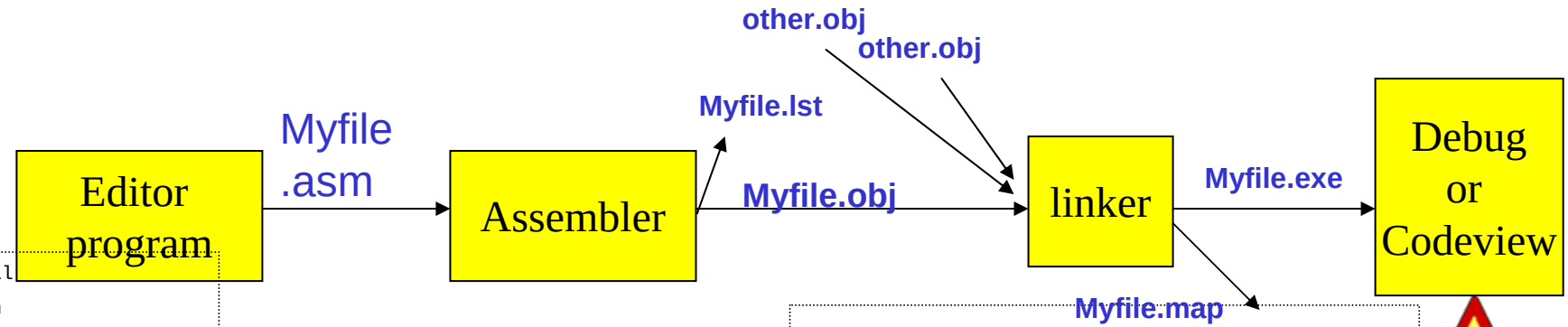
Week 4

8088/8086 Microprocessor Programming

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

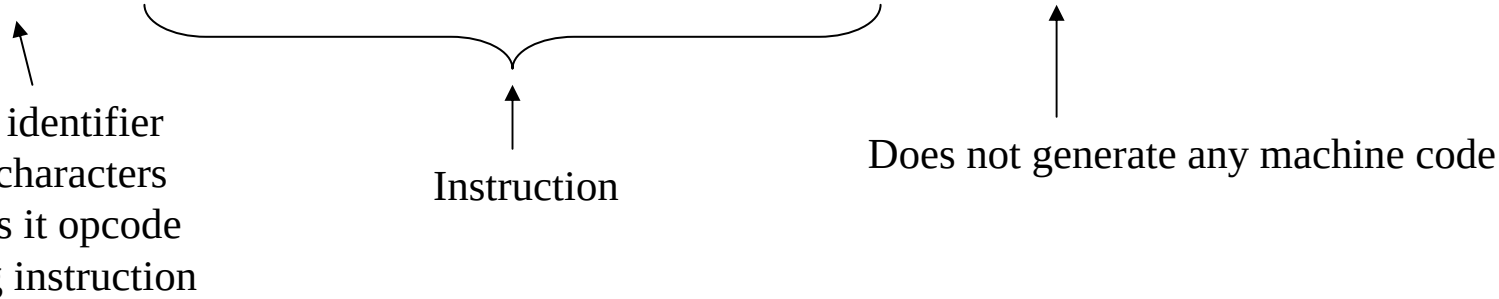



```
.model small
.stack 100h
main proc
                mov ah,01h
                int 21h
                MOV AH, 4Ch
                INT 21H
main endp
end main
```

```
 Start  Stop   Length Name          Class
 00000H 00007H 00008H _TEXT         CODE
 00008H 00008H 00000H _DATA         DATA
00010H 0010FH 00100H STACK          STACK

             Origin   Group
             0000:0   DGROUP

  Program entry point at 0000:0000
```

Instructions

[LABEL:] MNEMONIC [OPERANDS] [; COMMENT]



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

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 0
    int  21h   ; or LEA dx,message will do!
    mov  ax,4C00h ; halt the program and return
    int  21h
main endp
end main
```

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
- Name limit of 247 characters
- Case insensitive
- Variables
  - 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
```

DataTypes and Data Definition

```
DATA1   DB      25
DATA2   DB      10001001b
DATA3   DB      12h
                ORG  0010h
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'
    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 30 00 00
```

More Examples

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

DW  954
DW  253Fh      ; allocates two bytes
DW  253Fh,'HI'

DD 5C2A57F2h   ;allocates four bytes
DQ       "HI"  ; 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 <=64K and Data<= 64K
- Medium Data<=64K code any size multiple code seg
- Compact Code<=64K data any size multiple data seg
- Large 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

- 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?
- Byte or word or doubleword?

- 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?

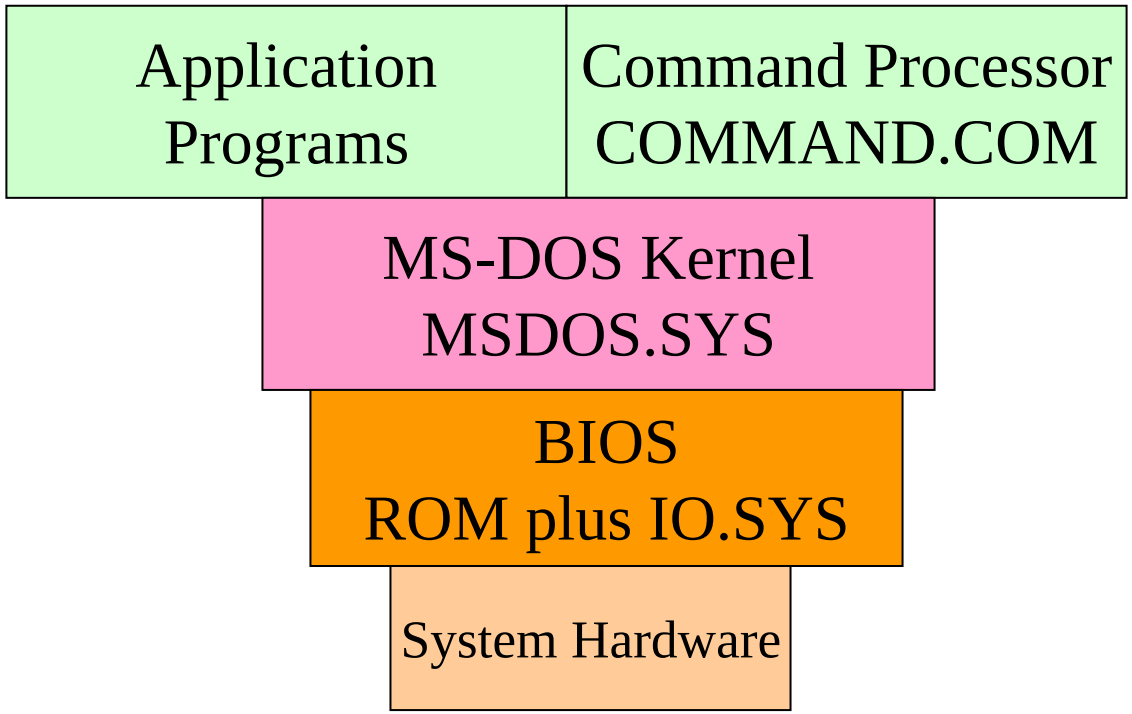
```
MOV AL,BVAL
MOV DL,[BX]
SUB [BX],2
MOV CL,WVAL
ADD AL,BVAL+1

.data
BVAL DB 10H,20H
WVAL DW 1000H
```

