Software Interrupts

Irvine Edition IV : Section 13.1.4

Software Interrupts

- **Definition** : A software interrupt is a special call to an procedure previously defined as part of the Operating System
  - Alternate Terminology : TRAP, System Call
  - Implemented using a hardware mechanism: interrupt service routine (ISR).
- **Examples** :
  - DOS Functions
    - Print a string message Functions
    - Exit
    - Character Input functions
    - Printer Output functions
  - BIOS Interrupts
    - Video Display
    - Disk I/O functions
    - Keyboard
    - Printers

Application Subroutines versus SW Interrupts

- Subroutines are part of an application program.
  - Developed during program development : assemble, link and load.
    - During assembly, the *assembler* determines the target's address offset
    - During linking, the application program is *linked* with other software …. that has also been assembled.
    - During loading, the segment values for the application program are initialized (May be different each time)
  - Whenever a change is made any file in the application program and/or a software library, the entire application must be re-built :
    - It must be assembled, linked and loaded again. Why ?

Application Subroutines versus SW Interrupts

- Traps are like subroutines
  - Transfer of control to an encapsulated activity terminated by a return to the invocation point.
- Traps are different than subroutines
  - They are not part of the application program
  - They are not part of the program development process
    - The OS is not a software library.
    - Your program's OBJ file are not linked with OS OBJ files.

- Key Difference : The O/S is a permanent resident of memory whereas the application is temporary resident

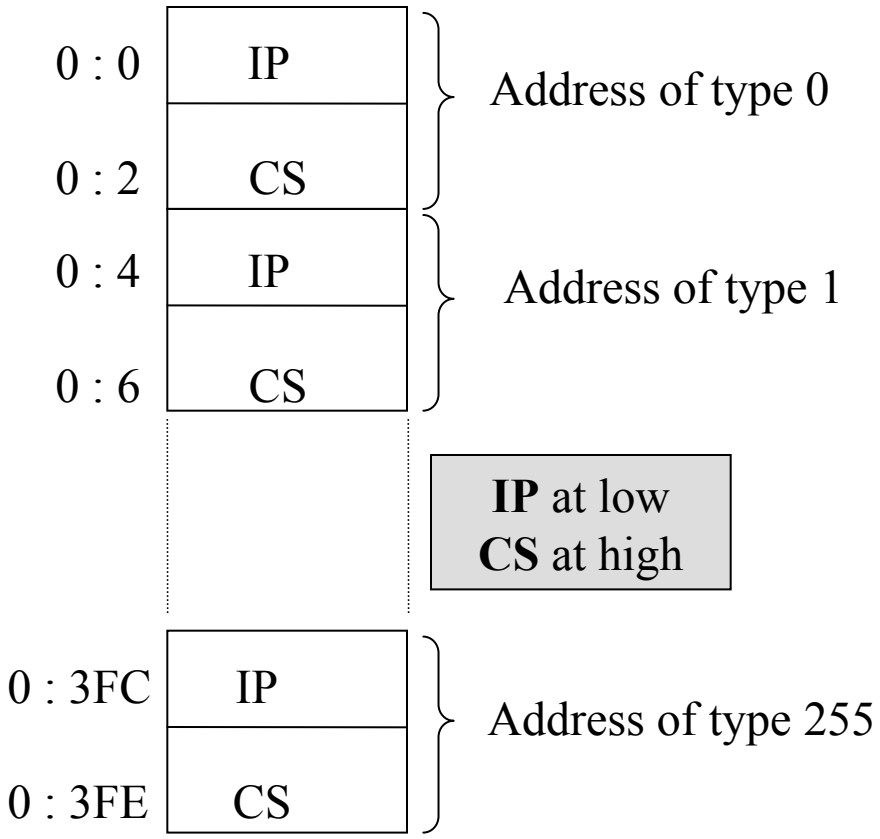
An application program is NOT LINKED with the OS
How does your program "call" the OS Procedure ?

- **First-Step Solution** : Locate each OS Procedure at "reserved" global locations
  - Locations are published as part of the OS (Contract/Policy)
  - Analogy : Direct memory addressing mode
  - What if the OS procedure is revised ?
- **Second-Step Solution** : Locate an array of addresses at a "reserved" global location containing the addresses of the OS procedures.
  - Analogy : Indirect Memory Addressing.
  - O/S puts pointers to O/S procedures in variables when O/S is loaded
  - Application programs are developed under the assumption that these global variables exist
  - What if the OS procedure is revised ?
- **Question** : Do we already have an array of addresses at a reserved location ?

8086 Vector Table

- An array of 256 entries located at the reserved memory location 0:0
  - Each entry is an address of an interrupt service routine (ISR).
  - The address is a FAR Pointer (CS:IP pair) (32-bits = 4bytes)
  - The array occupies addresses from 0:0 to 0:3FF (256*4)

- Each entry is identified by unique "interrupt-type" (number), ranging from 0 to 255
  - If interrupt-type = i then the offset to relevant entry in the vector table = 0000H + 4 * i



Software Interrupt (or TRAP) Instruction : INT

- Software interrupts are based on the stored pointer approach
  - Because new pointers can be installed, offer "dynamic subroutine invocation"

•Subroutines are identified by their label •Subroutines are invoked with CALL instruction	•SW Interrupts are identified by their interrupt-type •SW Interrupts are invoked with INT instruction

