Software Interrupts

Software Interrupts

- A software interrupt is requested by the processor itself upon executing particular instructions or when certain conditions are met
- They do not exactly fit the definition of interrupts given in slide 2.
- Most of these interrupt handlers, provide input-output capability to application programs
- They are used for such tasks as the following:
    - Displaying characters and strings
    - Reading characters and strings from the keyboard
    - Displaying text in color
    - Opening and closing files
    - Reading data from files
    - Writing data to files
    - Setting and retrieving the system time and date

INT Instruction

- Software interrupt are invoked using INT instruction (call to interrupt procedure). It calls an ISR.
- The syntax is
  - INT <number>
  - where number is an integer in the range 0 to FF hex
- Parameters are passed to INT in registers, not on stack. Output is also returned in registers. e.g.
  - mov ax, 0x4c00
  - int 21h
- CPU-generated software interrupts are invoked automatically (e.g. int 0, int 1). So you will never write int 0 in your code.

Example: int 0

- You can run the following code and see that int 0 will occur

  ```
  mov ax, 10
  mov bl, 0
  div bl ; int 0 will be invoked automatically
  mov ax, 0x4c00
  int 21h
  ```

- However you will not be able to debug it and if you run it without debugging you will be stuck in infinite loop
  - This is because int 0 will return to div bl instead of next instruction
- Later, we will hook this interrupt to alter the default CPU response

Operation of INT n instruction

- Steps taken by the CPU when the **INT n** instruction is invoked by a program:
  - The operand of the INT instruction is multiplied by 4 to locate the matching interrupt vector table entry.
  - The CPU pushes the flags and the current CS and IP (return address) on the stack and disables hardware interrupts (by clearing interrupt flag and trap flag)
  - The CPU Loads segment of interrupt n (n*4) from IVT to IP and segment to CS (n*4+2)
  - The interrupt handler at new segment and offset executes until it reaches an IRET (interrupt return) instruction
  - The IRET instruction pops the flags and the return address off the stack, causing the processor to resume execution immediately following the INT n instruction in the calling program.

Common Software Interrupts

• INT 0	Division by zero
• INT 1	Trap, Single step interrupt
• INT 2	NMI: Non maskable interrupt
• INT 3	Debug Interrupt
• INT 4	Arithmetic overflow, change of sign bit
• INT 10h	Video services
• INT 16h	Keyboard services
• INT 1Ah	Time of day
• INT 17h	Printer services
• INT 1Ch	User timer interrupt
• INT 21h	MS-DOS services

Multiple services via one Interrupt

- Many of the software interrupts provide more than one functionality (or service).
- Each service is then identified by a **service number** (or function number).
- Service number is usually passed in via AH register.
- Other arguments, if any, will be passed via other registers

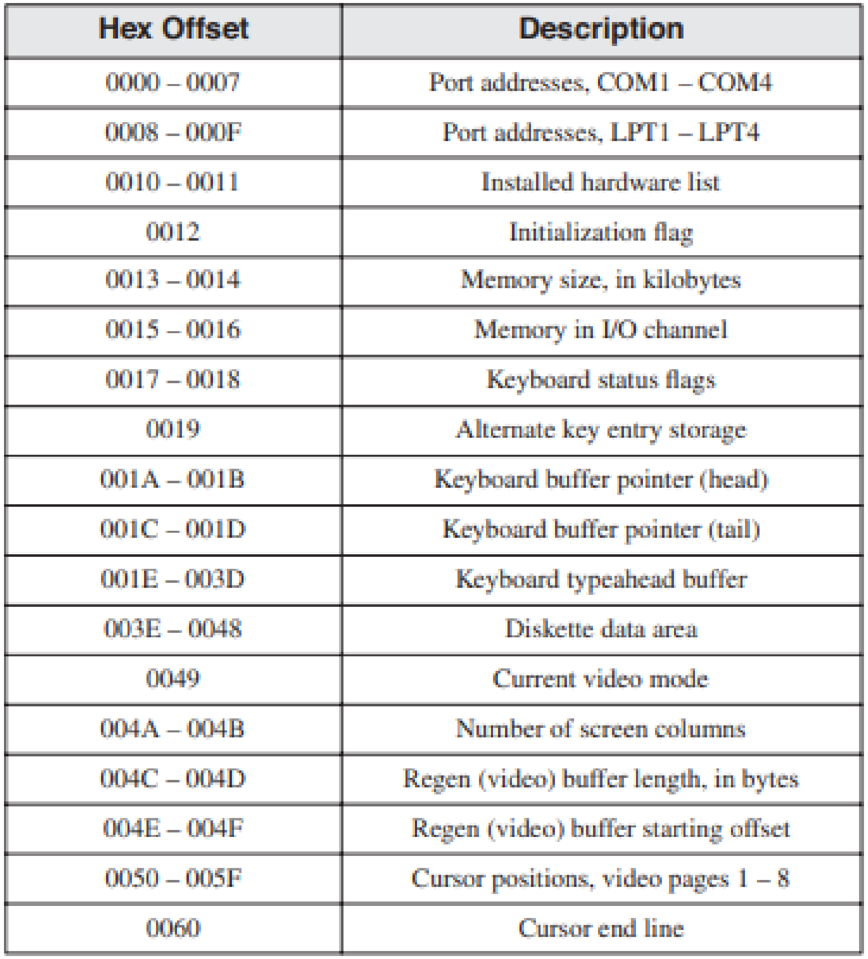
BIOS and DOS interrupts

- In IBM PC there are certain interrupts designated for user programs to communicate with system software to access various standard services like access to the floppy drive, hard drive, VGA, clock etc.
- Since the manufacturer knows the hardware it burns the software to control its hardware in ROM.
- This basic interface to the hardware is called BIOS (basic input output services).

