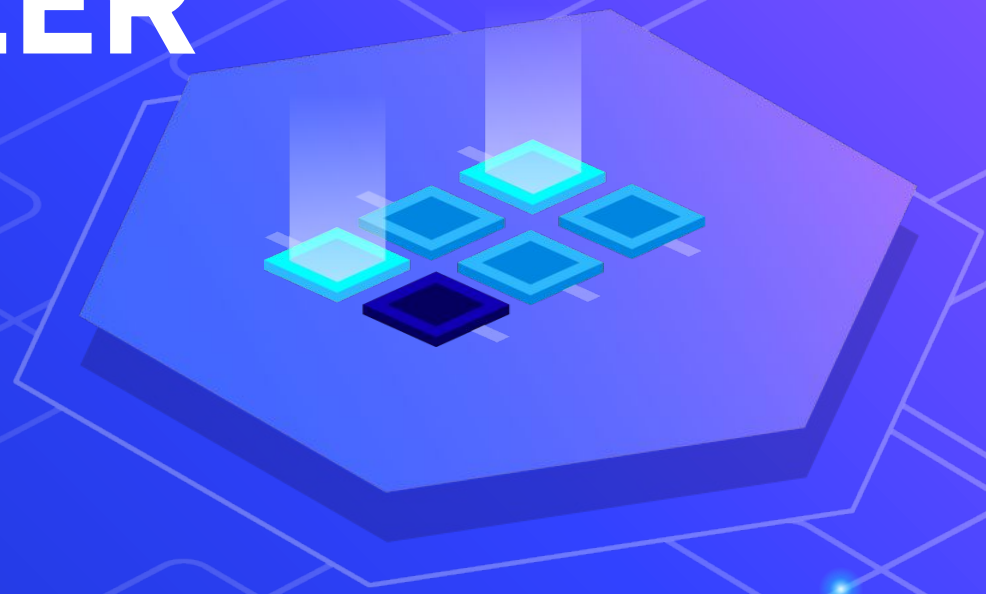
MÓDULO ASSEMBLER

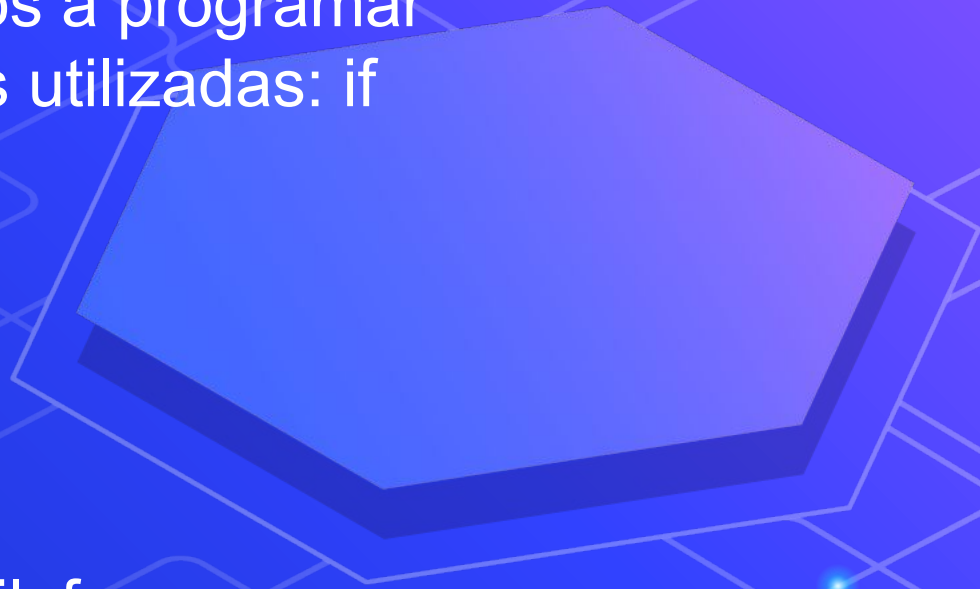
Estructuras de control



Autores:
Alejandro Héctor Gonzalez
Silvana Lis Gallo
Junio 2021

RESUMEN

En esta clase se trabaja con los saltos incondicionales del Assembler del 8088. Con estos saltos vamos a programar en bajo nivel las estructuras de control más utilizadas: if ...then..else, while, repeat until y for