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

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

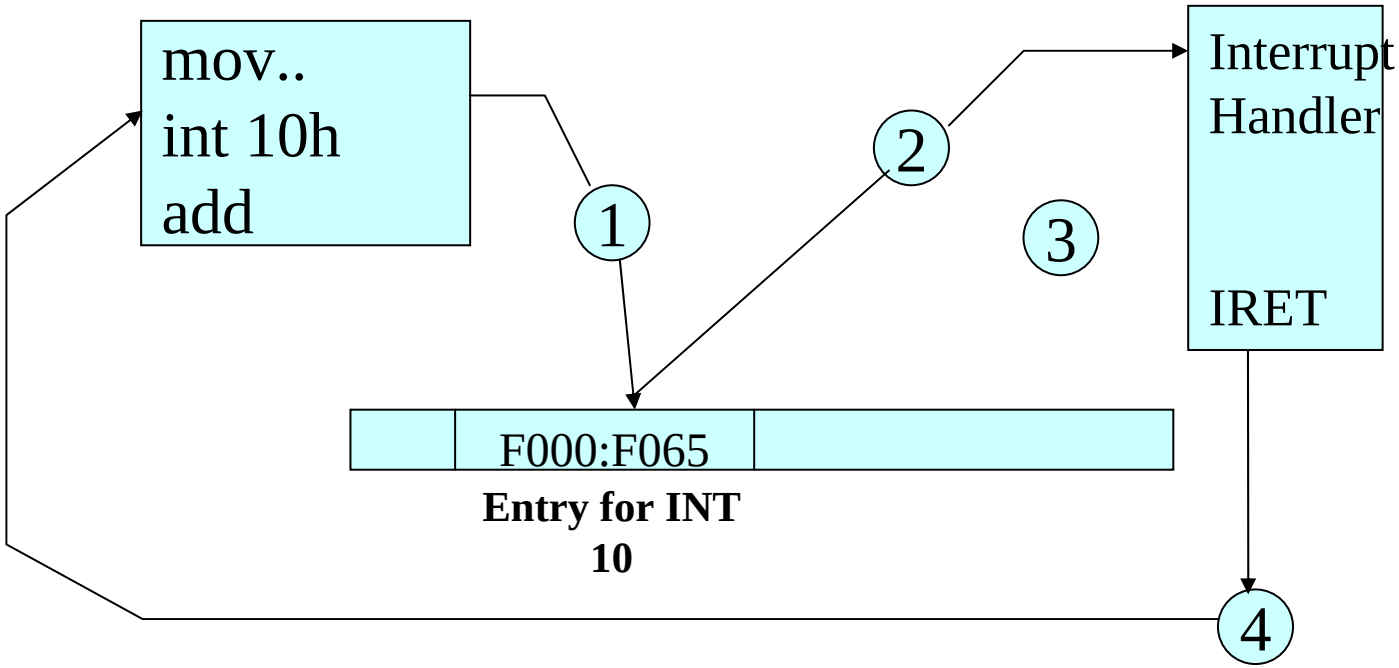
80x86 Interrupts

- 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→ FFh. Before INT is executed AH usually contains a function number that identifies the subroutine.

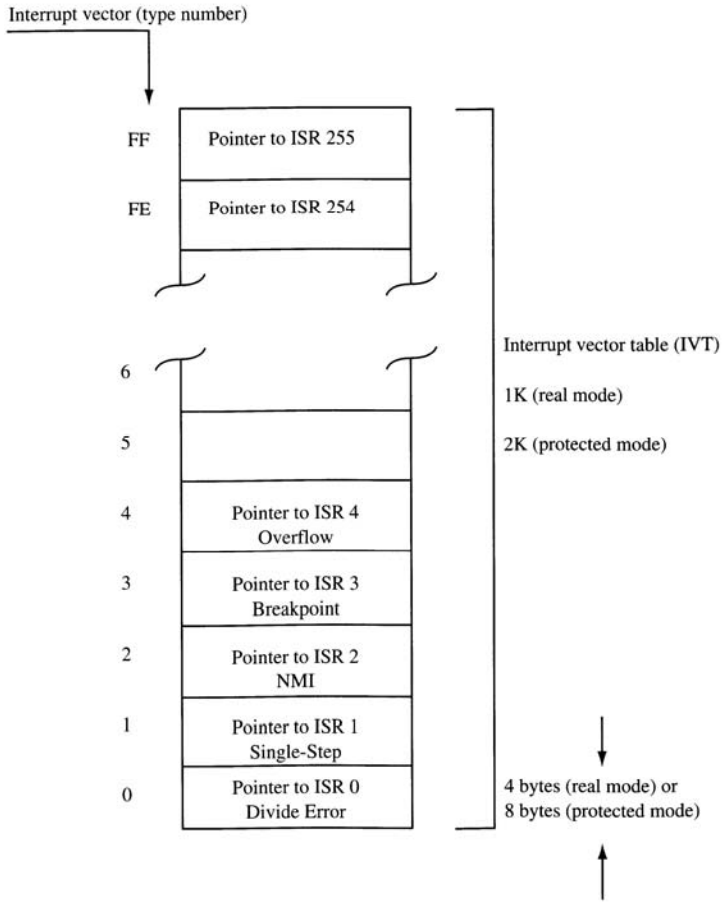
- Each interrupt must supply a type number which is used by the processor as a pointer to an interrupt vector table (IVT) to determine the address of that interrupt's service routine
- Interrupt Vector Table: CPU processes an interrupt instruction using the interrupt vector table (This table resides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