BIOS and DOS data area

- The BIOS data area, partially shown in table, contains system data used by the ROM BIOS service routines.
- For example, the keyboard typeahead buffer (at offset 001Eh) contains the ASCII codes and keyboard scan codes of keys waiting to be processed by the BIOS.

Table 16-1 BIOS Data Area, at Segment 0040h.

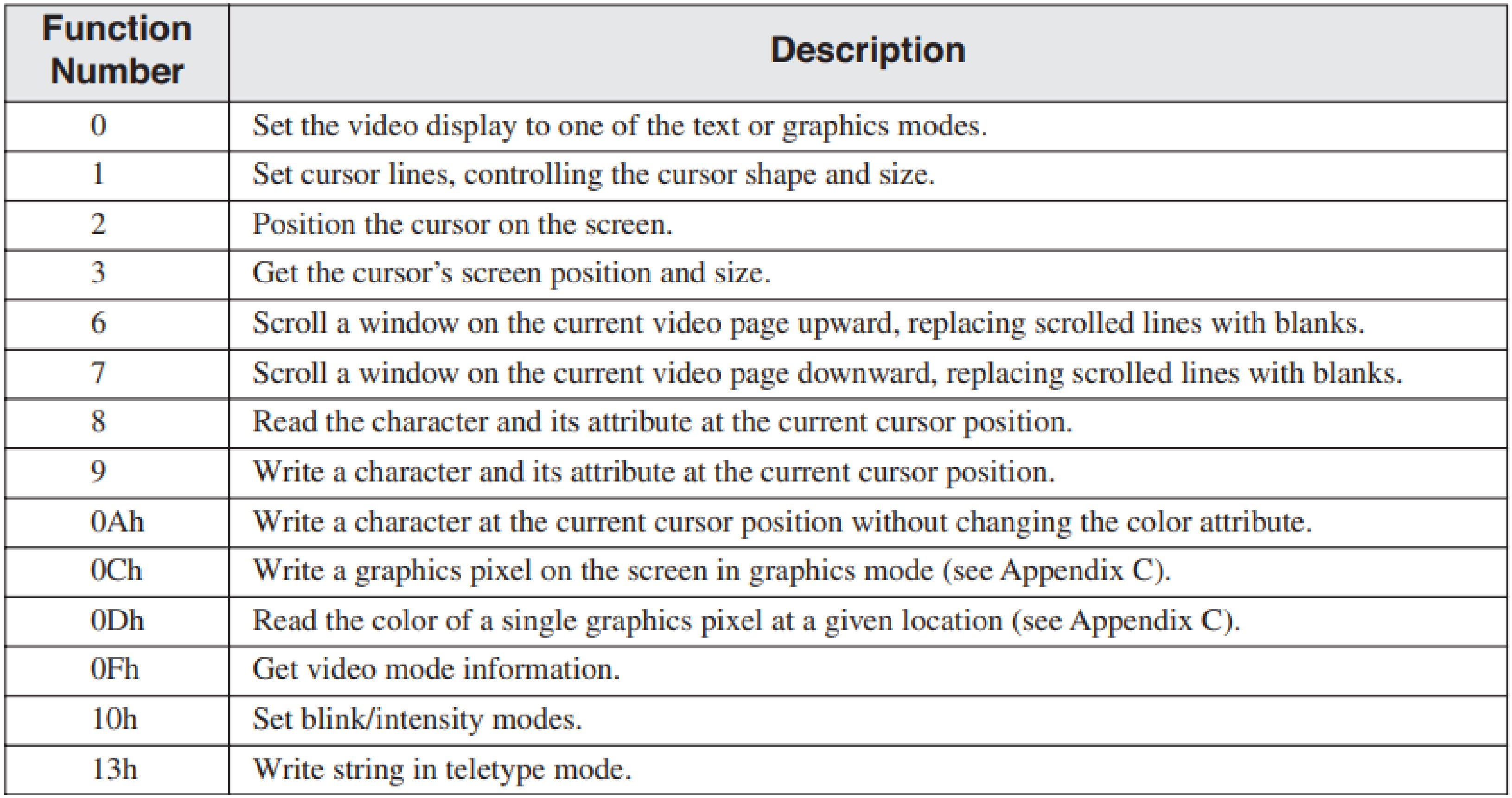


Hex Offset	Description
0000 – 0007	Port addresses, COM1 – COM4
0008 – 000F	Port addresses, LPT1 – LPT4
0010 – 0011	Installed hardware list
0012	Initialization flag
0013 – 0014	Memory size, in kilobytes
0015 – 0016	Memory in I/O channel
0017 – 0018	Keyboard status flags
0019	Alternate key entry storage
001A – 001B	Keyboard buffer pointer (head)
001C – 001D	Keyboard buffer pointer (tail)
001E – 003D	Keyboard typeahead buffer
003E – 0048	Diskette data area
0049	Current video mode
004A – 004B	Number of screen columns
004C – 004D	Regen (video) buffer length, in bytes
004E – 004F	Regen (video) buffer starting offset
0050 – 005F	Cursor positions, video pages 1 – 8
0060	Cursor end line

Video programming using int 10h

- When an application program needs to write characters on the screen in text mode, it can choose among three methods:
  - Direct video memory access – we have done this previously
  - BIOS level access: using int 10h
  - MS-DOS level access: using int 21h
- int 10h is software interrupt provided by BIOS to program video memory
  - int 10h provides several functions/services, depending on the code is given in AH
  - The list of most commonly used functions is given in table next slide