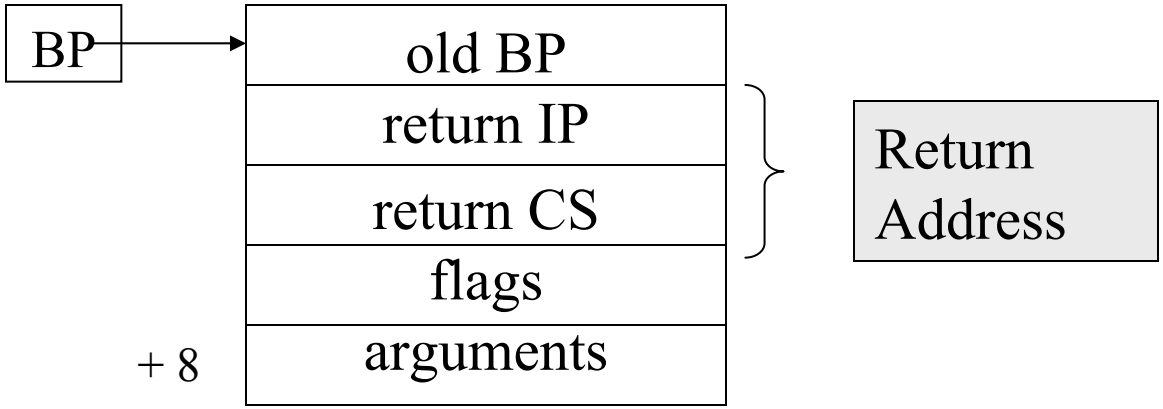
- 8086's INT Instruction
  - It is based on the globally assumed vector table at 0:0
  - Syntax : INT i where i = index into vector table (0..255)
- Example : INT 5
  - Invokes ISR whose FAR address is stored in 6th element of vector table at address = 0:14h (5*4=20)

Software versus Hardware Interrupts

- Software interrupts : invoked by execution of INT instruction
- Hardware interrupts : invoked by event on external I/O device
- **But both share the same execution semantics**
  - Push FLAGS register
  - Clear IF and TF bit in flags
  - Push CS and IP
  - Fetch new CS:IP from 0 : n*4 + 2 and 0:n*4
- Both thus share the same stack frame

Can we pass parameters to an ISR for a HW interrupt ? For a SW interrupt ?

- Is it different than passing parameters to subroutines ?



Subroutines versus Software Interrupts

- Both Subroutines and software interrupts transfer control to an encapsulated activity that is terminated by a return to the invocation point.

```
; subroutine initialization
            none
            . . .
; call set up
            push arguments
            CALL    subr
            ADD     SP,  2*numArgs
            . . .
subr:
            standard entry code
            access param's   [ BP + 4+ ]
            standard exit code
            RET
```

n=0..255

```
; ISR initialization
            install address at 0:4*n
            . . .
; call set up
            push arguments
            INT     n
            ADD     SP,  2*numArgs
            . . .
subr:
            standard entry code
            access param's   [ BP + 8+ ]
            standard exit code
            IRET
```

We've already been using DOS Functions

What can you say about parameter passing to DOS Functions ?

```
.data
message db "Hello, world!", 0dh, 0ah, '$`

.code
   ; Print a string
      MOV     AH, 9
      MOV     DX, OFFSET message
      INT     21h
  ; Exit to DOS
      MOV     AX, 4C00h
  ; AH = 4Ch
  ; AL= 0 (exit status)
      INT     21h
```

INT 21h is the DOS Function Call

How does this one ISR handle all the services?

- The type of service is passed in as a parameter
- But parameters are passed by register, not the stack!!

AH=1	Keyboard input
AH=2	Character output
AH=5	Printer output
AH = 9	String output
AH = 4Ch	Terminate program

- What about ISR return-value?
  - Depends on the service code parameter!

BIOS Function Calls

Other examples : BIOS Interrupts

- INT 10H for video display functions
- INT 13h for disk I/O functions
- INT 16h for keyboard functions
- INT 17h for printers functions

Intel 8086 Vector Table

- The 256 vector table has been mapped out – See Appendix C
- A *brief* look:

Interrupt Types	**Descriptions**
0 … 1F	Reserved by Intel Includes : BIOS 10-16h
20h	Terminate a COM program
21-24h	DOS Functions
33h	Mouse Functions
60-6Bh	Available for Applications
80-F0h	Reserved
F1-FFh	Available for Applications

The 5 Dedicated Interrupt (0..4)

- **Interrupt 0** (divide error)
  - Invoked by the CPU after a DIV or IDIV if the calculated quotient is larger than the destination
  - How big is the quotient if an attempt is made to divide by 0?

- **Interrupt 1** (single step)
  - Used by debuggers to support single stepping
  - If TF flag set, CPU invokes this ISR after executing most instructions
    - TF is cleared as part of the INT execution (after the flags are pushed)
    - Why is TF cleared ?
      - When ISR starts executing, processor is no longer in single-step mode
      - It avoids an infinite loop!

The 5 Dedicated Interrupt (0..4)

- **Interrupt 2** (non-maskable interrupt)
  - Hardware interrupt which cannot be disabled. More later….
- **Interrupt 3** (breakpoint interrupt)
  - A special version of the INT instruction that is encoded in one byte: CEH
  - Used to provide breakpoint capabilities for debuggers
- **Interrupt 4** (overflow interrupt) **INTO**
  - If OF set when INTO instruction is executed, the CPU invokes this ISR
  - Used in numeric libraries to trap overflow errors

- Higher processors (80186, 80286, etc.) have additional dedicated interrupts
  - IBM / Microsoft decided to use interrupts reserved by Intel for their own purposes. It caused problems when the AT was released