U.N.J.U. – F.I.

INGENIERÍA INFORMÁTICA

LABORATORIO DE COMPUTADORAS

ARQUITECTURA X86

TEMA: PROGRAMACIÓN ASSEMBLER, PROCEDIMIENTOS Y MACROS

Interrupciones

Una interrupción es una instrucción que detiene la ejecución de un programa para permitir el uso de la CPU a un proceso prioritario. Una vez concluido este último proceso se devuelve el control a la aplicación anterior.

El manejo directo de interrupciones es una de las partes más fuertes del lenguaje ensamblador, ya que con ellas es posible controlar eficientemente todos los dispositivos internos y externos de una computadora.

Tipos de Interrupciones

Internas de Hardware: son generadas por ciertos eventos que surgen durante la ejecución de un programa. Este tipo de Interrupciones son manejadas, en su totalidad, por el hardware y no es posible modificarlas.

Externas de Hardware: las generan los dispositivos periféricos, como pueden ser: teclado, impresoras, tarjetas de comunicaciones, etc. También son generadas por los coprocesadores. No es posible desactivar a las Interrupciones externas.

Estas Interrupciones no son enviadas directamente a la CPU, sino que se mandan a un circuito integrado cuya función es exclusivamente manejar este tipo de Interrupciones. El circuito, llamado PIC 8259A, es controlado por la CPU utilizando para tal control una serie de vías de comunicación llamadas puertos.

- **IRQ**: llamadas Petición de Interrupción, son interrupciones enmascarables.
- **NMI**: son llamadas Interrupciones No Enmascarables.

Tipos de Interrupciones

De Software: Las interrupciones por software se comportan de igual manera que las de hardware pero en lugar de ser ejecutadas como consecuencia de una señal física, lo hacen con una instrucción. Estas interrupciones pueden ser activadas directamente por el ensamblador invocando al número de interrupción deseada con la instrucción INT.

El uso de las Interrupciones ayuda en la creación de programas; al utilizarlas nuestros programas son más cortos, es más fácil entenderlos y usualmente tienen un mejor desempeño debido en gran parte a su menor tamaño.

Este tipo de interrupciones podemos separarlas en dos categorías: las Interrupciones del sistema operativo DOS y las Interrupciones del BIOS.

Interrupciones de Software

La diferencia entre ambas es que las interrupciones del sistema operativo son más fáciles de usar pero también son más lentas ya que estas interrupciones hacen uso del BIOS para lograr su cometido, en cambio las interrupciones del BIOS son mucho más rápidas pero tienen la desventaja que, como son parte del hardware, son muy específicas y pueden variar dependiendo incluso de la marca del fabricante del circuito.

La elección del tipo de interrupción a utilizar dependerá únicamente de las características que le quiera dar a su programa: velocidad (utilizando las del BIOS) o portabilidad (utilizando las del DOS).

Interrupciones de BIOS:

- Int 10h (Entrada/Salida de video)
- Int 16h (Entrada/Salida de teclado)
- Int 17h (Entrada/Salida de impresora)

Interrupciones de DOS:

- Int 21h (Múltiples llamadas a funciones del DOS)

Procedimientos

Un procedimiento es un conjunto de instrucciones a los que podemos dirigir el flujo de nuestro programa, y una vez terminada la ejecución de dichas instrucciones se devuelve el control a la siguiente línea a procesar del código que mando llamar al procedimiento. Los procedimientos nos ayudan a crear programas legibles y fáciles de modificar.

Al momento de invocar a un procedimiento se guarda en la pila la dirección de la siguiente instrucción del programa para que, una vez transferido el flujo del programa y terminado el procedimiento, se pueda regresar a la línea siguiente del programa original (el que llamó al procedimiento).

Sintaxis de un procedimiento

Existen dos tipos de procedimientos, los intrasegmentos, que se encuentran en el mismo segmento de instrucciones y los intersegmentos que pueden ser almacenados en diferentes segmentos de memoria.