Selected INT 10h functions



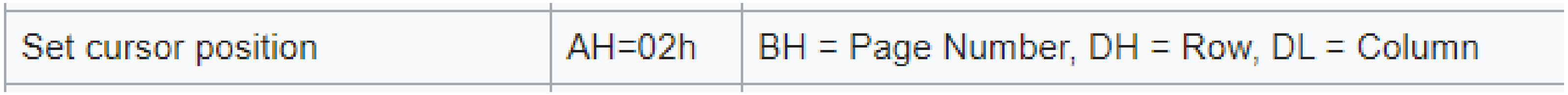
Function Number	Description
0	Set the video display to one of the text or graphics modes.
1	Set cursor lines, controlling the cursor shape and size.
2	Position the cursor on the screen.
3	Get the cursor's screen position and size.
6	Scroll a window on the current video page upward, replacing scrolled lines with blanks.
7	Scroll a window on the current video page downward, replacing scrolled lines with blanks.
8	Read the character and its attribute at the current cursor position.
9	Write a character and its attribute at the current cursor position.
0Ah	Write a character at the current cursor position without changing the color attribute.
0Ch	Write a graphics pixel on the screen in graphics mode (see Appendix C).
0Dh	Read the color of a single graphics pixel at a given location (see Appendix C).
0Fh	Get video mode information.
10h	Set blink/intensity modes.
13h	Write string in teletype mode.

Also see this link https://en.wikipedia.org/wiki/INT_10H

int 10h service 0

- Video modes
  - There are two basic video modes on Intel-based systems, text mode and graphics mode.
  - A program can run in one mode or the other, but not both at the same time:
    - In text mode, programs write ASCII characters to the screen.
    - In graphics mode, programs control the appearance of each screen pixel.
  - A programmer can set the mode using int 10h service 0

int 10h service 02h

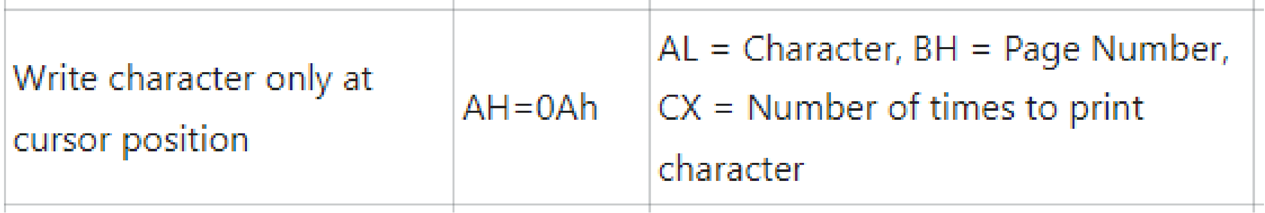


Set cursor position	AH=02h	BH = Page Number, DH = Row, DL = Column

```
mov ah, 2  ; set cursor position service
mov dh, 10 ; row #
mov dl, 20 ; column #
mov bh, 0  ; first video page
int 10h
```

Note: row and column number start from zero.

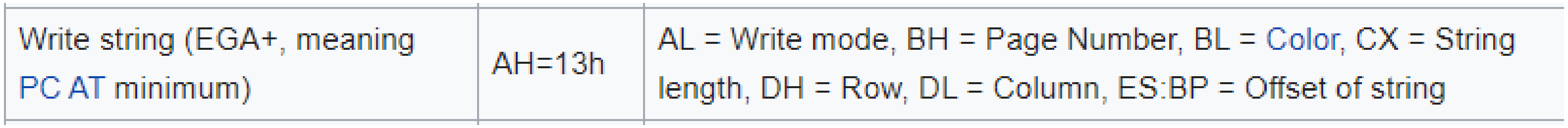
int 10h service 0Ah



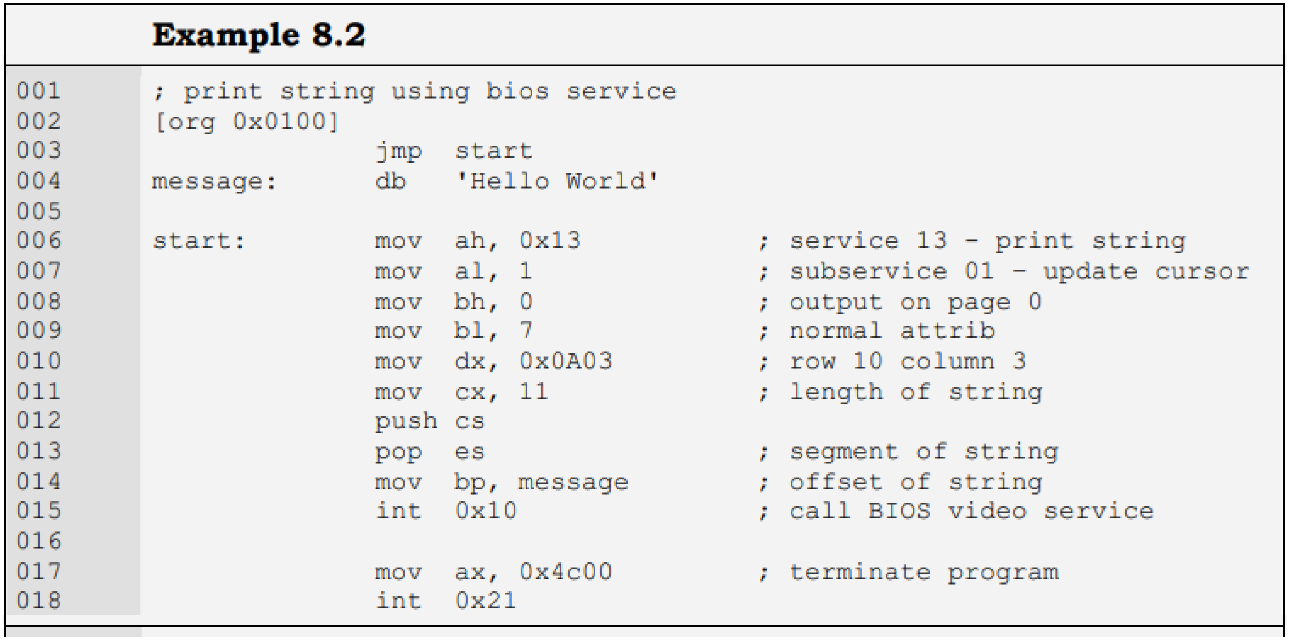
Write character only at cursor position	AH=0Ah	AL = Character, BH = Page Number, CX = Number of times to print character

```
mov ah, 0Ah ; print char service
mov al, 'K' ; character to print
mov bh,  0  ; first page
mov cx, 5   ; number of times to print this char
int 10h
```