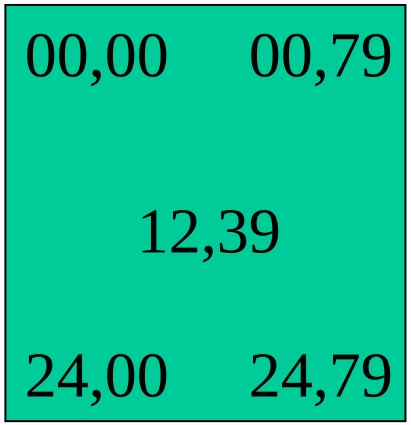
Processor	Pointer Size	IVT Location
Real Mode	4 bytes	Address 00000000–000003FF
Protected Mode	8 bytes	Anywhere in Physical Memory

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

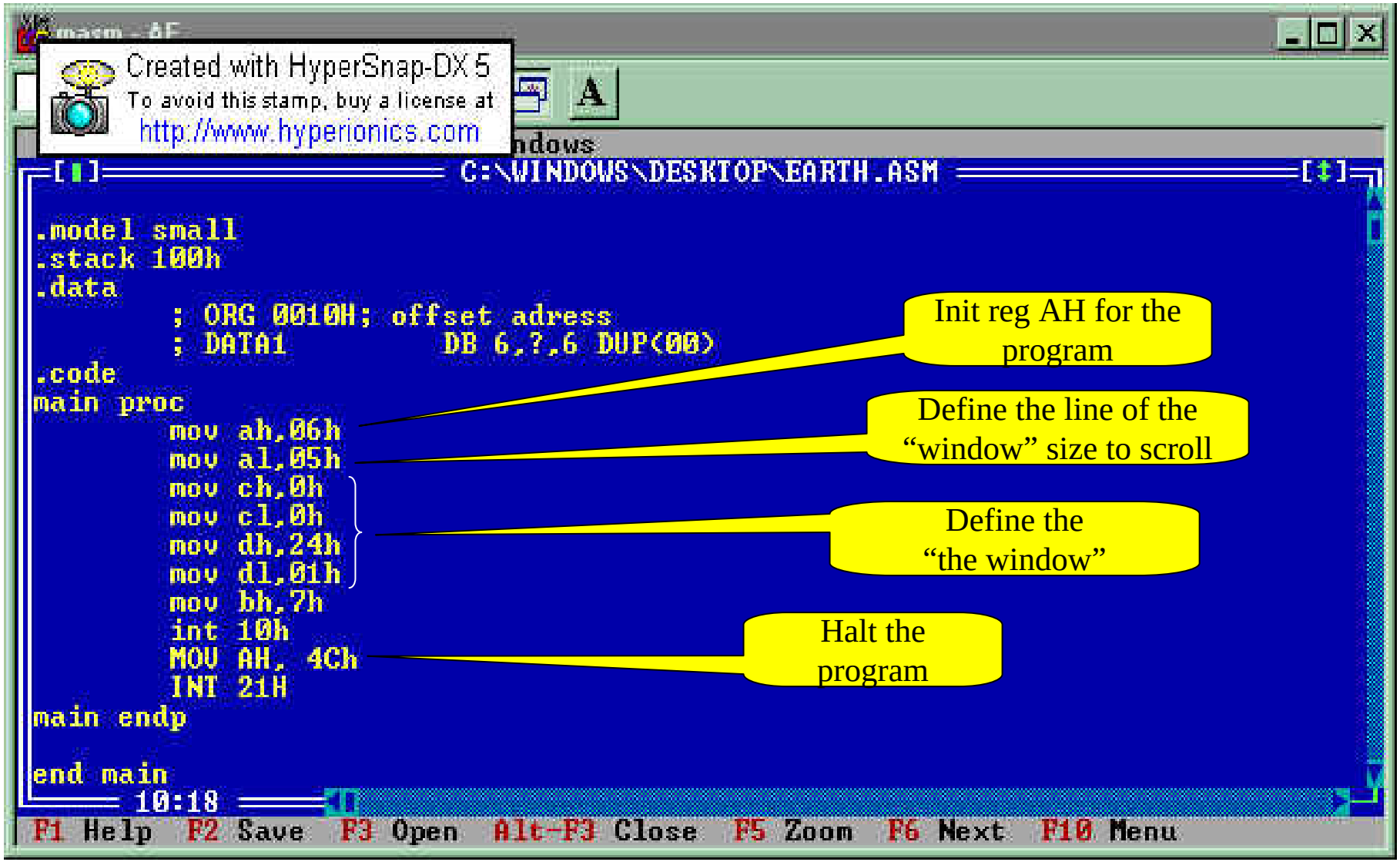


Cursor Locations

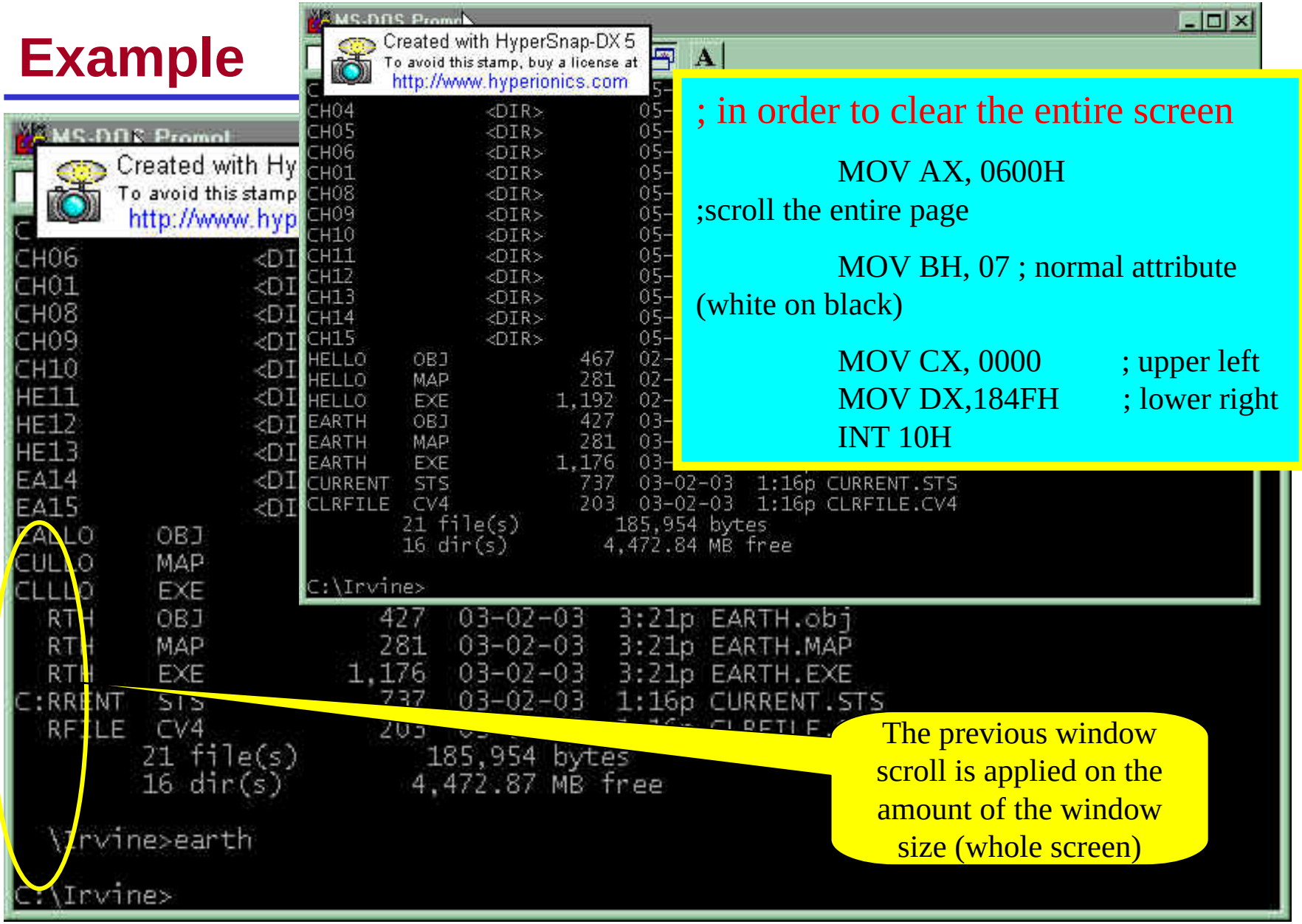
<u>Example:</u> Clear the screen by scrolling it upward with a normal attribute

