INSTRUCCIONES DE CONTROL DE FLUJO

Sistemas Digitales con Microprocesadores.

M.C. Juan Carlos Olguín R.

Los programas que se han desarrollado hasta ahora se ejecutan en forma secuencial, esto es, el programa inicia su ejecución con la primera instrucción y continúa de arriba hacia abajo ejecutando cada una de las instrucciones hasta llegar a la última.

En la mayoría de los casos, sin embargo, se requiere que el programa ejecute una serie de instrucciones dependiendo de sí una condición se cumple o de que ejecute una serie de instrucciones en forma repetitiva. Este tipo de programas se puede implementar mediante las instrucciones de **control de flujo**. Las instrucciones de control de flujo se clasifican en

- ✓instrucciones de salto incondicional
- ✓instrucciones de salto condicional
- ✓instrucciones repetitivas.

Instrucción De Salto Incondicional

La instrucción de salto incondicional hace que el control de flujo del programa salte a la instrucción cuya dirección está especificada por su operando.

jmp *dirección* ;La instrucción **jmp** hace que la ejecución del programa ;continué en la dirección especificada por el operando.

Sintaxis:

jmp *etiqueta*
jmp *regW\|memW*
jmp *memDW*

La dirección puede ser una etiqueta o estar en un registro o localidad de memoria.
En el caso de usar un registro o localidad de memoria de una palabra, el salto es a una instrucción que se encuentra en el mismo segmento y el valor representa el desplazamiento de la instrucción con respecto al segmento.
En el caso de que la dirección esté en una localidad de memoria de tipo palabra doble, el salto es a otro segmento y el valor es el segmento:desplazamiento de la instrucción.
La instrucción **jmp** no afecta a las banderas.

Instrucciones De Salto Condicional

Las instrucciones de salto condicional hacen que el control de flujo del programa salte a la instrucción cuya dirección está especificada por su operando si se cumple una condición dada.

jcond *dirección* ***;***Cambian la dirección de la siguiente instrucción a ;ser ejecutada dependiendo del valor de ciertas banderas .

Sintaxis:

j*cond* *etiqueta*

Todas las instrucciones de salto condicional operan en forma similar. ***j****cond* es el mnemónico de la instrucción y *etiqueta* es la etiqueta de la dirección donde está *la instrucción* a la que va a saltar el programa.

Los mnemónicos de las instrucciones de salto condicional pueden clasificarse en tres grupos:

Saltos si la condición es una comparación no signada

Instrucción	Salta si...	Banderas
ja | jnbe	mayor | no menor o igual	**C** = 0 & **Z** = 0
jae | jnb	mayor o igual | no menor	**C** = 0
jb | jnae	menor | no mayor o igual	**C** = 1
jbe | jna	menor o igual | no mayor	**C** = 1 | **Z** = 1
je	igual	**Z** = 1
jne	diferente	**Z** = 0

Aquí están los saltos cuya condición de salto es el resultado de una comparación de dos números no signados;

Saltos si la condición es una comparación signada

Instrucción	Salta sí ...	Banderas
jg | jnle	mayor | no menor o igual	**S** = **O** & **Z** = 0
jge | jnl	mayor o igual | no menor	**S** = **O**
jl | jnge	menor | no mayor o igual	**S** ≠ **O**
jle | jng	menor o igual | no mayor	**S** ≠ **O** | **Z** = 1
je	igual	**Z** = 1
jne	diferente	**Z** = 0

Aquí se muestran los saltos cuya condición de salto es el resultado de una comparación de dos números signados;

Saltos si la condición es un estado de una bandera o el registro CX

Instrucción	Salta sí ...	Banderas
jc	acarreo	**C** = 1
jo	sobreflujo	**O** = 1
jp | jpe	paridad | paridad par	**P** = 1
jpo | jnp	paridad non | no paridad	**P** = 0
js	signo	**S** = 1
jz	cero	**Z** = 1
jnc	no acarreo	**C** = 0
jno	no sobreflujo	**O** = 0
jns	no signo	**S** = 0
jnz	no cero	**Z** = 0
jcxz	si **CX** es cero	-

Una restricción que tienen todos los saltos condicionales es de que la dirección a la que saltan debe de estar como máximo 128 bytes hacia atrás o 127 bytes hacia adelante del primer byte de la siguiente instrucción a la instrucción de salto.