int 10h service 13h



Write string (EGA+, meaning PC AT minimum)	AH=13h	AL = Write mode, BH = Page Number, BL = Color, CX = String length, DH = Row, DL = Column, ES:BP = Offset of string



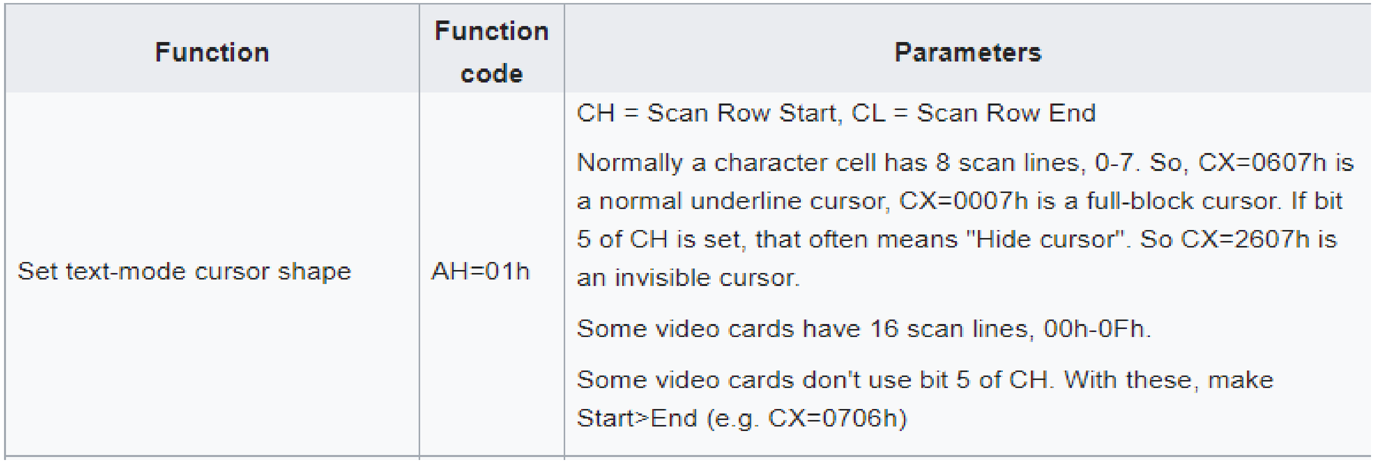
Example 8.2

```
; print string using bios service
[org 0x0100]
              jmp  start
message:      db   'Hello World'

start:        mov  ah, 0x13           ; service 13 - print string
              mov  al, 1              ; subservice 01 - update cursor
              mov  bh, 0              ; output on page 0
              mov  bl, 7              ; normal attrib
              mov  dx, 0x0A03         ; row 10 column 3
              mov  cx, 11             ; length of string
              push cs
              pop  es                 ; segment of string
              mov  bp, message        ; offset of string
              int  0x10               ; call BIOS video service

              mov  ax, 0x4c00         ; terminate program
              int  0x21
```

AL is the write mode in int 10h service 13.

- 0th bit represents if cursor should be updated or not
- 1st bit represents if string contains only characters or (attribute, char) pairs.

int 10h service 01h



Function	Function code	Parameters
Set text-mode cursor shape	AH=01h	CH = Scan Row Start, CL = Scan Row End Normally a character cell has 8 scan lines, 0-7. So, CX=0607h is a normal underline cursor, CX=0007h is a full-block cursor. If bit 5 of CH is set, that often means "Hide cursor". So CX=2607h is an invisible cursor. Some video cards have 16 scan lines, 00h-0Fh. Some video cards don't use bit 5 of CH. With these, make Start>End (e.g. CX=0706h)