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

Example Int10 06

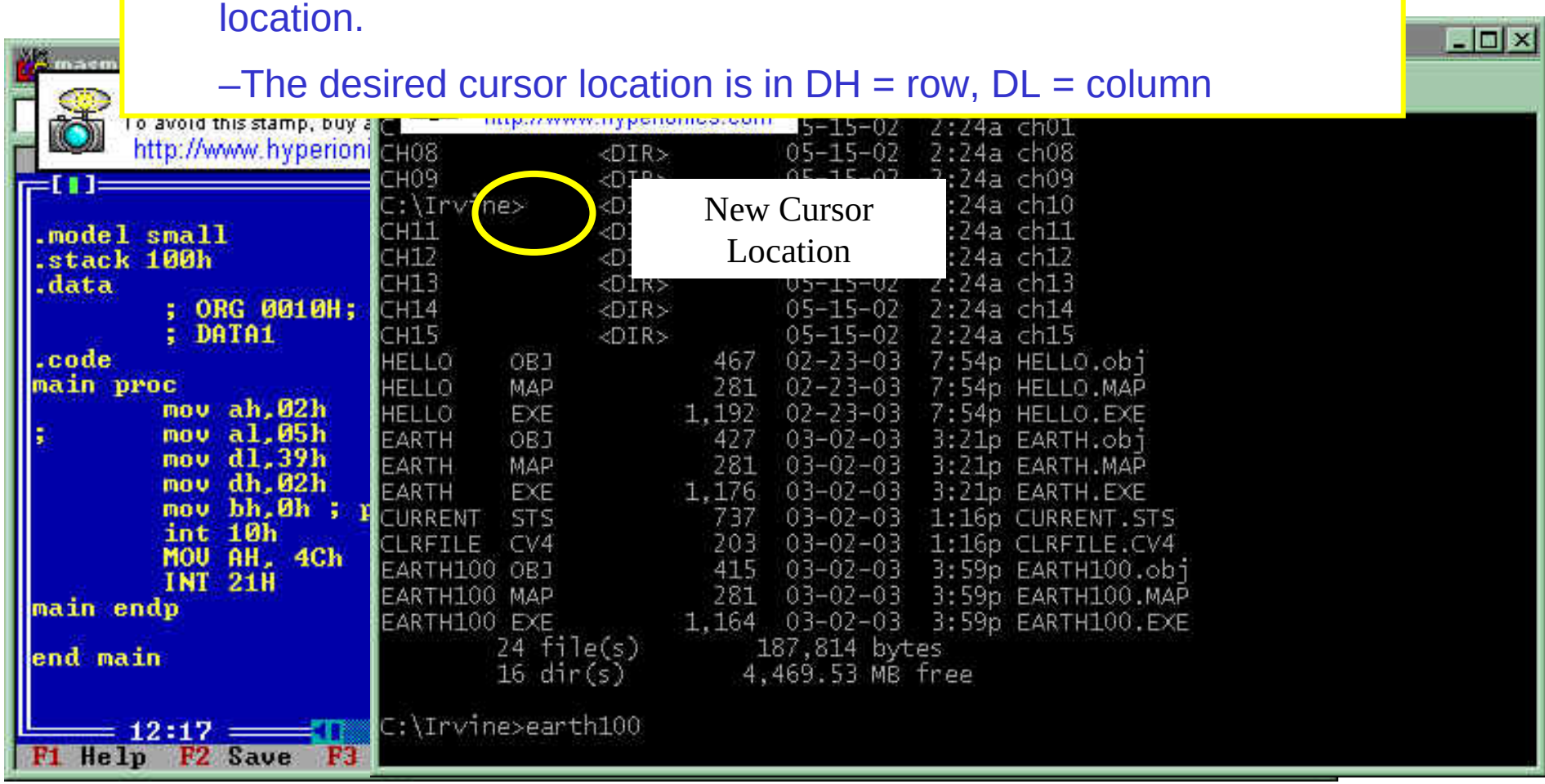


Example


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



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

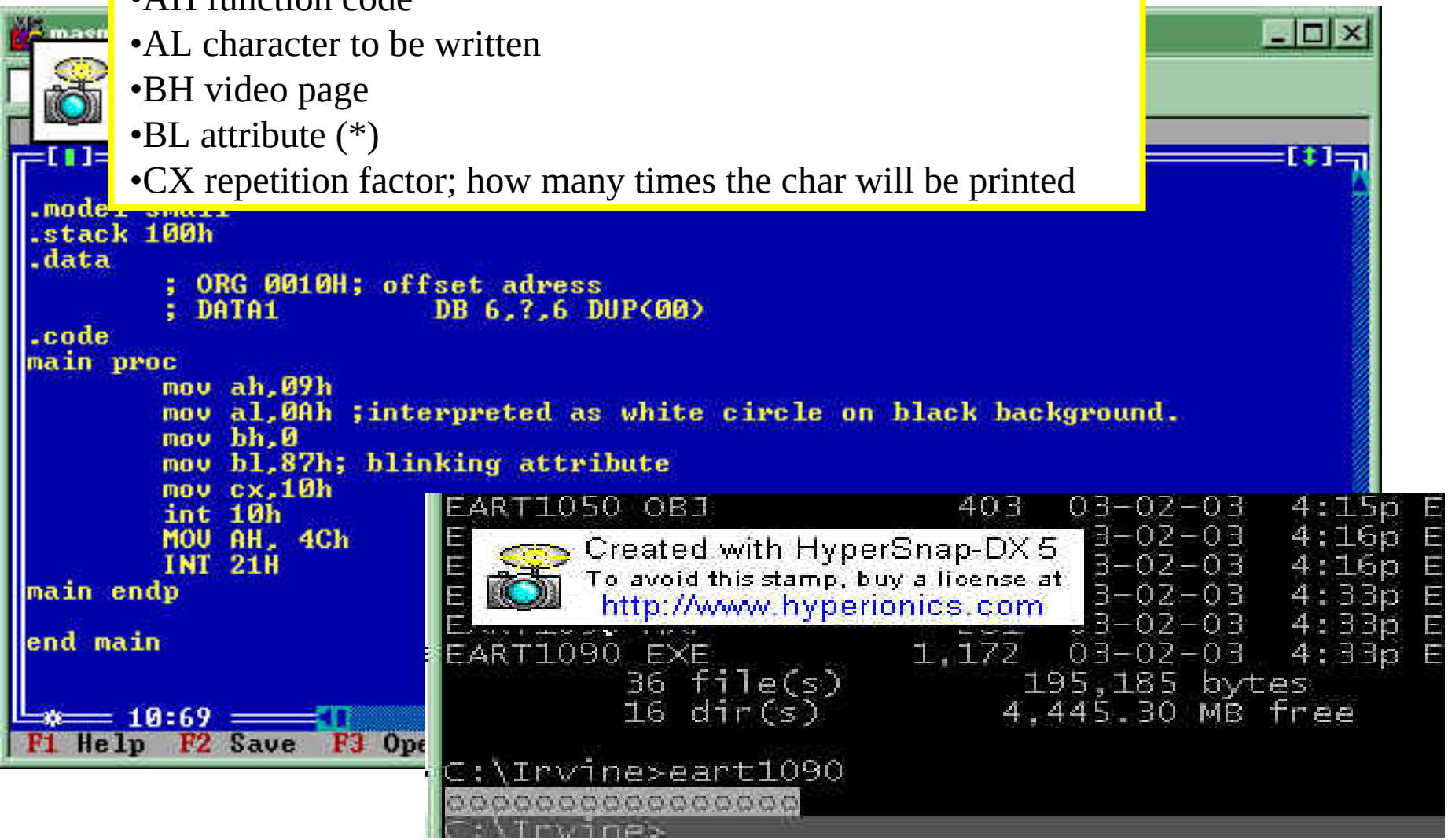