Procedimientos

Cuando se utilizan los procedimientos intrasegmentos se almacena en la pila el valor de IP y cuando se utilizan los intersegmentos se almacena el valor CS:IP

Para desviar el flujo, de nuestro programa, a un procedimiento (llamarlo) se utiliza la instrucción CALL:

```
CALL NombreDelProcedimiento
```

Las partes que componen a un procedimiento son:

- Declaración del procedimiento.
- Código del procedimiento.
- Directiva de regreso.
- Terminación del procedimiento.

Procedimientos

Ejemplo: si queremos una rutina que nos sume dos bytes, almacenados en AH y AL cada uno y guardar la suma en el registro BX:

```
Suma Proc Near  ;Declaración del procedimiento
  Mov Bx, 0     ;Contenido del procedimiento
  Mov Bl, Ah
  Mov Ah, 00
  Add Bx, Ax
  Ret           ;Directiva de regreso
Suma Endp       ;Declaración de final del proc.
```

En la declaración la primera palabra, Suma, corresponde al nombre de nuestro procedimiento, Proc lo declara como tal y la palabra Near le indica al MASM que el procedimiento es intrasegmento. La directiva Ret carga la dirección IP almacenada en la pila para regresar al programa original. Por último, la directiva Suma Endp indica el final del procedimiento.

Procedimientos

Para declarar un procedimiento intersegmento sustituimos la palabra Near por la palabra FAR.

El llamado de este procedimiento se realiza de la siguiente forma:

Call Suma

Macros

Cuando un conjunto de instrucciones en ensamblador aparece frecuentemente repetidas a lo largo de un listado, es conveniente agruparlas bajo un nombre simbólico que las sustituirá en aquellos puntos donde aparezcan. Esta es la misión de las macros; por el hecho de soportarlas el ensamblador eleva su categoría a la de macroensamblador, al ser las macros una herramienta muy cotizada por los programadores.

No conviene confundir las macros con subrutinas: en éstas últimas, el conjunto de instrucciones aparece una sola vez en todo el programa y luego se invoca con CALL. Sin embargo, cada vez que se referencia a una macro, el código que ésta representa se *expande* en el programa definitivo, duplicándose tantas veces como se use la macro. Por ello, aquellas tareas que puedan ser realizadas con subrutinas siempre será más conveniente realizarlas con las mismas, con objeto de economizar memoria.

Macros

Es cierto que las macros son algo más rápidas que las subrutinas (se ahorra un CALL y un RET) pero la diferencia es tan mínima que en la práctica es despreciable en el 99,99% de los casos.

DEFINICIÓN Y BORRADO DE LAS MACROS

La macro se define por medio de la directiva MACRO. Es necesario definir la macro **antes** de utilizarla. Una macro puede llamar a otra. Con frecuencia, las macros se colocan juntas en un fichero independiente y luego se mezclan en el programa principal con la directiva INCLUDE:

```
IF1
    INCLUDE fichero.ext
ENDIF
```

La sentencia IF1 asegura que el ensamblador lea el fichero fuente de las macros sólo en la primera pasada, para acelerar el ensamblaje y evitar que aparezcan en el listado (generado en la segunda fase).

Invocación a una macro:

NombreMacro [parámetros actuales]

Macros

Conviene hacer hincapié en que la definición de la macro no consume memoria, por lo que en la práctica es indiferente declarar cientos que ninguna macro.

Sintaxis de una MACRO

nombre_simbólico **MACRO [parámetros]**

…

… ; instrucciones de la macro

ENDM

Para borrar una macro, la sintaxis es la siguiente:

PURGE nombre_simbólico[,nombre_simbólico,...]

