5. INTERRUPCIONES Y TRAPS

Para escribir programas en ensamblador se requiere más que instrucciones, ensambladores y ligadores. Un aspecto importante de mencionar en el lenguaje es la entrada y salida de datos, para llevar a cabo esta tarea se utilizan interrupciones.

5.1 DEFINICIÓN Y TIPOS

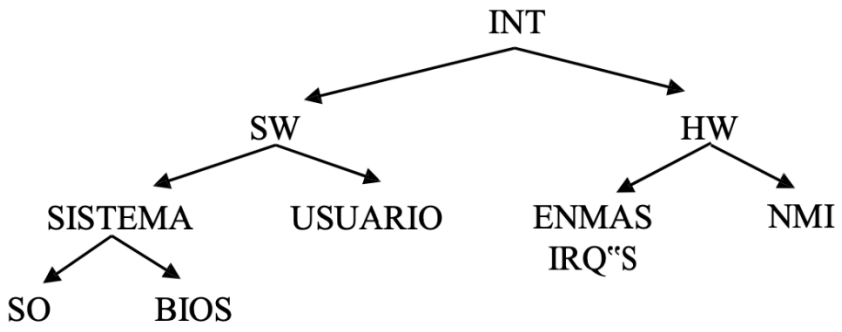
Definición: Una interrupción es el rompimiento en la secuencia de un programa para ejecutar un programa especial llamando una rutina de servicio cuya característica principal es que al finalizar regresa al punto donde se interrumpió el programa.

Existen dos clases de interrupciones:

- ***Interrupciones por software***. Son aquellas programadas por el usuario, es decir, el usuario decide cuándo y dónde ejecutarlas; generalmente son utilizadas para realizar entrada y salida.
- ***Interrupciones por hardware***. Van a ser aquellas que son provocadas por dispositivos externos al procesador, su característica principal es que no son programados, esto es, que pueden ocurrir en cualquier momento en el programa. Existen dos tipos de esta clase de interrupciones:

  Interrupciones por hardware mascarables. Aquellas en las que el usuario decide si quiere o no ser interrumpido.

  Interrupciones por hardware no mascarables (NMI). Aquellas que siempre interrumpen al programa.



Las interrupciones por software se ejecutan con ayuda de las instrucciones: INT e IRET, además se tiene 256 interrupciones: de la 00 a la FF.

En el procesador 8088/86 las instrucciones por software se ejecutan con ayuda de las instrucciones INT e IRET. Se tienen 256 interrupciones diferentes. Desde la interrupción 0 hasta la interrupción 255 (FF).

5.2. USO DE INTERRUPCIONES

Cuando se ejecuta una interrupción, varias acciones se llevan cabo:

Acciones que realiza la instrucción INT.

1. Salvar el registro de banderas
2. Salvar el cs de la dirección de regreso
3. Salvar el IP de la dirección de regreso
4. Calcula el área donde está la dirección de la rutina de servicio tipo*4 en el vector de interrupciones.
5. Ejecuta la rutina de servicio

Acciones que realiza la instrucción IRET

1. Desempila dirección de regreso
2. Desempila banderas

Existen, como ya se mencionó 256 interrupciones:

Tipo	*Dirección*	*Uso*	*Sistema*
0	0000	División por cero	BIOS
1	0004	Ejecución paso a paso	DEBUG
2	0008	NMI	BIOS
3	000C	Puntos de ruptura	DEBUG
4	0010	Overflow	BIOS
5	0014	Print Screen	BIOS
6 – 7		No usadas	
8	0020	Timer	BIOS
9	0024	Teclado	BIOS
A – D		No usadas	
E	0038	Disco	BIOS
F	003C	Impresora	BIOS
10	0040	E/S video	BIOS
11	0044	Lista del equipo	BIOS
12	0048	Tamaño de memoria	BIOS
13	004C	E/S disco	BIOS

14	0050	E/S Serial	BIOS
15	0054	E/S casette	BIOS
16	0058	Entrada Teclado	BIOS
17	005C	Salida impresora	BIOS
18	0060	ROM BASIC	BASIC
19	0064	Boot Strap (reset)	BIOS
1A	0068	Fecha y hora	BIOS
1B	006C	Break (Teclado)	BIOS
1C	0070	Int. de timer	BIOS
1D	0074	Tabla del video	BIOS
1E	0078	Tabla de disco	BIOS
1F	007C	Tabla del video	BIOS
20	0080	Termina programa	DOS
21	0084	Funciones	DOS
22	0088	Dir. De regreso	DOS
23	008C	Control-C	DOS
24	0090	Errores críticos	DOS
25	0094	Lectura absoluta del disco	DOS
26	0098	Escritura absoluta del disco	DOS
27	009C	Termina programa (Deja residente)	DOS

Las interrupciones del BIOS siempre están disponibles al usuario (ROM), en cambio las del DOS solo si el sistema se ha cargado en memoria.