•**INT 10H function 05**; switch between video modes by adjusting AL

```
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
```

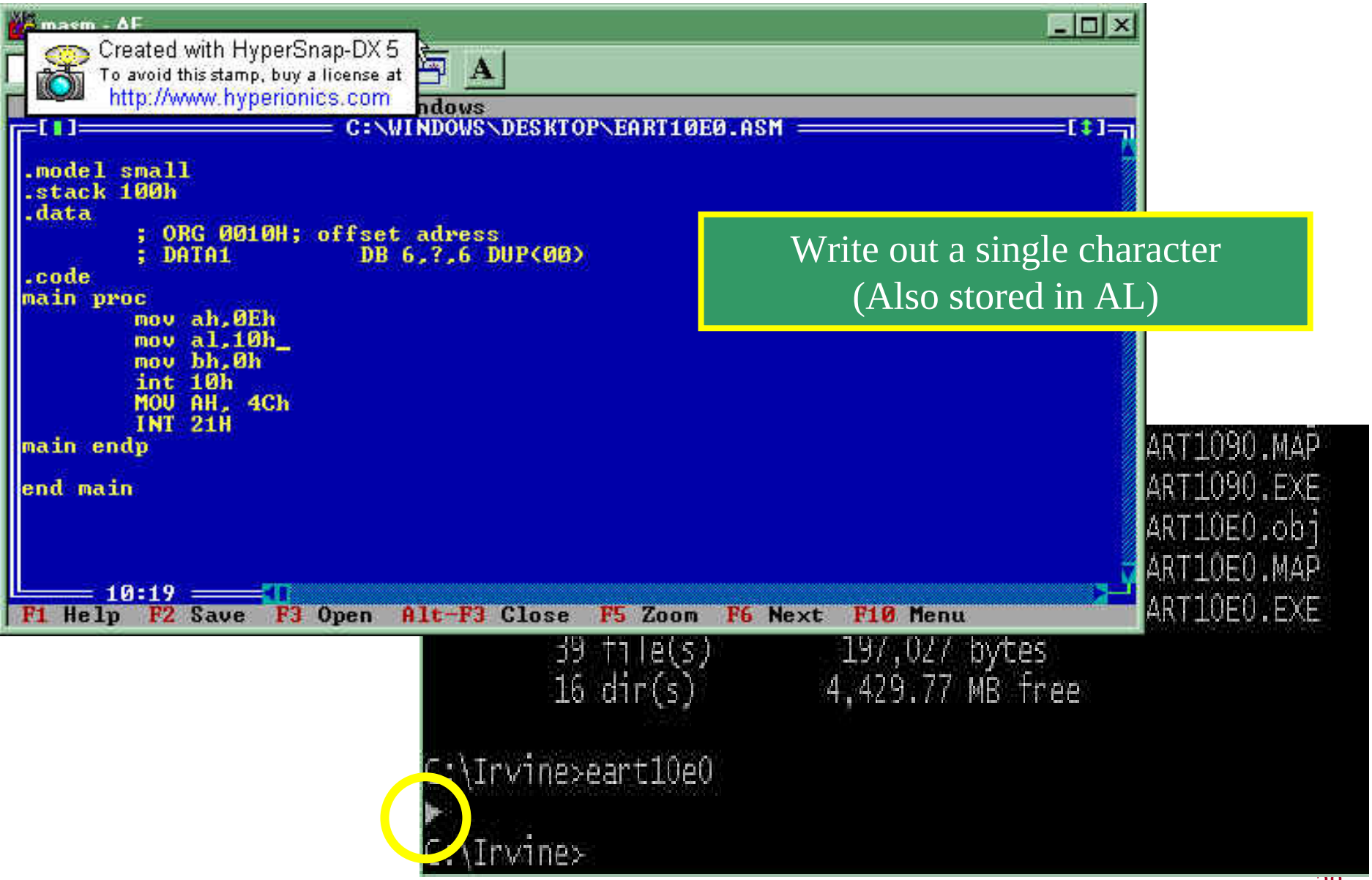
Extremely useful in text modes that support multiple pages!
This is what we had before Windows™

