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

IIC2343 - Arquitectura de Computadores (I/2026)

Interrogación 1

Respuestas sin desarrollo o justificación no tendrán puntaje.

Instrucciones

Lea atentamente los enunciados. Responda cada pregunta en hojas separadas. Debe escribir su nombre, número de alumno y número de lista. La prueba tiene 3 preguntas; debe contestarlas todas.

Pregunta 1: Representación de números enteros (6 ptos.)

(a) (2 ptos.) En un videojuego, usted ha logrado acumular un total de 127 vidas. Al seguir jugando, usted adquiere una vida adicional, pero comete un error y pierde una de ellas. En este escenario, puede leer en la pantalla el texto "`GAME OVER`", pero no entiende cómo perdió considerando el total de vidas que tenía acumulado. A partir de los contenidos del curso en torno a representaciones numéricas, explique por qué podría haber sucedido la situación antes descrita.

> **Solución:** Una posible explicación es que el total de vidas se represente como un número positivo a través de una secuencia de 7 bits. En este escenario, lo que ocurrió fue un *overflow*: 1111111b + 0000001b = 0000000b (dado que el bit de acarreo se descarta). Luego, como el jugador pierde una vida, el juego al revisar el total nota que este es igual a cero, generando la condición de término de juego. Otra explicación plausible es que el total de vidas se represente como un número entero a través de una secuencia de 8 bits, pero en este caso sería necesario asumir, de forma explícita, que el juego interpreta un total de vidas menor o igual a cero como condición de término de juego. Se otorga **1 pto.** por señalar uno de los casos anteriores y **1 pto.** por explicarlo.

(b) (2 ptos.) Escriba la representación en decimal del resultado de la operación 0xBABA + 0x4542. Este resultado debe interpretarse como binario en complemento de dos.

> **Solución:** En primer lugar, se tiene que 0xBABA + 0x4542 es igual a 0xFFFC. Luego, en base binaria este resultado equivale a 1111111111111100b. Si se interpreta como binario en complemento de dos, este valor es **negativo** e igual a $-C_2(1111111111111100\text{b}) =$

$-0000000000000100\text{b} = -4$. Se otorga **1 pto.** por obtener correctamente el resultado de la operación, ya sea en base hexadecimal o binaria; y **1 pto.** por obtener correctamente su representación en base decimal.

(c) (2 ptos.) A partir de la fórmula de transformación de bases:

$$\sum_{k=0}^{n-1} s_k \times b^k$$

Justifique por qué funciona el algoritmo de conversión directa de base hexadecimal ($b = 16$) a base binaria ($b = 2$) visto en clases.

Solución: El funcionamiento se justifica porque 16 es potencia de 2, es decir, ambas bases son potencia del mismo número. Entonces, si tomamos de ejemplo un número cualquiera, como 0x12 = 18 = 00010010b, con la fórmula de transformación de bases podemos hacer lo siguiente:

$$\begin{aligned}
0x12 = \sum_{k=0}^{1} s_k \times 16^k &= s_0 \times 16^0 + s_1 \times 16^1 \\
&= 2 \times 1 + 1 \times 16 \\
&= 2 \times 2^0 + 1 \times 2^4 \\
&= 2 \times 2^0 + 0 \times (2^1 + 2^2 + 2^3) + 1 \times 2^4 + 0 \times (2^5 + 2^6 + 2^7) \\
&= \sum_{k=0}^{7} s_k \times 2^k = 00010010\text{b}
\end{aligned}$$

Justificando su funcionamiento. Se otorga **1 pto.** por justificar el funcionamiento del algoritmo, y **1 pto.** por ilustrar su correctitud a través del uso de la fórmula de transformación de bases.

Pregunta 2: Circuitos digitales y almacenamiento (6 ptos.)

(a) (3 ptos.) Usted se encuentra desarrollando el sistema de combates para un nuevo videojuego. Para optimizar el uso de memoria, ha decidido implementar un sistema de tipos reducido utilizando solo 2 bits. Para mantener un estándar, los tipos se han asignado en estricto orden alfabético: **Fantasma (00)**, **Lucha (01)**, **Psíquico (10)** y **Siniestro (11)**. Durante un turno, el sistema compara el tipo del monstruo que ataca (A_1, A_0) con el tipo del monstruo que defiende (D_0), que solo puede ser fantasma o lucha. La efectividad del ataque se rige por la siguiente tabla:

Ataca \ Defiende	**0 (Fantasma)**	**1 (Lucha)**
00 (Fantasma)	$\times 2$	$\times 1$
01 (Lucha)	$\times 1/2$	$\times 1$
10 (Psíquico)	$\times 1$	$\times 2$
11 (Siniestro)	$\times 2$	$\times 1/2$

Dependiendo del resultado de la pelea, la pantalla del juego debe encender distintas señales (R_1 a R_3) para mostrar un mensaje de texto al jugador, definidos de la siguiente forma:

- R_1: El ataque es **Neutral** ($\times 1$).
- R_2: El ataque es **Superefectivo** ($\times 2$).
- R_3: El ataque es **Poco efectivo** ($\times 1/2$).