- Using function 1 of int 10 to change cursor size

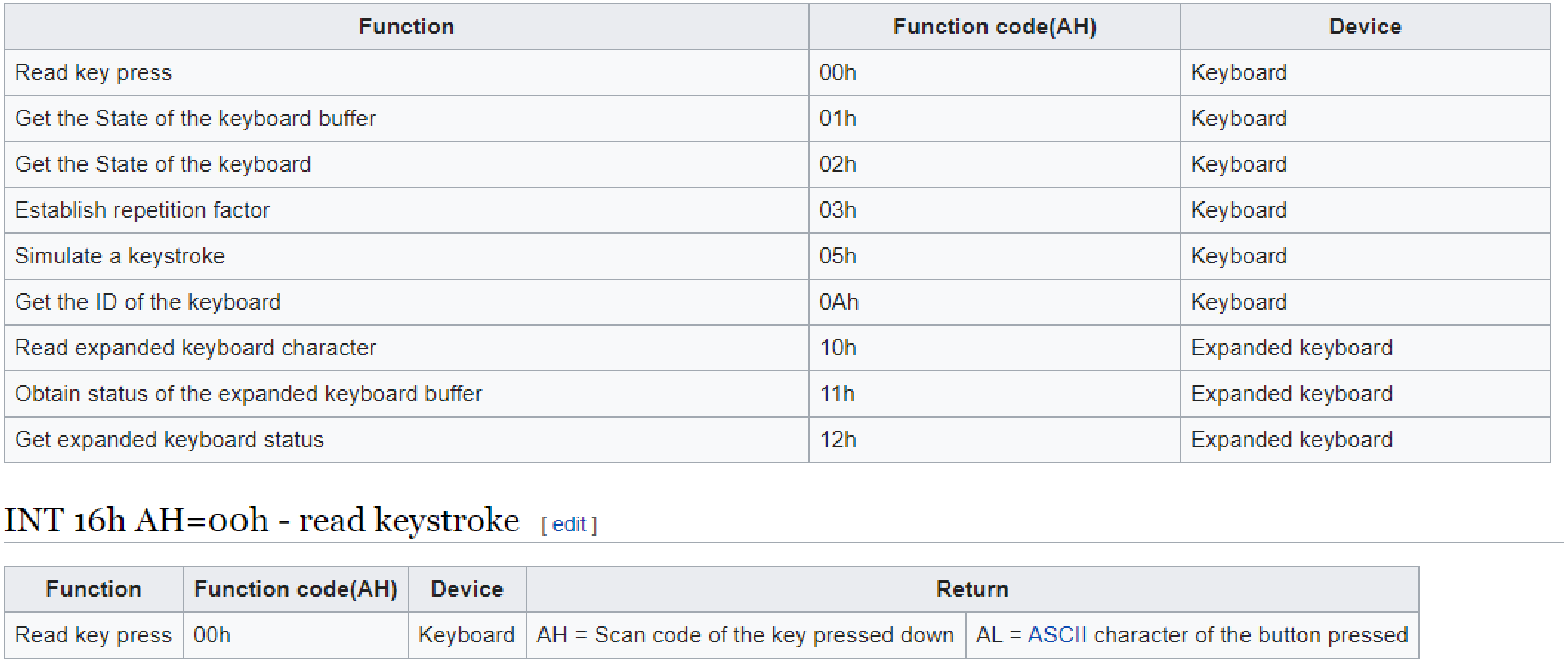
```
mov ah, 1     ; function
mov cx, 0407h ; parameter, thick underline cursor
int 10h
```

- Try different values of CH and CL

Keyboard Input with INT 16h

- int 16h is a software interrupt provided by BIOS to handle keyboard input.
- Keyboard input follows an event path beginning with the keyboard controller chip and ending with characters being placed in an array called the keyboard typeahead buffer.
  - More details later when we see hardware interrupts.
- Up to 15 keystrokes can be held in the buffer because a keystroke generates 2 bytes (ASCII code + scan code).

Services in INT 16h

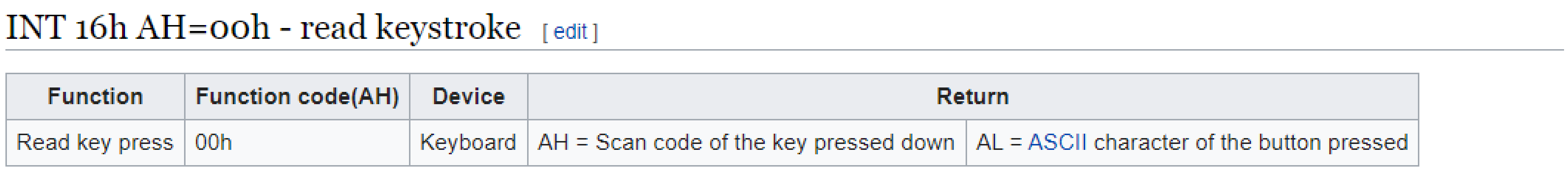


Function	Function code(AH)	Device
Read key press	00h	Keyboard
Get the State of the keyboard buffer	01h	Keyboard
Get the State of the keyboard	02h	Keyboard
Establish repetition factor	03h	Keyboard
Simulate a keystroke	05h	Keyboard
Get the ID of the keyboard	0Ah	Keyboard
Read expanded keyboard character	10h	Expanded keyboard
Obtain status of the expanded keyboard buffer	11h	Expanded keyboard
Get expanded keyboard status	12h	Expanded keyboard

INT 16h AH=00h - read keystroke [edit]

Function	Function code(AH)	Device	Return	
Read key press	00h	Keyboard	AH = Scan code of the key pressed down	AL = ASCII character of the button pressed

Read keystroke using int 16h



INT 16h AH=00h - read keystroke [edit]