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

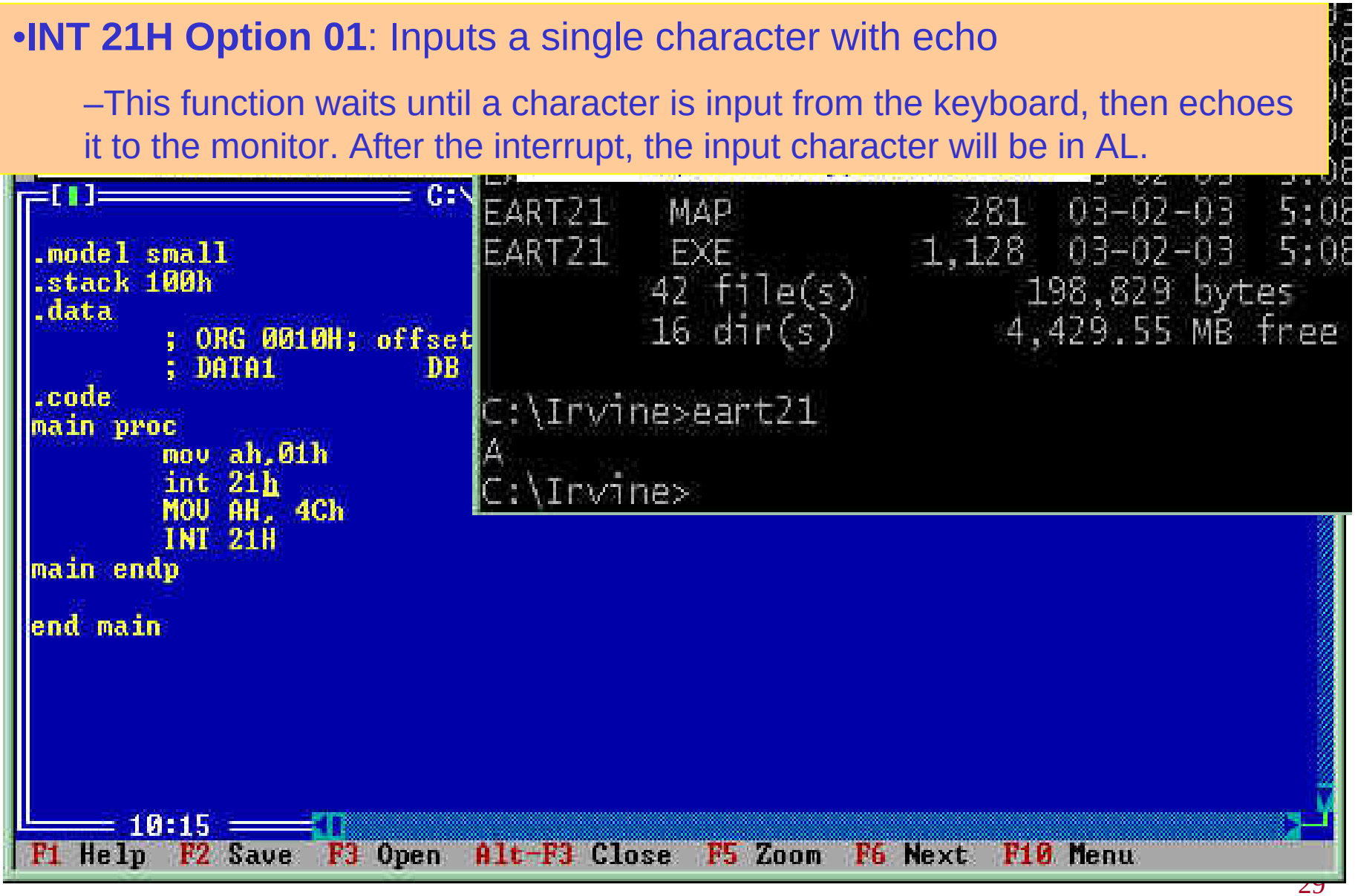


Int 10 - 0e


Created with HyperSnap-DX 5
To avoid this stamp, buy a license at
http://www.hyperionics.com
C:\WINDOWS\DESKTOP\EART10E0.ASM
.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
10:19
F1 Help F2 Save F3 Open Alt-F3 Close F5 Zoom F6 Next F10 Menu
Write out a single character
(Also stored in AL)
ART1090.MAP
ART1090.EXE
ART10E0.obj
ART10E0.MAP
ART10E0.EXE
16 dir(s) 4,429.77 MB free
C:\Irvine>eart10e0
C:\Irvine>

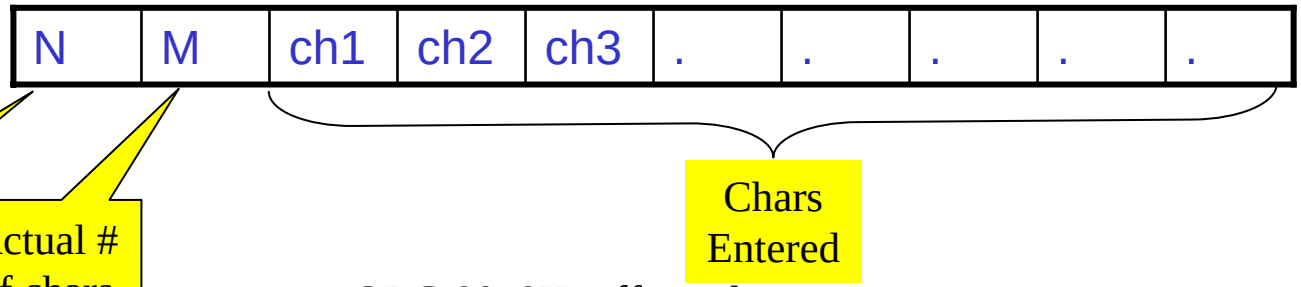
INT 21h

- **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.