Por ejemplo: suponga que en el siguiente código, la instrucción a la que se desea saltar se encuentra más allá del rango de los -128 a 127 bytes:

```
        cmp     dx,1            ; if(DX == 1)
        je      error           ; goto error
        ...
error:                          ; Esta dirección está más
                                ; allá de los 127 bytes
```

El ensamblador nos desplegará el siguiente mensaje:

```
**Error** Relative jump out of range by n bytes
(**Error** Salto relativo fuera de rango por n bytes)
```

Ejemplo 1:

```
;*************************************************************
; MAYOR3.ASM
;
; Este programa encuentra el mayor de tres datos signados
; almacenados en variables de una palabra. El pseudocódigo
; de este programa es:
;
;    AX = dato1
;    if(AX > dato2) goto sigcmp
;    AX = dato2
;
;sigcmp:
;    if(AX > dato3) goto fincmp
;    AX = dato3
;
;    mayor = AX
```

```
;*************************************************************

;****** CÓDIGO DE INICIO ***************************************

        ideal
        dosseg
        model   small
        stack   256

;****** VARIABLES DEL PROGRAMA *********************************

        dataseg
codsal  db      0
dato1   dw      ?
dato2   dw      ?
dato3   dw      ?
mayor   dw      ?
```

```
;****** CÓDIGO DEL PROGRAMA ******************************

        codeseg
inicio:
        mov     ax, @data       ; Inicializa el
        mov     ds, ax          ; segmento de datos

        mov     ax, [dato1]     ; AX = dato1
        cmp     ax, [dato2]     ; if(AX > dato2)
        jg      sigcmp          ;    goto sigcmp
        mov     ax, [dato2]     ; AX = dato2

sigcmp: cmp     ax, [dato3]     ; if(AX > dato3)
        jg      fincmp          ;    goto fincmp
        mov     ax, [dato3]     ; AX = dato3

fincmp: mov     [mayor], ax     ; mayor = AX

salir:
        mov     ah, 04Ch
        mov     al, [codsal]
        int     21h

;****** CÓDIGO DE TERMINACIÓN ****************************

        end     inicio
```

Ejemplo 2

```
;*************************************************************
; SERIE1.ASM
;
; Este programa suma los números enteros de 1 hasta nfinal.
; El pseudocódigo de este programa es:
;
;    AX = 0
;    CX = nfinal
;
;    while(CX > 0)
;    {
;       AX += CX
;       CX—
;    }
;
;    suma = AX
;*************************************************************

;****** CÓDIGO DE INICIO ***********************************

        ideal
        dosseg
        model   small
        stack   256
```

```
;****** VARIABLES DEL PROGRAMA ******************************

        dataseg
codsal  db      0
nfinal  dw      ?
suma    dw      ?

;****** CÓDIGO DEL PROGRAMA *********************************

        codeseg
inicio:
        mov     ax, @data       ; inicializa el
        mov     ds, ax          ; segmento de datos

        xor     ax, ax          ; AX = 0
        mov     cx, [nfinal]    ; CX = nfinal

while:  jcxz    endwhi          ; while(CX > 0)
                                ; {
        add     ax, cx          ;    AX += CX
        dec     cx              ;    CX--
        jmp     while           ; }

endwhi:
        mov     [suma], ax      ; suma = AX
```

Instrucciones Repetitivas

El ensamblador del 8086 posee tres instrucciones especiales que permiten la construcción de ciclos.

loop *dirección* ;Decrementa **CX** y luego salta si **CX** no es **0**.

Sintaxis:

loop *etiqueta*

Esta instrucción decrementa el contenido del registro **CX** en uno, si el valor que queda en **CX** es diferente de cero, entonces la instrucción salta a la dirección especificada por *etiqueta*, *la cual no debe* encontrarse más allá de los 126 bytes hacia atrás o de los 127 bytes hacia adelante.

Esta instrucción se utiliza para crear ciclos que se repiten el número de veces especificado por el registro **CX.** Como la instrucción loop decrementa **CX** antes de probar si vale cero.
Si **CX** vale cero al principio, al decrementar **CX** tomará el valor de **65535** y por lo tanto el ciclo ejecutará **65536** veces.
Para prevenir esto preceda el ciclo con la instrucción **jcxz** como se muestra en el siguiente código:

```
        jcxz    enddo           ; if(CX == 0) goto enddo

do:                             ; do
                                ; {
        ...                      ;     instrucciones dentro
del
                                ;     ciclo
        loop    do              ; }
                                ; while(CX > 0)

enddo:
```

la instrucción **loop** no modifica las banderas.

loope | loopz *dirección* ;Decrementa **CX** y luego salta si **CX** no es **0** **Y** la ;bandera de cero **Z** vale **1.**

Sintaxis:

loope | loopz *etiqueta*

Los mnemónicos **loope y loopz** representan la misma instrucción.

