Organización del Computador I

Práctica 3: Arquitectura del CPU

1er cuatrimestre 2022

Índice

1. Ejercicio 1

1.a. Pseudocódigo: leftShift

```
while posiciones > 0
    valor = valor + valor
    posiciones = posiciones - 1
endwhile
```

1.b. Assembler: leftShift

```
; R0 = Valor a shiftear.
; R1 = Cantidad de posiciones a shiftear.
ciclo: SUB R1, 0x0001 ; Restamos 1 al contador de posiciones R1.
       JNEG fin       ; Si R1 es <= 0 terminamos.
       ADD R0, R0     ; Sumarse a sí mismo es lo mismo que valor * 2.
       JMP ciclo      ; Saltamos a la etiqueta ciclo.
fin:   RET            ; Retornamos de la subrutina.
```

1.c. Otros registros

La rutina propuesta no altera los valores de los otros registros.

2. Ejercicio 2

Pedimos como precondición que la longitud del vector sea al menos 1.

2.a. Pseudocódigo: minMax

```
max = vector[0]
min = vector[0]
i = 0
while i < longitud(vector)
    if vector[i] > max then
        max = vector[i]
    endif
    if vector[i] < min then
        min = vector[i]
    endif
    i = i + 1
endwhile
```

2.b. Assembler: minMax

```
; R0 = Posición de inicio del vector.
; R1 = Longitud del vector.
; R2 = Valor del máximo.
; R3 = Valor del mínimo.
main:     MOV R2, [R0]   ; Asignamos el primer elemento del vector como el máximo.
          MOV R3, [R0]   ; Asignamos el primer elemento del vector como el mínimo.
checkMax: CMP R2, [R0]   ; Comparamos el elemento del vector con el máximo.
          JGE checkMin   ; Si es <= que el máximo saltamos.
          MOV R2, [R0]   ; Guardamos el nuevo máximo.
checkMin: CMP R3, [R0]   ; Comparamos el elemento del vector con el mínimo.
          JLE next       ; Si es >= que el mínimo saltamos.
          MOV R3, [R0]   ; Guardamos el nuevo mínimo.
next:     ADD R0, 0x0001 ; Avanzamos al siguiente elemento del vector.
          SUB R1, 0x0001 ; Restamos 1 a la longitud.
          JNE checkMax   ; Si aún no llegamos a 0 repetimos el ciclo.
fin:      RET            ; Retornamos de la subrutina.
```

3. Ejercicio 3

3.a. Pseudocódigo: sumar64

El algoritmo para sumar en complemento a 2 es sumar bit a bit (no importa el total de bits). Por lo tanto, si la ALU de ORGA1 opera con palabras de 16 bits a la vez, tendremos que hacer en total 4 sumas: sumamos la primer palabra (los primeros 16 bits), luego la segunda palabra, y así sucesivamente. La primer suma la hacemos con ADD, y las otras 3 con ADDC para contemplar el carry de la palabra anterior. Al realizar la operación de esta forma, el resultado final va a ser correcto, pero los flags de la ALU no sirven, ya que solo van a indicar lo sucedido con la suma de la última palabra.

3.b. Assembler: sumar64

```
; R0 = Posición del primer número de 64bits a sumar (lo llamamos A).
; R1 = Posición del segundo número de 64bits a sumar (lo llamamos B).
; R2 = Posición donde guardar el resultado.

; Sumamos la primer palabra de A con la primera de B.
; Ya lo guardamos en la primer palabra del resultado.
MOV  [R2], [R0]
ADD  [R2], [R1]
; Sumamos la segunda palabra de A con la segunda de B.
; Utilizamos el modo de direccionamiento indexado para obtener la palabra deseada
; dentro de los 64 bits. No podemos hacer ninguna cuenta en la ALU ya que perderíamos
; el flag de carry de la suma anterior.
MOV  [R2 + 0x0001], [R0 + 0x0001]
ADDC [R2 + 0x0001], [R1 + 0x0001]
; Sumamos la tercer palabra de A con la tercera de B.
MOV  [R2 + 0x0002], [R0 + 0x0002]
ADDC [R2 + 0x0002], [R1 + 0x0002]
; Sumamos la cuarta palabra de A con la cuarta de B.
MOV  [R2 + 0x0003], [R0 + 0x0003]
ADDC [R2 + 0x0003], [R1 + 0x0003]
; Retornamos de la subrutina.
RET
```