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




```
        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
```

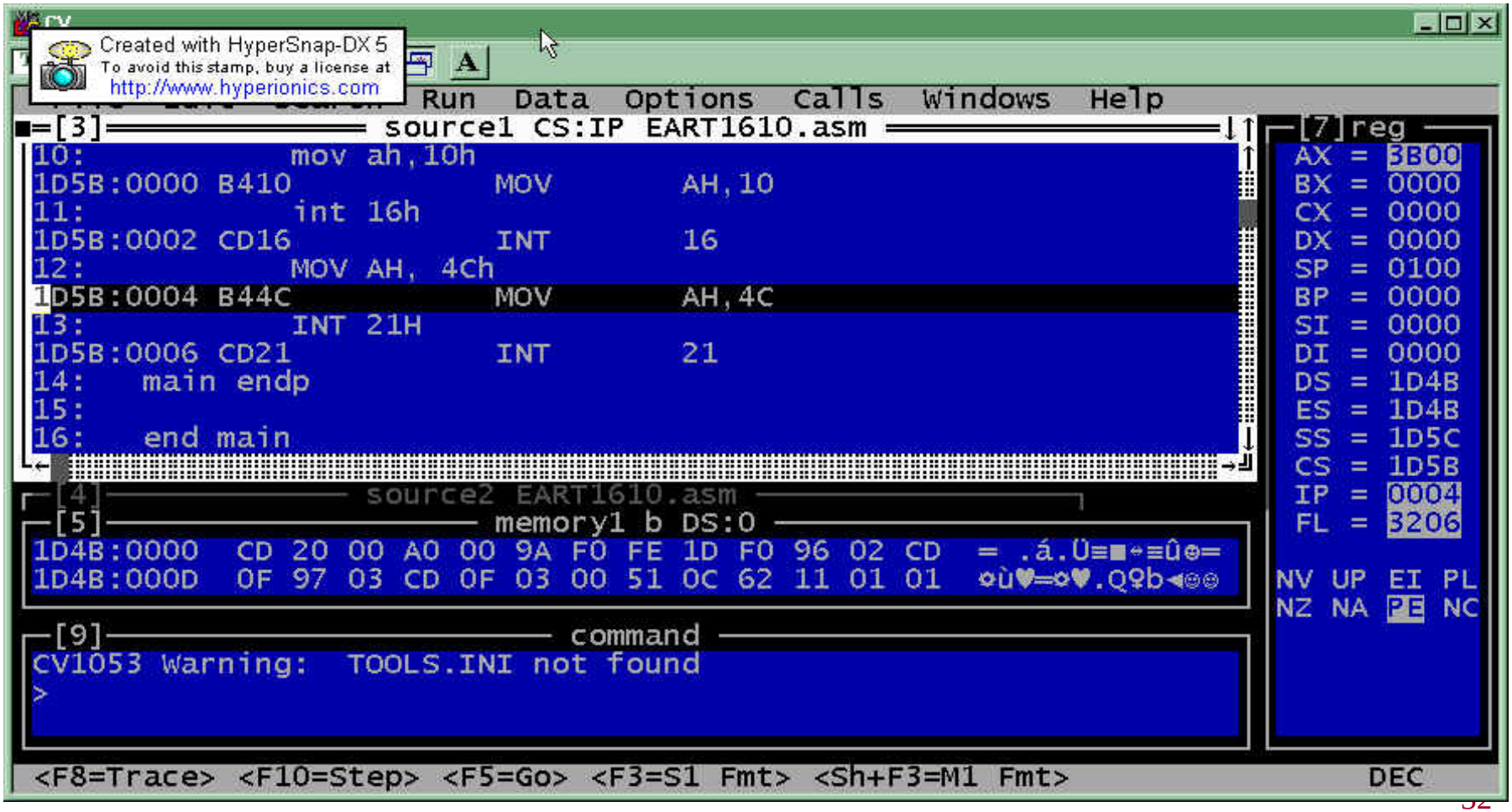
- Upon return, ZF = 0 if there is a key press; ZF = 1 if there is no key press
- Whick key is pressed?
- To do that, INT 16h function can be used immediately after the call to INT 16h function AH=01

```
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).



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) ↔ (Source)	None

Destination	Source
Reg16	Reg16
Memory	Register
Register	Register
Register	Memory

Example: XCHG [1234h], BX

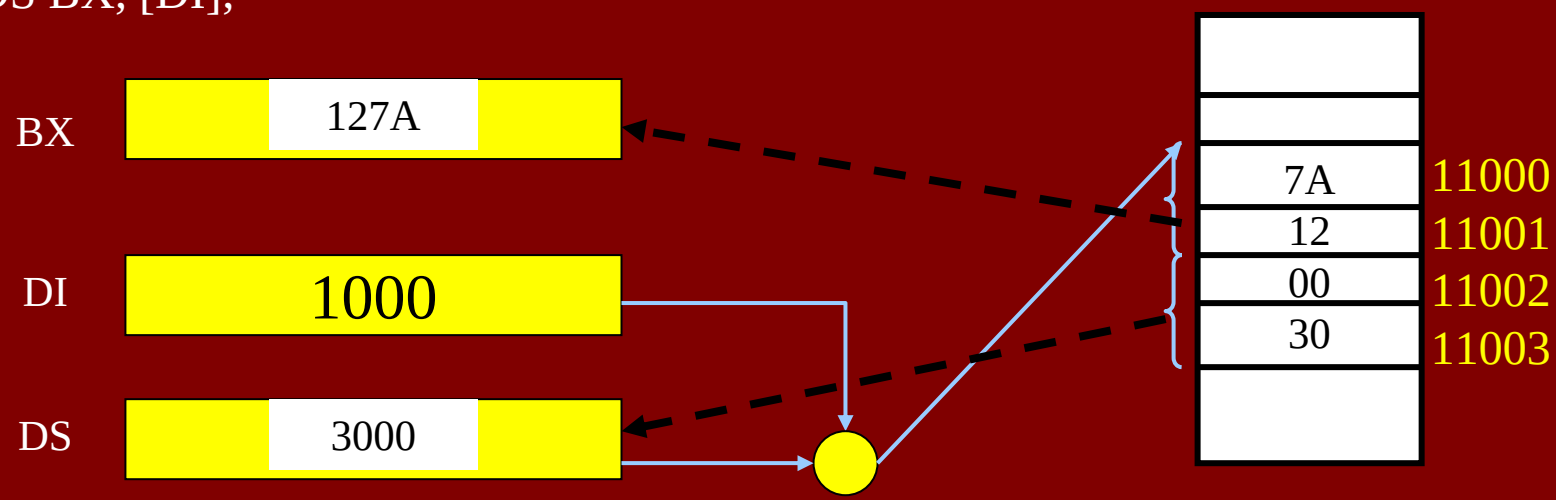
Data Transfer Instructions – LEA, LDS, LES

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) ➔ (D) Carry ➔ (CF)	All
ADC	Add with carry	ADC D, S	(S) + (D) + (CF) ➔ (D) Carry ➔ (CF)	All
INC	Increment by one	INC D	(D) + 1 ➔ (D)	All but CY
AAA	ASCII adjust after addition	AAA	Use AX for the source	AF,CF
AAD	ASCII adjust before! div	AAD	AX has two unpacked BCD before div	
DAA	Decimal adjust after addition	DAA	Adjusts AX 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
ADD AX, 3030H; answer to ASCII → 0107

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

ADD AL, BL
DAA

