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

IIC2343 - Arquitectura de Computadores (II/2023)

Actividad práctica 1

Sección 1 - 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         3
  x_value   24
  f_x       0
CODE:
  loop:
    MOV A,(n)
    CMP A,0
    JEQ end
    SUB A,1
    MOV (n),A
    MOV A,(x_value)
    SHR A,A
    MOV (x_value),A
    JMP loop
  end:
    MOV A,(x_value)
    MOV (f_x),A
```

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 realiza la operación `SHR` una cantidad n de veces. Si tomamos la variable `f_x` como $f(x)$, entonces el fragmento de código es equivalente a $f(x) = \frac{x}{2^n}$. No es necesario que se explicite la función, pero sí que se señale que el valor de x se divide n veces por 2.

Pregunta 2: Arregle el código (2 ptos.)

El siguiente programa, desarrollado en el Assembly del computador básico visto en clases, **debiese verificar si el número `n` es múltiplo de `m`**:

```
DATA:
  multiplo    0 ; si es múltiplo (1), si no (0)
  n           14
  m           3
CODE:
  MOV A, (n)
  MOV B, (m)
  loop:
    SUB A, B
    CMP A, 0
    JEQ es_multiplo
    JMP loop
  es_multiplo:
    INC (multiplo)
  end:
```

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: El problema consiste en que el código no controla el caso en el que `SUB A,B` llega a ser menor que cero, por lo que el programa termina solo para los números n que sí son múltiplos de m, pero no si no lo son. Si recordamos que las instrucciones de salto se pueden ocupar en cualquier momento y no necesariamente después de una instrucción `CMP`, podemos resolver el problema de la siguiente forma:

```
DATA:
  multiplo    0 ; si es múltiplo (1), si no (0)
  n           16
  m           3
CODE:
  MOV A,(n)
  MOV B,(m)
  loop:
    SUB A,B
    JLT end
    CMP A,0
    JEQ es_multiplo
    JMP loop
  es_multiplo:
    INC (multiplo)
  end:
```

No importa cómo se haya resuelto el problema siempre que el fragmento de código funcione cambiando los valores de `n` y `m`. 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 **calcule el área del triángulo que se forma a partir de tres puntos en un plano de dos dimensiones**. Las coordenadas de los puntos se guardarán individualmente en la memoria, como se muestra en el fragmento de código a continuación:

```
; Calcular el área de un triángulo que tiene un segmento paralelo al eje x
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
          10  ; coordenada y del punto P2
  P3      1   ; coordenada x del punto P3
          10  ; coordenada y del punto P3
CODE:
```

Puede asumir lo siguiente:

- Los tres puntos **siempre** formarán un triángulo válido.
- **Siempre** habrá un lado del triángulo paralelo al eje x.
- Los puntos `P1`, `P2` y `P3` **no siguen un orden predeterminado**. Es su tarea encontrar los puntos que forman la base y altura del triángulo.

Se otorgan **0.75 ptos.** por encontrar correctamente la base; **0.75 ptos.** por encontrar correctamente la altura; y **1.5 ptos.** por encontrar correctamente el área del triángulo. 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, es importante seguir el siguiente flujo:

1. Detectar la base según los dos puntos cuyos valores de coordenada y sean iguales.
2. Calcular la base y asegurar que el valor sea positivo.
3. A partir de la base encontrada, calcular la altura con el tercer punto y asegurar que el valor sea positivo.
4. Multiplicar ambos valores (por ejemplo, con sumas sucesivas) y realizar una división entera por 2 al final.

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.

```
; Calcular el área de un triángulo que tiene un segmento paralelo al eje x
DATA:
  area    0
  base    0
  altura  0
  caso    0
  P1      6   ; coordenada x del punto P1
          1   ; coordenada y del punto P1
  P2      1   ; coordenada x del punto P2
          10  ; coordenada y del punto P2
  P3      8   ; coordenada x del punto P3
          10  ; coordenada y del punto P3
  count   0   ; Variable auxiliar para multiplicar
  sum_val 0   ; Variable auxiliar para multiplicar
  caso 0
CODE:
  ; Detección de base
  detect_base:
    MOV B,P1
    INC B
    MOV A,(B)
    MOV B,P2
    INC B
    MOV B,(B)
    CMP A,B
    JEQ get_P1_P2_base      ; P1.y == P2.y
    MOV B,P3
    INC B
    MOV B,(B)
    CMP A,B
    JEQ get_P1_P3_base      ; P1.y == P3.y
    JMP get_P2_P3_base      ; P2.y == P3.y
  get_P1_P2_base:
    INC (caso)
    MOV A,(P1)
    SUB A,(P2)
    JGE set_P1_P2_base ; Si es negativo, calculamos el complemento de 2 para obtener el valor positivo
    NOT A,A
    ADD A,1
    set_P1_P2_base:
      MOV (base),A
      ; Para calcular altura: A = P2.y, B = P3.y
      MOV B,P2
      INC B
      MOV A,(B)
      MOV B,P3
      INC B
      MOV B,(B)
      JMP get_height
  get_P1_P3_base:
    MOV A,(P1)
    SUB A,(P3)
    JGE set_P1_P3_base ; Si es negativo, calculamos el complemento de 2 para obtener el valor positivo
    NOT A,A
    ADD A,1
    set_P1_P3_base:
      MOV (base),A
      ; Para calcular altura: A = P1.y, B = P2.y
      MOV B,P1
      INC B
      MOV A,(B)
      MOV B,P2
      INC B
      MOV B,(B)
      JMP get_height
```

```
get_P2_P3_base:
  MOV A,(P2)
  SUB A,(P3)
  JGE set_P2_P3_base ; Si es negativo, calculamos el complemento de 2 para obtener el valor positivo
  NOT A,A
  ADD A,1
  set_P2_P3_base:
    MOV (base),A
    ; Para calcular altura: A = P1.y, B = P2.y
    MOV B,P1
    INC B
    MOV A,(B)
    MOV B,P2
    INC B
    MOV B,(B)
    JMP get_height

get_height:
  SUB A,B
  JGE set_height
  NOT A,A
  ADD A,1
  set_height:
    MOV (altura),A

get_area:
  ; A = Altura, B = Base
  MOV B,(base)
  CMP A,B
  JGT base_count
  ; Usamos de contador el menor valor para que ejecute en menos iteraciones.
  height_count:
    MOV (count),A
    MOV (sum_val),B
    JMP start_sum_loop
  base_count:
    MOV (count),B
    MOV (sum_val),A
  start_sum_loop:
    MOV B,(sum_val)
  sum_loop:
    MOV A,(area)
    ADD (area)
    MOV A,(count)
    SUB A,1
    MOV (count),A
    CMP A,0
    JLE set_area
    JMP sum_loop
  set_area:
    MOV A,(area)
    SHR (area)   ; Area = Base * Altura // 2
```