Para el uso de las interrupciones por Software se utiliza la instrucción Int seguida del tipo de la interrupción. Cada tipo de interrupción realiza varios servicios, tareas o funciones, así, antes de "llamar" a la interrupción, se coloca en el registro AH el servicio que se solicita lleve a cabo la interrupción.

Por ejemplo:

```
Mov   ah,1
Int   21h
```

Realiza llamado a la interrupción 21h solicitándole que lea un carácter del teclado (servicio 1)

5.3. EJEMPLO DE ELABORACIÓN DE PROGRAMAS EN MACROENSAMBLADOR INCLUYENDO USO DE INTERRUPCIONES

; Programa numero 1: <u>suma de 2 números</u>

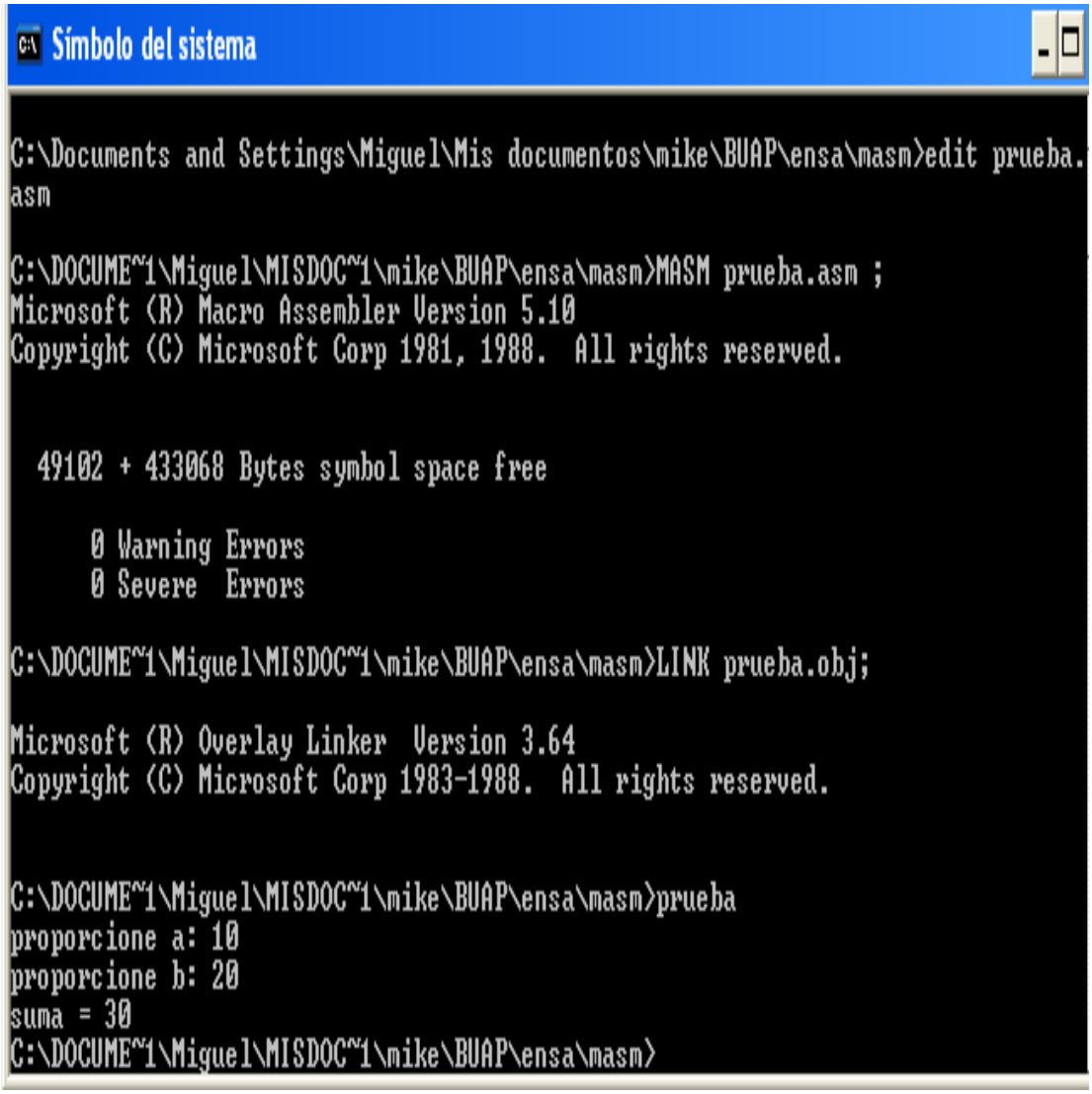
```
Pila    segment para stack ´stack"
    DW 30 Dup (0)
pila    ends
```

```
datos       segment para 'data'        ;se definen las variables y constantes a usar
    mens1   db 'proporcione a: $'
    mens2   db 'proporcione b: $'
    mens3   db 'suma = $'
          a db  0
datos       ends

código      segment para 'code'

princi  proc        far
        assumess:pila, ds:datos, cs:co digo
        push        ds                  ; protocolo
        xor         ax, ax              ;guarda dirección de retorno al SO
        push         ax
        mov         ax, datos           ;restaura ds
        mov         ds,a x

        lea         dx, mens1           ;carga en dx la dirección del mensaje 1
        call        escribe_cadena      ;se escribe el mensaje 1 en pantalla
        call        empaqueta           ; Lectura del dato 1 (a)
        mov         a, al               ;se guarda el dato uno en la variable “a”
        call        ali_linea           ;se pasa a la siguiente línea
        lea         dx, mens2           ;carga en dx la dirección del mensaje 2
        call        escribe_cadena      ;se escribe el mensaje 2 en pantalla
        call        empaqueta           ;lectura del dato 1 (b) se almacena en „al"
        add         a, al               ;se suma el dato 1 (a) y el dato 2  „al"
        call        ali_linea           ;se pasa a la siguiente línea
        lea         dx, mens3           ;carga en dx la dirección del mensaje 3
        call        escribe_cadena      ;se escribe el mensaje 3 en pantalla
        mov         dl, a               ;se traspasa el resultado a „dl"
        call        desempaqueta        ;se escribe el resultado leído de „dl"
        ret
princi  endp

; RUTINAS (estas rutinas se reutilzan en los programas siguientes)
escribe_cadena      proc                ; escribe a pantalla cadena apuntada por dx
        push        ax
        push        bx
        push        cx
        push        dx
```

```
        mov     ah, 09
        int     21H
        pop     dx
        pop     cx
        pop     bx
        pop     ax
        ret
escribe_cadena  endp


empaqueta   proc                ; devuelve dato leído en al
        push    cx
        call    lee
        call    ascii_bin
        mov     cl,4