```
  25
  56
+ ----------
  7B  → 81
```

Ex. 5 MOV BL,9
MOV AX,0702H
; convert to binary first
AAD; 00-99
DIV BL

Example

Write a program that adds two multiword numbers:

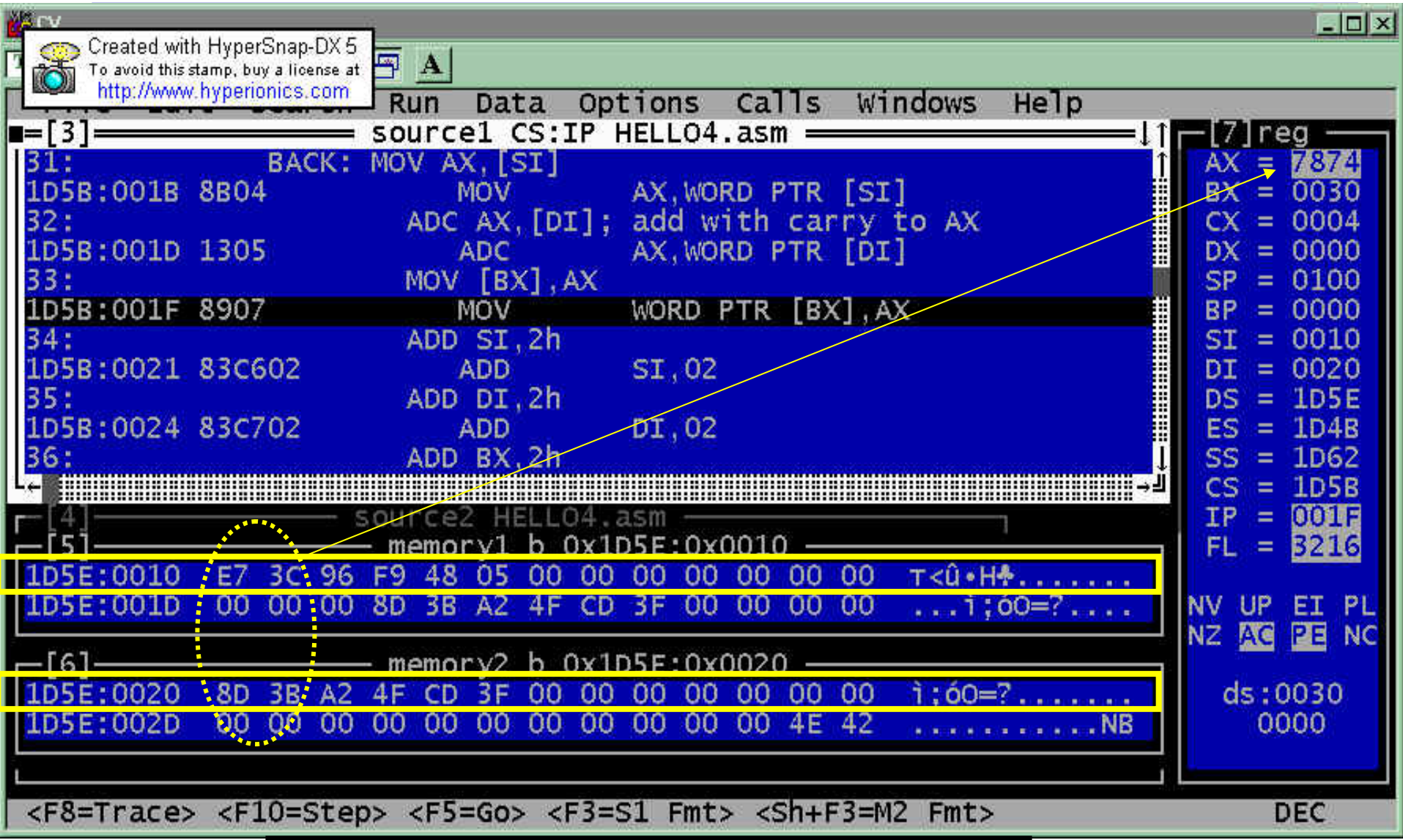
```
.MODEL SMALL
.STACK 64
.DATA
        DATA1 DQ 548F9963CE7h; allocate 8 bytes
ORG 0010h
        DATA2 DQ 3FCD4FA23B8Dh; allocate 8 bytes
ORG 0020h
        DATA3 DQ ?
```

Example Cont'd

```
.CODE
        MAIN PROC FAR
        MOV AX,@DATA; receive the starting address for DATA
        MOV DS,AX
        CLC; clear 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



After 1st word addition

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

Mnemonic	Meaning	Format	Operation	Flags Affected
SUB	Subtract	SUB D, S	(D) - (S) ➔ (D) Borrow ➔ (CF)	All
SBB	Subtract with borrow	SBB D, S	(D) - (S) - (CF) ➔ (D)	All
DEC	Decrement by one	DEC D	(D) - 1 ➔ (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 = (DS:203h)(DS:202h)(DS:201h)(DS:200h)
  - 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