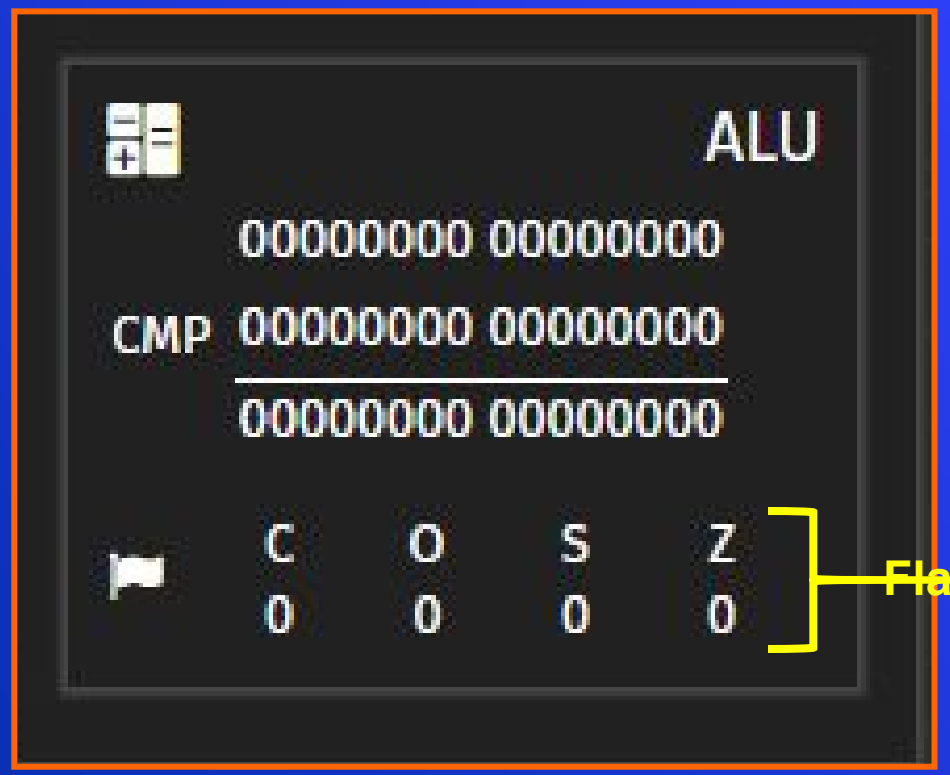
Palabras clave

estructuras de control, if, while, repeat until ,for

Repaso: Modos de direccionamiento

- Inmediato
- Directo de memoria o Absoluto
- Directo de Registro
- Indirecto de memoria (en desuso)
- Indirecto con registro
- Indirecto con Desplazamiento
  - basado, indexado o relativo al PC
  - Pila (o relativo al SP)

Flags de la CPU



- Son bits que indican el estado de la ALU luego de la ejecución de alguna operación
- Utilizaremos principalmente los bits C, O, S y Z.
  (que son los vistos en la ALU de Vonsim)

Flags de la CPU - C (Carry)

- **Carry** : Esta bandera esta en 1 después de una operación, el resultado excede el tamaño del operando.

Ejemplo:

```
  11111111 +
  00000001
 ----------
 100000000
```

El resultado no cabe en 8 bits por lo tanto CF=1

Flags de la CPU – O (Overflow)

- **Overflow** : Si el resultado de una oración de números con signo esta fuera del rango.

Ejemplo:

```
01111111 +      = 127 +
00000001        =   1
--------
10000000        = 128 Rango de números de 8 bits con signo es -128..127,
```

como 128 esta fuera del rango OF=1.

Flags de la CPU - S (Signo)

Signo : Esta bandera es 1 si después de una operación el bit mas significativo esta en 1.

