DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN
ESCUELA DE INGENIERÍA
PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE

IIC2343 - Arquitectura de Computadores (II/2023)

Actividad práctica 1

Sección 3 - Pauta de evaluación

Pregunta 1: Explique el código (1 ptos.)

Detalle, basándose en los nombres de las variables y *labels*, lo que realiza el siguiente fragmento de código desarrollado en el Assembly del computador básico visto en clases:

```
DATA:
  n           8
  x_value     12
  f_x         0
CODE:
  MOV A, (x_value)
  main:
    SHR A, A
    JCR case_2
  case_1:
    MOV B, A
    MOV A, (n)
    SUB A, 1
    CMP A, 0
    JEQ end
    MOV (n), A
    MOV A, B
    JMP main
  case_2:
      INC (f_x)
      JMP case_1
  end:
```

No es necesario que explique lo que realiza cada instrucción del programa. Basta con que detalle el objetivo general de este. En esta pregunta no se otorgará puntaje parcial.

Solución: En este caso, se observa que se incrementa el valor de `f_x` cada vez que se realiza el salto por *carry*. Ahora, es importante notar que este salto ocurre cuando la *flag* `C` del registro `Status` es igual a 1. Esto no ocurre por un *carry* de la suma, sino por ser el **bit descartado en la operación *shift* como se vio en clases**. Por lo tanto, si tomamos la variable `f_x` como $f(x)$, entonces el fragmento de código computa $f(x)$ como la cantidad de bits iguales a 1 que tiene el valor x de 8 bits (establecido a partir de n). No es necesario que se explicite la función, pero sí que se señale que el cómputo depende de los bits iguales a 1 dentro de x. No es válido señalar que es un contador de *carry*, ya que no es precisamente por un *carry* la razón por la que se activa la *flag* `C`.

Pregunta 2: Arregle el código (2 ptos.)

El siguiente programa, desarrollado en el Assembly del computador básico visto en clases, **debiese calcular el área de un triangulo dado la coordenadas de sus tres puntos en el plano**. Este asume que los puntos están ordenados, *i.e.* $P1_x < P2_x$ y $P1_y = P2_y < P3_y$.

```
DATA:
  area        0
  base        0
  altura      0
  P1          6   ; coordenada x del punto P1
              1   ; coordenada y del punto P1
  P2          8   ; coordenada x del punto P2
              1   ; coordenada y del punto P2
  P3          1   ; coordenada x del punto P3
              10  ; coordenada y del punto P3
  contador    0
CODE:
  MOV A, (P2)
  MOV B, (P1)
  SUB (base)
  MOV A, P3
  MOV B, P1
  SUB (altura)
  loop:
    MOV A, (altura)
    MOV B, (contador)
    CMP A, B
    JEQ end
    MOV A, (area)
    MOV B, (base)
    ADD (area)
    INC (contador)
    JMP loop
  end:
    MOV A, (area)
    SHL A, A
    MOV (area), A
```

Sin embargo, no lo hace correctamente. Si compila y ejecuta el código, se dará cuenta que no entrega el resultado esperado. Busque el error y, una vez encontrado, arréglelo para que el fragmento anterior entregue el resultado esperado. Suba el código completo como respuesta. Se otorga **1 pto.** del total si encuentra el error y trata de arreglarlo sin éxito, lo que debe estar comentado en el código. No se otorga puntaje si el programa no compila o se sube sin arreglo alguno.

Solución: Este código presenta dos problemas fundamentales:

1. La altura se está calculando como la resta de las direcciones de memoria de los puntos `P2` y `P3`, no como la resta de sus alturas.
2. El resultado de la multiplicación final se multiplica por 2 a través de un *shift left* y no se divide por dos a través de un *shift right*.

