PRÁCTICA 2

Carlos Raso Alonso

Introducción a los ordenadores

Objetivos:

El objetivo principal de esta práctica es familiarizarnos con el simulador 8085 y con sus instrucciones, ya que varias diferencias significativas con el Ripes. También teníamos que entender el concepto de subrutina e implementarlo, además de entender cómo se relacionan las subrutinas con el “stack” y aprender a usar el “stack”.

Ejercicio 1:

Pregunta 1

- ¿En qué simplificaría el código uno de los modos de direccionamiento del Ripes?

Se podría simplificar mucho el código si pudiéramos hacer direccionamiento directo a dos registros para así no tener que operar con el acumulador. Por ejemplo para sumar dos registros no haría falta cargar uno en el acumulador.

- Calcula el número de ciclos en la ejecución del programa, así como la medida.

La medida de todas las instrucciones son 35 Bytes. Podemos calcularla sumando la medida de cada instrucción o bien mirando cuánto ocupa en la memoria la sección .org 100h

El número de ciclos al ejecutar todo el programa está entre 703 y 739 ciclos.

Pregunta 2

-Indica la medida media de tus instrucciones y calcula los ciclos por instrucción media de tus instrucciones

La medida media de mis instrucciones es de 5/3 de Byte. Lo he calculado dividiendo la medida total de las instrucciones, 35, entre el número de instrucciones, 21

Los ciclos medios por instrucción son 7’1, lo he calculado del mismo modo.

Pregunta 3

-¿cuál es la instrucción que más ciclos tarda en ejecutarse? Adjunta una captura de tu código

La instrucción que más ciclos tarda tarda 10 ciclos. Y es lxi, de carga de direcciones en registro

Código del programa: (el código está explicado en el ejercicio 2, cuya primera parte del código es igual que este código)

Las capturas con el código de los programas están en la siguiente página

Código utilizando 3 vectores con el que he hecho las preguntas:

```
.define
      num 5
.data 00h
      mat1: db 1, 2, 3, 4, 5
      mat2: db 6, 7, 8, 9, 0
      mat3: db 0, 0, 0, 0, 0
.org 100h
      lxi H, mat1 ; 3 bytes 10 ciclos, esta es de las instrucciones que más ciclos cuestan
      lxi B, mat3 ; 3 bytes 10 ciclos
      mvi D, num ; 2 bytes 7 ciclos
carga: ; 6 iteraciones de 64/67 ciclos
      mov A, M ; 1 bytes  7 ciclos
      stax B ; 1 bytes  7 ciclos
      inx H ; 1 bytes  6 ciclos
      inx B ; 1 bytes  6 ciclos
      dcr D ; 1 bytes  4 ciclos
      jp carga ; 3 bytes  7/10 ciclos
      lxi H, mat2 ; 3 bytes  10 ciclos
      lxi B, mat3 ; 3 bytes  10 ciclos
      mvi D, num ; 2 bytes  7 ciclos
loop: ; 6 iteraciones de 48/51 ciclos
      mov E, M ; 1 bytes  7 ciclos
      ldax B ; 1 bytes  7 ciclos
      add E ; 1 bytes  4 ciclos
      stax B ; 1 bytes  7 ciclos
      inx H ; 1 bytes  6 ciclos
      inx B ; 1 bytes  6 ciclos
      dcr D ; 1 bytes  4 ciclos
      jp loop ; 3 bytes  7/10 ciclos
hlt ; 1 bytes  4 ciclos
```

Código utilizando solo 2 vectores introduciendo la solución en uno de ellos:

```
.define
      num 5
.data 00h
      mat1: db 1, 2, 3, 4, 5
      mat2: db 6, 7, 8, 9, 0
.org 100h
      lxi H, mat1
      mvi C, mat2
      mvi B, 0
      mvi D, num
loop:
      mov E, M
      ldax B
      add E
      stax B
      inx H
      inx B
      dcr D
      jp loop
hlt
```

Pregunta 4:

-Pasa el código a ripes y compáralo

```
.data
mat1: .byte 1, 2, 3, 4, 5
mat2: .byte 6, 7, 8, 9, 0
mat3: .byte 0, 0, 0, 0, 0
num: .byte 5
.text
la a0, mat1
lw a3, 24(a0) # cargamos en a3 el número de iteracuones que debemos realizar
loop:
# tomamos los valores de los dos vectores correspondientes a la iteración en la que
# estamos y los almacenamos en  a1 y a2
lb a1, 0(a0)
lb a2, 8(a0)
# almacenamos la suma de ambos valores en a4 para luego almacenarla en el tercer
#vector
add a4, a1, a2
sb a4, 16(a0)
# restamos 1 al número de iteraciones que quedan y sumamos 1 a a0 para avanzar
# la posición en todos los vectores
addi a0, a0, 1
addi a3, a3, -1
bgtz a3, loop
```