Ejemplo:

```
01111111 +
00000001
--------
10000000
```

El bit mas significativo es 1 por lo tanto SF=1

Flags de la CPU - Z (Cero)

- **Cero** : Esta bandera esta en 1 si el resultado de una operación es cero.

Ejemplo:

```
 11111111 +
 00000001
----------
100000000
```

El resultado final es cero por lo tanto ZF=1.

Instrucciones de salto

```
JMP dirección     ; Salta siempre

JZ  dirección     ; Salta si el flag Z=1
JNZ dirección     ; Salta si el flag Z=0
JS  dirección     ; Salta si el flag S=1
JNS dirección     ; Salta si el flag S=0
JC  dirección     ; Salta si el flag C=1
JNC dirección     ; Salta si el flag C=0
JO  dirección     ; Salta si el flag O=1
JNO dirección     ; Salta si el flag O=0
```

Ejemplos de Saltos

JZ salta si el resultado de la última operación que alteró los flags es igual a cero (Z=1) , también funciona para números con signo

```
CMP AL ‘S’          ; Compara AL con la letra ‘S’
JZ  ESTA_BIEN ; Salta si es igual a 0 a la etiqueta ESTA_BIEN
```

Ejemplos de Saltos

JNZ salta si el resultado de la última operación que alteró los flags NO es igual a cero (Z=O) , también funciona para números con signo

```
            CMP AL 0 ; Compara AL con el número 0 en decimal
            JNZ  ES_UNO ;
ES_UNO:
            CMP AL, 1
            JNZ ES_DOS
ES_DOS: …
```

¿Cómo armar estructuras de control en Assembler?

¿Cómo hacer un if?

En Pascal sería:

```
IF AL = 4 THEN
  BEGIN
    BL = 1;
    CL = CL + 1;
  END;
```

En assembler:

```
        CMP AL, 4 ; (a)
        JZ Then ; (b)
        JMP Fin_IF ; (c)
Then:   MOV BL, 1 ; (d)
        INC CL ; (e)
Fin_IF: HLT
```

CMP alterará los flags y en particular, nos interesa ver al flag Z, ya que si dicho flag está en 1, implica que al resta AL con 4, el resultado dio 0, por lo que AL tiene que valer 4

Si la comparación en (a) establece el **flag Z en 1**,

- el salto en (b) se produce, haciendo que la ejecución continúe en la etiqueta **Then**.

- Ahí, se ejecutan las instrucciones (d) y (e) que hacen lo que se encuentra en el THEN del IF y continúa la ejecución en la instrucción apuntada por la etiqueta **Fin_IF** .

Si el **flag Z quedó en 0** en (a),

- el salto en (b) no se produce, por lo que la ejecución continúa en la próxima instrucción,
- el JMP en (c), que saltea las instrucciones y continúa la ejecución en la instrucción apuntada por la etiqueta **Fin_IF** , que señala el final del IF

IF …. Then … else

```
IF AL = 4 THEN
    BEGIN
        BL = 1;
        CL = CL + 1;
    END
ELSE
    BEGIN
        BL = 2;
        CL = CL – 1;
    END;
```

```
        ORG 2000h
        CMP AL, 4 ;
        JZ Then ;
        JMP Else ;
Then:   MOV BL, 1 ;
        INC CL ;
        JMP Fin_IF ;
Else:   MOV BL, 2 ;
        DEC CL ;
Fin_IF: HLT ;
        END
```

Este salto incondicional hace que no se ejecute el Then, cuando AL no es igual a 4

Lazos

```
ORG 2000h
        MOV AX, 10
Lazo: ... ;
        ... ; <Instrucciones a repetir>
        ... ;
        DEC AX
        JNZ Lazo
        JMP Fin
Fin:    HLT
        END
```

El ejemplo plantea un esquema básico de iteración usado comúnmente como algo equivalente a un **“ For i := n downto 1”** de Pascal. En AX tengo la cantidad de repeticiones

El programa inicializa AX en 10, hace lo que tenga que hacer dentro del bucle, decrementa AX en 1 y salta a la instrucción apuntada por la etiqueta *Lazo* (el DEC) siempre y cuando el flag Z quede en 0, o sea, que el resultado de la operación anterior no haya dado como resultado 0.