Con eso en consideración, podemos resolver el problema de la siguiente forma:

```
DATA:
  area        0
  base        0
  altura      0
  P1          6   ; coordenada x del punto P1
              1   ; coordenada y del punto P1
  P2          8   ; coordenada x del punto P2
              1   ; coordenada y del punto P2
  P3          1   ; coordenada x del punto P3
              10  ; coordenada y del punto P3
  contador    0
CODE:
  MOV A,(P2)
  MOV B,(P1)
  SUB (base)
  MOV B,P3
  INC B
  MOV A,(B)
  MOV B, P1
  INC B
  MOV B,(B)
  SUB (altura)
  loop:
    MOV A, (altura)
    MOV B, (contador)
    CMP A, B
    JEQ end
    MOV A,(area)
    MOV B,(base)
    ADD (area)
    INC (contador)
    JMP loop
  end:
    MOV A,(area)
    SHR (area)
```

No importa cómo se haya resuelto el problema siempre que el fragmento de código funcione cambiando los valores de las coordenadas de `P1`, `P2` y `P3`, siempre respetando los supuestos. Si no funciona, se otorga **1 pto.** si es que se entrega la explicación correcta del error dentro de la respuesta.

Pregunta 3: Elabore el código (3 ptos.)

Elabore, utilizando el Assembly del computador básico visto en clases, un programa que **determine si los elementos de un arreglo `seq` de largo `len` es una secuencia de Fibonacci**. El resultado debe almacenarse en una variable `is_fibonacci` señalando si la secuencia cumple con serlo (1) o no (0). Además, si la secuencia **no** es de Fibonacci, debe almacenar el índice del primer valor del arreglo que rompe la secuencia en `error_index`, siendo el primer índice igual a cero. Asuma que la secuencia de Fibonacci parte con los números 0 y 1. Utilice el siguiente fragmento de código como base para su programa:

```
DATA:
  is_fibonacci    0
  error_index     -1 ; Valor base, si no hay error no cambia.
  len             6
  seq             0
                  1
                  1
                  2
                  3
                  5
CODE:
```

Se otorgan **1.5 ptos.** por el cómputo correcto de `is_fibonacci`; y **1.5 ptos.** por el cómputo correcto de `error_index`. Por cada valor, se descuenta la mitad del puntaje si su programa falla como máximo en un caso; y no se otorga puntaje si falla en más de un caso.

Solución: En este caso, se hace lo siguiente:

1. Se verifica que `seq[0]` sea 0 y `seq[1]` sea 1.
2. Para `i` $<$ `len - 2`, se verifica que: `seq[i] + seq[i+1] == seq[i+2]`.
3. Si `seq[i] + seq[i+1] != seq[i+2]`, entonces `error_index = i+2`.
4. Si nunca ocurre lo anterior, entonces `is_fibonacci = 1`.

El puntaje se asigna tal como se detalla en el enunciado. En la siguiente página se incluye un programa de ejemplo que cumple lo pedido.

```
DATA:
  is_fibonacci    0
  error_index     -1 ; Valor base, si no hay error no cambia.
  len             6
  seq             0
                  1
                  1
                  2
                  3
                  5
  index           0  ; Valor auxiliar para recorrer la secuencia.
  current_value   0  ; Valor auxiliar para verificar Fibonacci.

CODE:
  check_initial_values:
    MOV A,(seq)
    CMP A,0
    JNE error_index_0
    MOV B,seq
    INC B
    MOV A,(B)
    CMP A,1
    JNE error_index_1
    JMP check_fibonacci_loop

  error_index_0:
    INC (error_index)
    JMP end
  error_index_1:
    MOV A,1
    MOV (error_index),A
    JMP end

  check_fibonacci_loop:
    MOV B,(index)
    MOV A,seq
    ADD B,A
    MOV A,(B)                ; A = seq[i]
    INC B
    MOV B,(B)                ; B = seq[i+1]
    ADD (current_value)      ; current_value = seq[i] + seq[i+1]
    MOV A,(index)
    ADD A,2
    MOV B,seq
    ADD B,A
    MOV A,(B)                ; A = seq[i+2]
    CMP A,(current_value)    ; current_value == seq[i+2] ?
    JNE set_error_index      ; current_value != seq[i+2] => error_index = i+2
    INC (index)
    MOV A,(index)
    ADD A,2
    CMP A,(len)
    JGE set_is_fibonacci
    JMP check_fibonacci_loop

  set_error_index:
    MOV A,(index)
    ADD A,2
    MOV (error_index),A
    JMP end

  set_is_fibonacci:
    INC (is_fibonacci)

  end:
```