        shl     al,cl
        mov     ch,al
        call    lee
        call    ascii_bin
        add     al,ch
        pop     cx
        ret
empaqueta   endp


lee     proc                    ; lectura de un dato(carácter) en al
        push    bx
        push    cx
        push    dx
        mov     ah,01
        int     21H
        pop     dx
        pop     cx
        pop     bx
        ret
lee     endp
```

```
ascii_bin       proc        ; conversión código Ascii a binario
        cmp     al,30H
        jl      error
        cmp     al,39H
        jle     resta30
        cmp     al,41H
        jl      error
        cmp     al,46H
        jle     resta37
error:  mov     al,0
        jmp     fin
resta30:sub     al,30H
        jmp     fin
resta37:sub     al,37H
fin:    ret
ascii_bin       endp


lee_cadena      proc        ; Lee una cadena dejándola donde apunta Dx
se
                            ; usara para los ejercicios que manejen cadenas
        Push    bx
        Push    cx
        Push    dx
        Mov     ah,0a
        Int     21

        Pop     dx
        Pop     cx
        Pop     bx
        Ret
lee_cadena      endp

ali_linea       proc        ; alimenta línea (Enter)
        Push    dx
        Mov     Dl,0a
        Call    escribe_caracter
        Mov     Dl,0d
        Call    escribe_caracter
        Pop     dx
        Ret
ali_linea       endp
```

```
escribe_caracter    proc                ; escribe a pantalla el  carácter almacenado
        en dl
        push        ax
        push        bx
        push        cx
         push       dx
        mov         ah,2
         int        21h
        pop         dx
         pop        cx
        pop         bx
        pop         ax
         ret
escribe_caracter    endp

desempaqueta        proc        ; escribe dato (número de dos dígitos)  almacenado
        en dl push  cx
        push        dx
        mov         dh,dl
        mov         cl,4
        shr         dl,cl
        call        bin_ascii
        call        escribe_caracter
        mov         dl,dh
        and         dl,0fH
        call        bin_ascii
        call        escribe_caracter
        pop         dx
        pop         cx
         ret
desempaqueta        endp

bin_ascii           proc                    ; convierte de binario a ascii
        cmp         dl,9
        jg          su37
        add         dl,30H
        jmp         fin1
su37: add           dl,37H
fin1:   ret
bin_ascii           endp

codigo      ends
end         princi
```



BIBLIOGRAFÍA

1. Abel, Peter. Lenguaje Ensamblador y Programación para PC IBM y Compatibles. Tercera edición. Prentice Hall. México, 1996.

2. "Assembly", http://www.linux.org/assembly

3. "Manual de Intel".

4. "TASM 2.x/MASM 6.x Assembly Language – Norton Guide", http://www.ousob.com/ng/masm/

5. Bacca Cortes, Eval Bladimir. Curso de Ensamblador para microprocesadores.