Como AX se decrementa en cada iteración, llegará un momento en que AX será 0, por lo que el salto condicional no se producirá y continuará la ejecución del programa en la siguiente instrucción luego de dicho salto.

¿Qué estructura de control implementa este código?

```
        MOV AL, 0
        MOV CL, 10
Volver: ADD AL, AL
        DEC CL
        CMP CL, 1
        JNZ Volver
Fin:    HLT
```

Podemos verlo como un **Repeat….until** de Pascal, donde sabemos la cantidad de veces, pero podría ser que no supiéramos cuantas veces ejecuta.

¿Podemos verlo como For este caso?
SI, también se lo puede interpretar como FOR

¿Cómo hacemos un **While <condición> do** , de Pascal?

Ejemplo de Repaso

```
        ORG 1000h
        tabla DB 1, 2, 3, 4, 5
        fin_tabla DB ?
        resultado DB 0

        ORG 2000h
        MOV BX, OFFSET tabla ; (a)
        MOV CL, OFFSET fin_tabla – OFFSET tabla ; (b)
Loop:   MOV AL, [BX] ; (c)
        INC BX ; (d)
        XOR resultado, AL ; (e)
        DEC CL
        JNZ Loop
        HLT
        END
```

(a) Inicializa BX con “OFFSET tabla”. Esto indica que se debe **cargar** en BX **la dirección de tabla**, no el contenido de dicha variable.

(b) Se asigna a CL la diferencia entre la dirección de tabla y la dirección de fin_tabla. Lo que se logra es calcular cuantos bytes hay entre el comienzo y el final de la tabla. De esta manera, obtenemos la **cantidad de datos** que contiene la tabla.

(c) Vemos que en el MOV aparece BX entre corchetes. Esto significa que se debe asignar en AL el contenido de la celda de memoria cuya dirección es el valor contenido en BX. Así, como BX se había inicializado con la dirección del comienzo de la tabla, esto causa que se cargue en AL el **contenido de la primer entrada de la tabla.**

(d) Se incrementa BX, con lo que ahora **apunta al siguiente byte** de la tabla

(e) Se calcula una **operación XOR** con el contenido de la variable resultado y el byte que se acaba de obtener en AL, dejando el resultado de esa operación en la misma variable. Esta operación es bit a bit VV es F y FF es F

Ejercicio para consultar con los ayudantes

Tener en cuenta que la tabla es de Word (2 bytes)

1. Realizar paso a paso los momentos de los registros de la CPU.

2. Armar la memoria de Datos.

3. ¿Qué realiza el siguiente programa?

```
        ORG 1000h
        tabla  dw 5, 2, 10, 4, 5, 0, 4, 8, 1, 9
        min    dw 0FFFFh

        ORG  2000h
        MOV BX, OFFSET tabla
        MOV CL, 0
        MOV AX, min
Loop:   CMP [BX], AX
        JNC Menor
        MOV AX, [BX]
Menor:  ADD BX, 2
        INC CL
        CMP CL, 10
        JNZ Loop
        MOV min, AX
        HLT
        END
```

Recordar que el CMP hace **Destino – Fuente** y no almacena nada en destino. Solo altera los flags de la CPU

¿Que valor queda en min?

Bibliografía e información

- Organización y Arquitectura de Computadoras. W. Stallings, 5ta Ed.

  Repertorios de instrucciones
  - Capítulo 9: características y funciones
  - Capítulo 10: modos de direccionamiento y formatos
  - Apéndice 9A: Pilas

  Ciclo de instrucción:
  - Capítulo 3 apartado 3.2.
  - Capítulo 11 apartados 11.1. y 11.3.

  Organización de los registros
  - Capítulo 11 apartado 11.2.

  Formatos de instrucciones
  - Capítulo 10 apartado 10.3. y 10.4.
- Simulador MSX88

Link de interés: www.williamstallings.com