Esta instrucción decrementa el contenido del registro **CX** en uno, si el valor que queda en **CX** es diferente de cero y la bandera de cero **Z** vale **1**, presumiblemente puesta por una comparación previa, entonces la instrucción salta a la dirección especificada por *etiqueta, la cual no debe encontrarse más allá de los 126 bytes hacia atrás o* 127 bytes hacia adelante.

loopne | loopnz *dirección* ;Decrementa **CX** y luego salta si **CX** no es **0** **Y** la ;bandera de cero **Z** vale **0.**

Sintaxis:

loopne | loopnz *etiqueta*

Los mnemónicos **loope y loopz** representan la misma instrucción. Esta instrucción decrementa el contenido del registro **CX** en uno, si el valor que queda en **CX** es diferente de cero **y** la bandera de cero **Z** vale **0,** presumiblemente puesta por una comparación previa, entonces la instrucción salta a la dirección especificada por *etiqueta, la cual no debe encontrarse más allá de los 126 bytes hacia atrás o* 127 bytes hacia adelante.

Ejemplo sobre instrucciones repetitivas

```
;*************************************************************
; SERIE2.ASM
;
; Este programa suma los números enteros de 1 hasta nfinal.
; Esta versión utiliza la instrucción repetitiva loop. El
; pseudocódigo de este programa es:
;
;    AX = 0
;    CX = nfinal
;
;    if(CX == 0) goto endo
;
;    do
;    {
;       AX += CX
;    }
;    while(--CX > 0)

; enddo:
;    suma = AX
;*************************************************************
```

```
;****** CÓDIGO DE INICIO **********************************

        ideal
        dosseg
        model   small
        stack   256

;****** VARIABLES DEL PROGRAMA ****************************

        dataseg
codsal  db      0
nfinal  dw      ?
suma    dw      ?

;****** CÓDIGO DEL PROGRAMA *******************************

        codeseg
inicio:
        mov     ax, @data       ; Inicializa el
        mov     ds, ax          ; segmento de datos
```

```
        xor     ax, ax              ; AX = 0
        mov     cx, [nfinal]        ; CX = nfinal

        jcxz    endo                ; if(CX == 0) goto endo

do:                                 ; do
                                    ; {
        add     ax, cx              ;     AX += CX
        loop    do                  ; }
                                      while(--CX > 0)
enddo:
        mov     [suma], ax          ; suma = AX

salir:
        mov     ah, 04Ch
        mov     al, [codsal]
        int     21h

;****** CÓDIGO DE TERMINACIÓN ********************************

        end     inicio
```

Lo que no queremos que les pase:

- En un foro en internet me encontré con la siguiente duda:
- Necesito resolver esta tarea para la universidad, pero no se como, alguien que me ayude porfavor?????

Restar 1 byte de contenido de registro E y un byte del dato inmediato 3f. Resultado, almacenar en memoria con direccion 1400H, si el resultado positivo y con direccion 14FF si resultado negativo.

Algunas Respuestas del foro:

- En primera no se hacen tareas, en segunda es **(triste) que no te des ala tarea de buscar un poco de información ese programa es de lo más sencillo, para que **(caray) te metes ala universidad si no quieres esforzarte en hacer ese programita...

  Te recomiendo buscar un poquito más y veras que es de lo más sencillo con solo esforzarte un poco.

Una respuesta del foro, más amable fue:

- No conozco el conjunto de instrucciones de la arquitectura. Pero si tú la conoces:

  E al acumulador
  Acumulador -1
  acumulador a e
  3f al acumulador
  acumulador -1
  acumulador a 3f
  si negativo, saltar a X
  acumulador a 1400
  terminar
  X: acumulador a 14FF
  terminar

  //Sólo tradúcelo.
  Saludos.

Nadie hará por ti el TRABAJO que te corresponde!!!

Nos Vemos la próxima clase para seguir trabajando!!!!