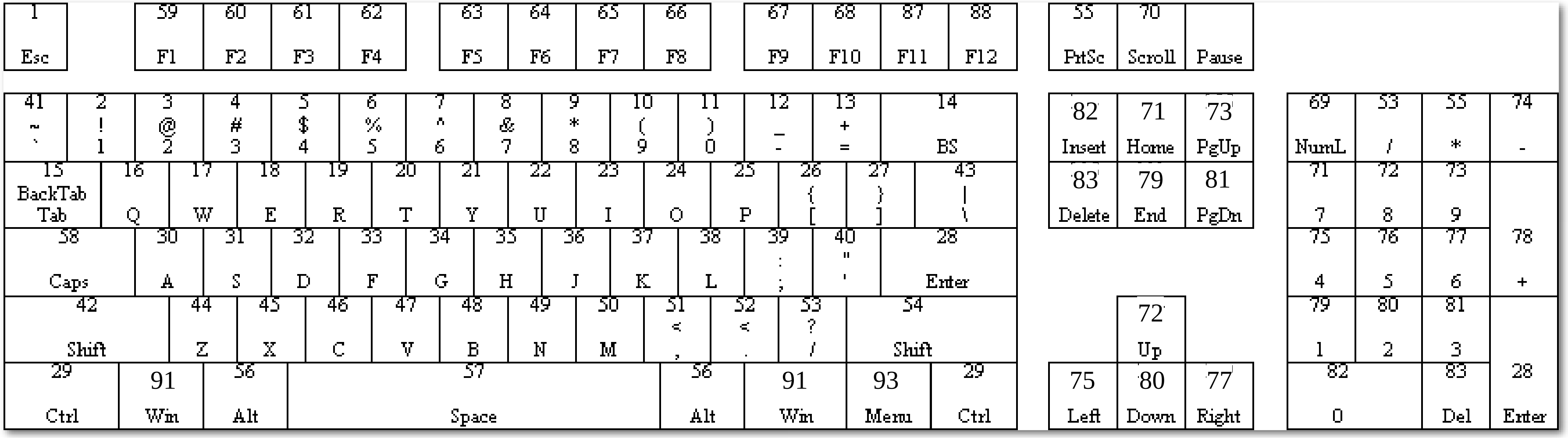
Function	Function code(AH)	Device	Return	
Read key press	00h	Keyboard	AH = Scan code of the key pressed down	AL = ASCII character of the button pressed

```
mov ah, 0
int 16h
```

Keyboard wait using BIOS services

- See example 8.3 in book.

Scan codes of keys

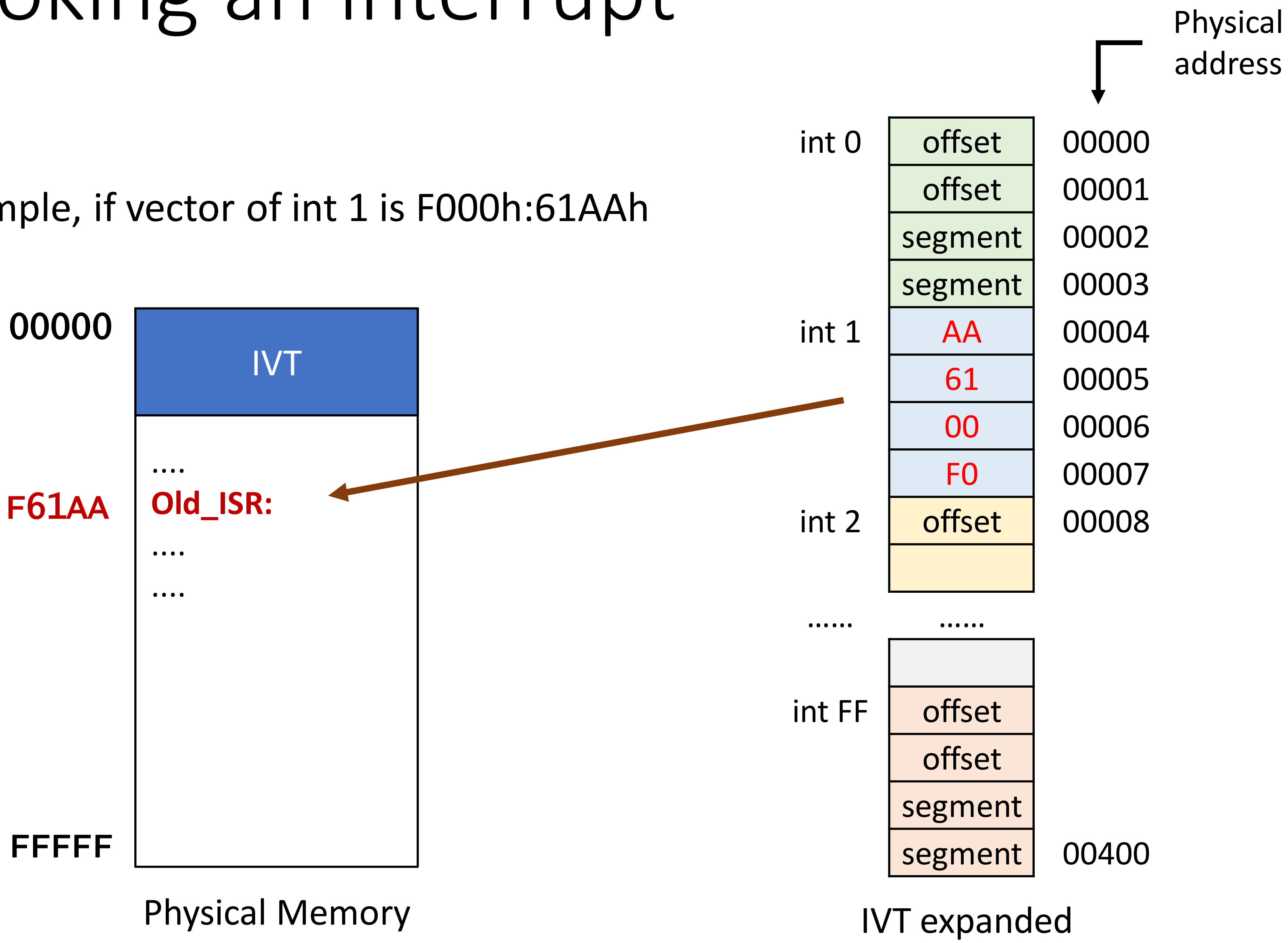


Hooking an interrupt

- In case you want to change the CPU action in response to an interrupt, you write a new handler routine and then hook it to that interrupt.
- To hook an interrupt we change the vector corresponding to that interrupt in IVT.
- As soon as the interrupt vector changes, that interrupt will be routed to the new handler.

