☺☺  NV UP EI PL
                                                              NZ NA PE NC
[9]  command
CV1053 Warning:  TOOLS.INI not found
>

<F8=Trace> <F10=Step> <F5=Go> <F3=S1 Fmt> <Sh+F3=M1 Fmt>        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,’$’
```