Arquitectura de computadoras
Lenguaje Ensamblador - Ejemplos resueltos

Los siguientes son ejemplos resueltos de programas escritos en lenguaje ensamblador utilizando el simulador http://schweigi.github.io/assembler-simulator/

Ejemplo 1:
Escriba un programa que realice la operación a=b*(c+d)/e. Almacene la variable a en la posición de memoria 0x50. Elija las posiciones donde almacenar las variables b, c, d y e.

Ejemplo 2:
Escriba un programa que realice la división de dos números. Debe indicar el cociente y el resto de la división. Almacene el dividendo en la posición 0x50, el divisor en la posición 0x51, el cociente en la posición 0x52 y el resto en la posición 0x53.

Ejemplo 3:
Escriba un programa que escriba 0x11 en las direcciones de memoria de la 0x30 a la 0x4F.

Ejemplo 4:
Escriba un programa que escriba números en las posiciones 0x40 a 0x45, luego busca el menor. El menor debe quedar en el registro B.

Ejemplo 5:
Repita el ejemplo 4 pero realizando la comparación dentro de una subrutina.

Resultados

Ejemplo 1:

```
        mov [0x51],10   ;La variable b se almacena en la posición 0x51
        mov [0x52],7    ;variable c
        mov [0x53],3    ;variable d
        mov [0x54],4    ;variable e
        mov A,[0x52]
aqui:
        add A,[0x53]
        mul [0x51]
```

```
        div [0x54]
        mov [0x50],A    ;La variable a se almacena en la posición 0x50
        hlt
```

Ejemplo 2:

```
        mov [0x50],50 ;dividendo
        mov [0x51],4  ;divisor
        mov A,[0x50]
        div [0x51]
        mov [0x52],A  ;movemos el cociente a la posición 0x52
                      ;resto = dividendo - cociente*divisor
        mov B,[0x50]  ;movemos el dividendo a B
        mov A,[0x52]  ;movemos el cociente a A
        mul [0x51]    ;cociente*divisor
        sub B,A       ;dividendo - cociente*divisor
        mov [0x53],B
        hlt
```

Ejemplo 3:

```
        mov A,0x30
aqui:
        mov [A],0x11
        add A,1
        cmp A,0x50
        jz fin
        jmp aqui
fin:
        hlt
```

Ejemplo 4:

Explicación del algoritmo: El algoritmo compara de a dos números. Siempre el menor lo almacena en el registro B. En el registro C va cargando uno a uno todos los números de la lista para cada comparación. Al final, el registro B contendrá el menor. El registro A se utiliza para recorrer la lista mediante un direccionamiento indirecto a registro.

```
        mov [0x40],0x12
        mov [0x41],0x34
        mov [0x42],0xA0
        mov [0x43],0x03
        mov [0x44],0x06
        mov [0x45],0x44
comienzo:
```

```
        mov A,0x40   ;Usaremos A para recorrer la lista de números
        mov B,[A]    ;Los números a comparar se almacenan en B y C.
loop:
        add A,1
        cmp A,0x46   ;Verificados si ya se recorrió toda la lista.
        jnz seguir
        jmp fin
seguir:
        mov C,[A]
        sub C,B      ;Comparamos los números mediante la resta.
                     ;Si C es menor, se produce acarreo.
        jae b_es_menor   ;Salta si no hay acarreo.
        mov B,[A]    ;Si hay un nuevo menor, lo almacenamos en B.
b_es_menor:
        jmp loop     ;Si B es menor, no hacemos nada y seguimos con el siguiente
                     ;número.
fin:
        hlt
```

Ejemplo 5:

Explicación del algoritmo. El algoritmo es igual al del ejemplo anterior. La comparación se realiza en la subrutina comparar. Dicha subrutina compara el contenido del registro B con el contenido del registro C, almacenando el menor numero en el registro B.

```
        mov [0x40],0x12
        mov [0x41],0x34
        mov [0x42],0x00
        mov [0x43],0x03
        mov [0x44],0x06
        mov [0x45],0x44
comienzo:
        mov A,0x40   ;Usaremos A para recorrer la lista de números
        mov B,[A]    ;Los números a comparar se almacenan en B y C.
loop:
        add A,1
        cmp A,0x46   ;Verificados si ya se recorrió toda la lista.
        jnz seguir
        jmp fin
seguir:
        mov C,[A]
        call comparar
        jmp loop
fin:
```

```
        hlt

comparar:
        sub C,B      ;Comparamos los números mediante la resta.
                     ;Si C es menor, se produce acarreo.
        jae b_es_menor  ;Salta si no hay acarreo.
        mov B,[A]    ;Si hay un nuevo menor, lo almacenamos en B.
b_es_menor:          ;Si B es menor, no hacemos nada y seguimos con el
        ret      ;siguiente número.
```