El código en Ripes realiza solo 55 ciclos con el 5 stage processor, muchos menos de los que realiza en 8085. La media de ciclos por instrucción en Ripes es de 5,5 ciclos por instrucción y la medida del código es de 10 instrucciones * 4 bytes por instrucción = 40 bytes (la medida de las instrucciones es fija en Ripes, de 4 bytes)

Ejercicio 2:

Pregunta 4

- ¿Qué instrucción usamos en ambos casos para asignar la posición inicial al SP?

La instrucción call que llama a la subrutina es la que asigna a SP la posición inicial que apunta al inicio de la cola, en este caso la posición FFFF. Tras esta primera instrucción el SP irá variando según las instrucciones pop y push que se hagan hasta que se llegue a la instrucción ret.

Pregunta 5

- ¿Qué instrucción se usa para guardar y recuperar de la pila el valor del PC en las subrutinas?

El PC se guarda automáticamente en la pila al llamar a la subrutina con la instrucción call sin que sea necesario usar la instrucción push. Del mismo modo, al ejecutar la instrucción ret el valor del PC es recuperado sin tener que usar la instrucción pop.

Tarea 2

Código del programa:

```
.define
      num 5
      clave 55h
.data 00h
      mat1: db 1, 2, 3, 4, 5
      mat2: db 6, 7, 8, 9, 0
      mat3: db 0, 0, 0, 0, 0
.org 100h
      lxi H, mat1 ; cargamos dirección del primer elemento de mat1 en HL
      lxi B, mat3 ; hacemos lo propio con mat3 en BC
      mvi D, num ; cargamos el contenido de num en el registro D
carga:
      mov A, M ; cargamos en A el contenido de la dirección de memoria contenida en HL
      stax B ; almacena el contenido de A en la dirección de memoria contenida en
      inx H ; HL <= HL + 1
      inx B ; BC <= BC + 1
      dcr D ; DE <= DE - 1
      jp carga ; si A < 0  el programa continúa, si no va a la etiqueta carga
      lxi H, mat2 ; cargamos dirección del primer elemento de mat2 en HL
      lxi B, mat3 ; hacemos lo propio con mat3
      mvi D, num ; cargamos el contenido de num en el registro D
loop:
      mov E, M ; cargamos en E el contenido de la dirección de memoria contenida en HL
      ldax B ; cargamos en A el contenido de la posición de memoria contenida en BC
      add E ; A <= A + E
      stax B ; guarda el contenido de A en la posición de memoria almacenada en BC
      inx H ; HL <= HL + 1
      inx B ; BC <= BC + 1
      dcr D ; DE <= DE - 1
      jp loop ; si A < 0  el programa continúa, si no va a la etiqueta loop
      call sub_codificadora ; push del contenido del PC en la pila y el Pc toma el valor sub_codificadora
hlt ; parada del procesador
```

Código de la subrutina:

```
.org 150h
sub_codificadora:
      push H ; push en la pila de los contenidos de HL
      push B ; igual con BC
      push D ; igual con DE
      push PSW ; push en la pila de AF
      mvi B, clave ; carga el contenido de clave en el registro B
codifica:
      mov A, M ; cargamos en A el contenido de la dirección de memoria contenida en HL
      xri clave ; A <= [clave] XOR A
      mov M, A ; cargamos en A el contenido de la dirección de memoria contenida en HL
      inx H ; HL <= HL + 1
      dcr D ; DE <= DE - 1
      jp codifica ; si A < 0  el programa continúa, si no va a la etiqueta codifica
      pop PSW ; pop en la pila que se almacena en AF
      pop D ; pop en la pila que se almacena en DE
      pop B ; pop en la pila que se almacena en BC
      pop H ; pop en la pila que se almacena en HL
ret
```

Conclusiones:

Finalmente me he sentido más cómodo con el simulador utilizado en esta práctica tras haber aprendido bastante sobre todas las instrucciones y opciones que ofrece, así como con la manera en la que se gestionan sus registros, que en ocasiones van en parejas.