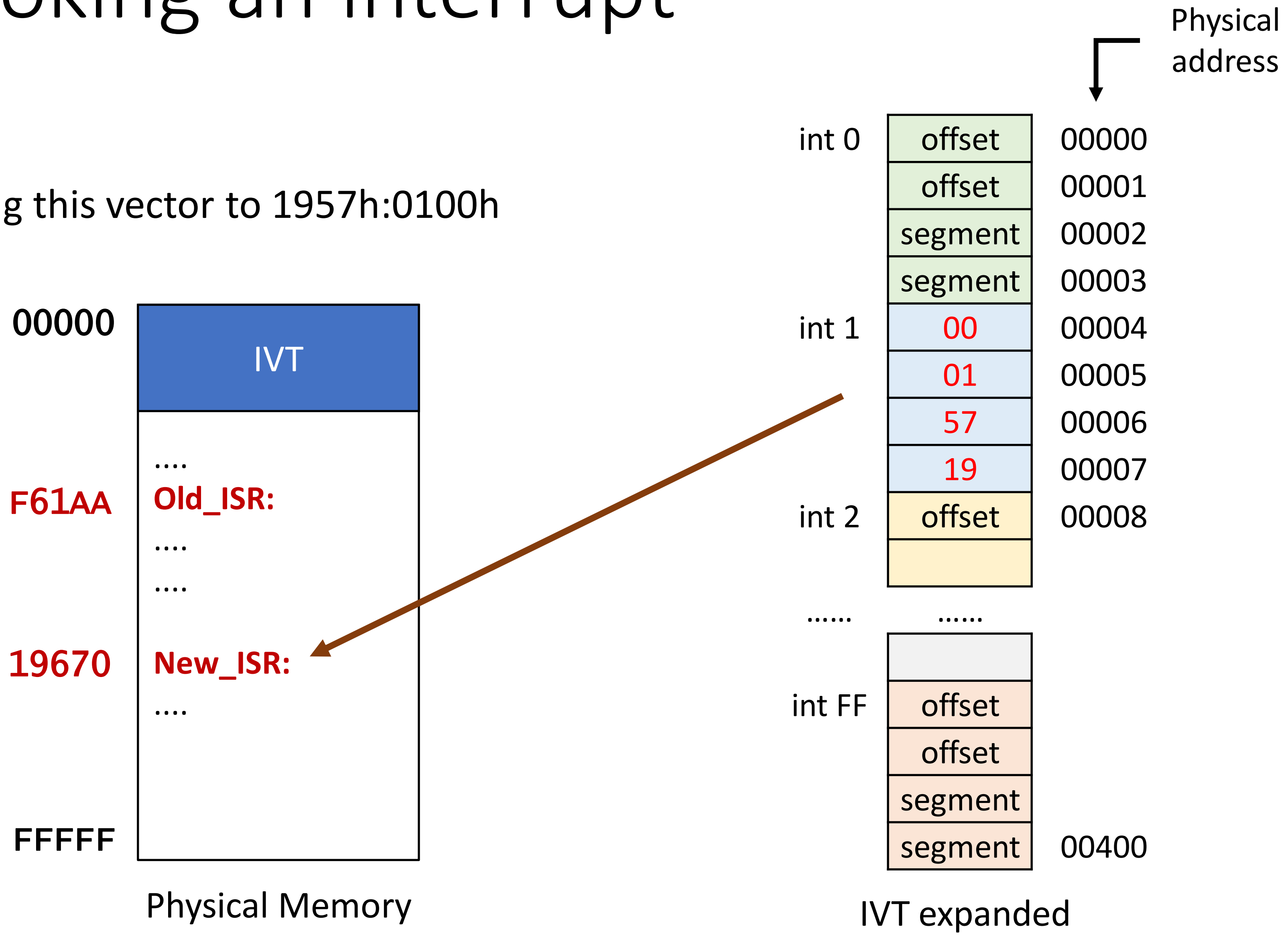
Hooking an interrupt

For example, if vector of int 1 is F000h:61AAh

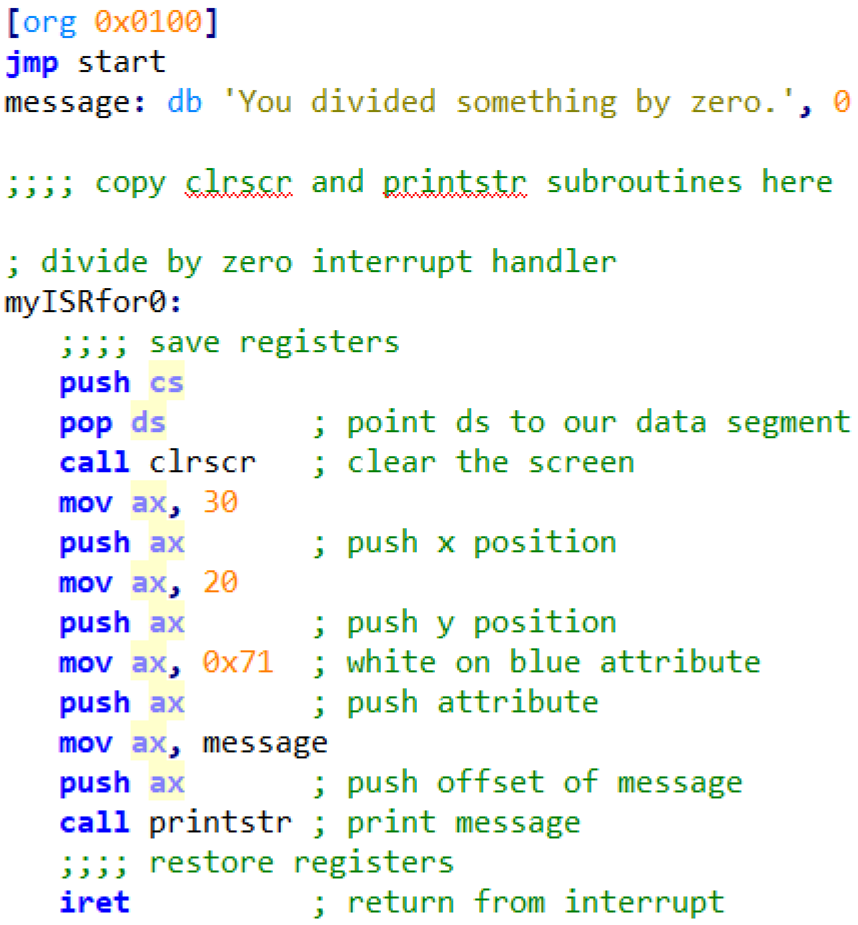


Hooking an interrupt

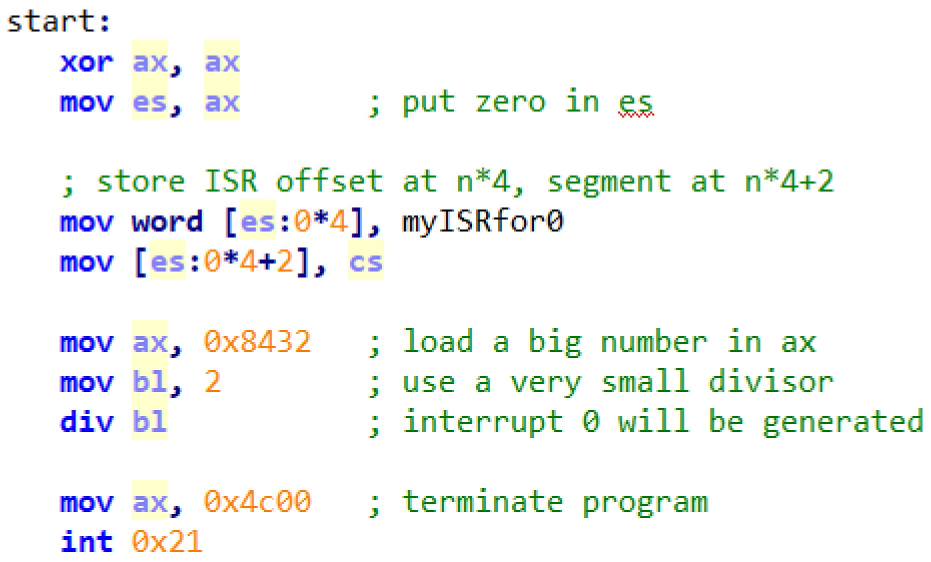
Changing this vector to 1957h:0100h



Example - Hooking int 0



```
[org 0x0100]
jmp start
message: db 'You divided something by zero.', 0

;;;; copy clrscr and printstr subroutines here

; divide by zero interrupt handler
myISRfor0:
   ;;;; save registers
   push cs
   pop ds          ; point ds to our data segment
   call clrscr     ; clear the screen
   mov ax, 30
   push ax         ; push x position
   mov ax, 20
   push ax         ; push y position
   mov ax, 0x71    ; white on blue attribute
   push ax         ; push attribute
   mov ax, message
   push ax         ; push offset of message
   call printstr   ; print message
   ;;;; restore registers
   iret            ; return from interrupt
```



```
start:
   xor ax, ax
   mov es, ax          ; put zero in es

   ; store ISR offset at n*4, segment at n*4+2
   mov word [es:0*4], myISRfor0
   mov [es:0*4+2], cs

   mov ax, 0x8432      ; load a big number in ax
   mov bl, 2           ; use a very small divisor
   div bl              ; interrupt 0 will be generated

   mov ax, 0x4c00      ; terminate program
   int 0x21
```

A short version of example 8.1 BH

Things to note in this example

- int 0 pushes flags, CS and IP on stack
- Pushed IP is of DIV BL, not of next instruction
  - Program will run infinitely as after IRET we will return to DIV BL
  - This is not the case for other interrupts

Template to hook software interrupt n

```
newISR:
  store registers
  <<< body >>>
  restore registers
  IRET


start:
  xor ax, ax
  mov es, ax                      ; load zero in es
  mov word [es:n*4], newISR ; store offset at n*4
  mov [es:n*4+2], cs              ; and segment at n*4+2
```

Documentation for all Interrupts

- Ralf Brown’s Interrupt List

http://www.delorie.com/djgpp/doc/rbinter/ix/

References

- Section 16.2 Irvine
- Chapter 8 BH
- http://jbwyatt.com/253/emu/8086_bios_and_dos_interrupts.html