Macros

EJEMPLO DE UNA MACRO SENCILLA

Desde el 286 existe una instrucción muy cómoda que introduce en la pila 8 registros, y otra que los saca (PUSHA y POPA, respectiuvamente). Se puede crear unas macros que simulen estas instrucciones en los 8086´s:

```
SUPERPUSH   MACRO
    PUSH   AX
    PUSH   CX
    PUSH   DX
    PUSH   BX
    PUSH   SP
    PUSH   BP
    PUSH   SI
    PUSH   DI
ENDM
```

Macros

La creación de SUPERPOP es análoga, sacando los registros en orden inverso. El orden elegido no es por capricho y se corresponde con el de la instrucción PUSHA original, para compatibilizar. A partir de la definición de esta macro, tenemos a nuestra disposición una nueva instrucción máquina (SUPERPUSH) que puede ser usada con libertad dentro de los programas.

PARÁMETROS FORMALES Y PARÁMETROS ACTUALES

Cuando se llama a una macro se le pueden pasar, opcionalmente, un cierto número de parámetros de cierto tipo. Estos parámetros se denominan *parámetros actuales*. En la definición de la macro, dichos parámetros aparecen asociados a ciertos nombres arbitrarios, cuya única misión es permitir distinguir unos parámetros de otros e indicar en qué orden son entregados: son los *parámetros formales*.

Macros

Cuando el ensamblador expanda la macro al ensamblar, los parámetros formales serán sustituidos por sus correspondientes parámetros actuales. Considere el siguiente ejemplo:

```
SUMAR      MACRO  a, b, total
PUSH   AX
MOV    AX, a
ADD    AX, b
MOV    total, AX
POP    AX
ENDM
....
SUMAR positivos, negativos, total
```

En el ejemplo, «a», «b» y «total» son los parámetros formales y «positivos», «negativos» y «total» son los parámetros actuales.

Macros

Tanto «a» como «b» pueden ser variables, etiquetas, etc. en otro punto del programa; sin embargo, dentro de la macro, se comportan de manera independiente. El parámetro formal «total» ha coincidido en el ejemplo y por casualidad con su correspondiente actual. El código que genera el ensamblador al expandir la macro será el siguiente:

```
PUSH   AX
MOV    AX, positivos
ADD    AX, negativos
MOV    total, AX
POP    AX
```

Las instrucciones PUSH y POP sirven para no alterar el valor de AX y conseguir que la macro se comporte como una *caja negra*; no es necesario que esto sea así pero es una buena costumbre de programación para evitar que los programas hagan cosas raras.

Macros

En general, las macros de este tipo no deberían alterar los registros y, si los cambian, hay que tener muy claro cuáles.

Si se indican más parámetros de los que una macro necesita, se ignorarán los restantes. En cambio, si faltan, el MASM asumirá que son nulos (0) y dará un mensaje de advertencia, el TASM es algo más rígido y podría dar un error. En general, se trata de situaciones atípicas que deben ser evitadas.

ETIQUETAS DENTRO DE MACROS. VARIABLES LOCALES

Son necesarias normalmente para los saltos condicionales que contengan las macros más complejas. Si se pone una etiqueta a donde saltar, la macro sólo podría ser empleada una vez en todo el programa para evitar que dicha etiqueta aparezca duplicada.

Macros

La solución está en emplear la directiva LOCAL que ha de ir colocada justo después de la directiva MACRO:

```
MINIMO     MACRO  dato1, dato2, resultado
LOCAL  ya_esta
MOV    AX, dato1
CMP    AX, dato2      ; ¿es dato1 el menor?
JB     ya_esta        ; sí
MOV    AX,dato2       ; no, es dato2
ya_esta:    MOV    resultado, AX
ENDM
```

En el ejemplo, al invocar la macro dos veces el ensamblador no generará la etiqueta «ya_esta» sino las etiquetas ??0000, ??0001, ... y así sucesivamente. La directiva LOCAL no sólo es útil para los saltos condicionales en las macros, también permite declarar variables internas a los mismos. Se puede indicar un número casi indefinido de etiquetas con la directiva LOCAL, separándolas por comas.