4. Ejercicio 4

4.a. Pseudocódigo: sumarVector64

```
resultado = 0
i = 0
while i < longitud(vector)
    resultado = resultado + vector[i]
    i = i + 1
endwhile
```

4.b. Assembler: sumarVector64

```
; R0 = Longitud del vector.
; R1 = Posición de inicio del vector.
; R3 = Posición donde guardar el resultado.
; R4 = Cantidad de elementos a sumar.

       ; Inicializamos el resultado en 0.
main:  MOV [R3], 0x0000
       MOV [R3 + 0x0001], 0x0000
       MOV [R3 + 0x0002], 0x0000
       MOV [R3 + 0x0003], 0x0000
       ; La cantidad de elementos a sumar es inicialmente la longitud del vector.
       MOV R4, R0
       ; Acomodamos los registros para que funcione la subrutina sumar64.
       ; Movemos la posición donde guardar el resultado a R2.
       ; También la movemos a R0 para que sumar64 haga la suma "in place".
       MOV R2, R3
       MOV R0, R3

       ; En cada ciclo restamos 1 de la cantidad de elementos a sumar (R4).
       ; Si la cantidad es <= 0 terminamos.
ciclo: SUB R4, 0x0001
       JNEG fin
       ; Invocamos la subrutina sumar64.
       ; El efecto será: [R2] = [R0] + [R1].
       ; Pero recordemos que R2 = R0 = R3, por lo tanto el efecto será: [R3] = [R3] + [R1].
       ; Y en [R1] tenemos el elemento actual del vector que queremos sumar al resultado.
       CALL sumar64
       ; Avanzamos R1 para que apunte a la posición del siguiente elemento.
       ; Hay que sumarle 4 palabras de 16 bits ya que los elementos ocupan 64 bits.
       ADD R1, 0x0004
       ; Repetimos el ciclo.
       JMP ciclo

fin:   RET
```

5. Ejercicio 5

Programa en assembler.

```
; R0 = x
; R1 = y
MOV R0, 0x0002
MOV R1, 0x0020
ADD R0, R1
```

Programa ensamblado para la máquina ORGA1.

Assembler	Cod. Op.	Destino	Fuente	Constante 1	Constante 2	Hex
MOV R0, 0x0002	0001	100000	000000	0000 0000 0000 0010	-	0x1800 0x0002
MOV R1, 0x0020	0001	100001	000000	0000 0000 0010 0000	-	0x1840 0x0020
ADD R0, R1	0010	100000	100001	-	-	0x2821

6. Ejercicio 6

Cod. Op.	Destino	Fuente	Constante 1	Constante 2	Assembler
0001	100000	000000	0000 0000 1111 1111	-	MOV R0, 0x00FF
0001	100001	000000	0001 0000 0000 0000	-	MOV R1, 0x1000
0010	100000	100001	-	-	ADD R0, R1

7. Ejercicio 7

Se asume que el 20 del enunciado es 0x0020.

a) MOV R1, 0x0020 $\equiv$ R1 = 0x0020

b) MOV R1, [0x0020] $\equiv$ R1 = [0x0020] = 0x0040

c) MOV R1, [[0x0020]] $\equiv$ R1 = [[0x0020]] = [0x0040] = 0x0060

d) MOV R1, R0 $\equiv$ R1 = 0x0030

e) MOV R1, [R0] $\equiv$ R1 = [0x0030] = 0x0050

f) MOV R1, [R0 + 0x0020] $\equiv$ R1 = [0x0030 + 0x0020] = [0x0050] = 0x0070

8. Ejercicio 8

8.a.