A partir de este contexto, diseñe para cada salida R_i de la pantalla, un circuito combinacional que reciba las señales de entrada del atacante (A_1, A_0) y del defensor (D_0) y compute si el mensaje R_i debe mostrarse (1) o no (0). Exprese claramente las funciones lógicas utilizadas. Puede, si lo desea, diseñar un único circuito para computar todas las salidas simultáneamente, aprovechando compuertas compartidas.

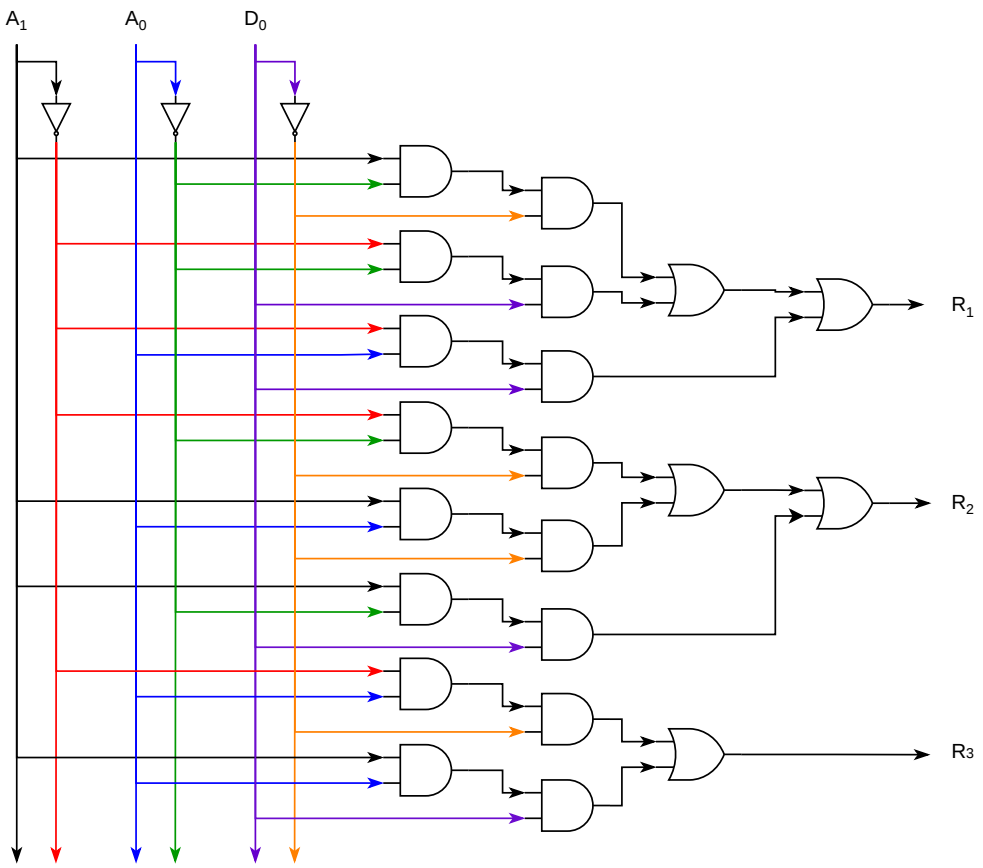
Solución: A continuación, la tabla de verdad para las salidas R_i a partir del atacante (A_1, A_0) y el defensor (D_0):

A_1	A_0	D_0	R_1	R_2	R_3
0	0	0	0	1	0
0	1	0	0	0	1
1	0	0	1	0	0
1	1	0	0	1	0
0	0	1	1	0	0
0	1	1	1	0	0
1	0	1	0	1	0
1	1	1	0	0	1

A partir de ella, se puede hacer uso de *minterms* y *maxterms* para obtener la expresión que representa cada salida. A continuación, se listan expresiones válidas para cada segmento, junto con su reducción a través de equivalencias lógicas.

- $R_1 = (A_1 \wedge \overline{A_0} \wedge \overline{D_0}) \vee (\overline{A_1} \wedge \overline{A_0} \wedge D_0) \vee (\overline{A_1} \wedge A_0 \wedge D_0)$
- $R_2 = (\overline{A_1} \wedge \overline{A_0} \wedge \overline{D_0}) \vee (A_1 \wedge A_0 \wedge \overline{D_0}) \vee (A_1 \wedge \overline{A_0} \wedge D_0)$
- $R_3 = (\overline{A_1} \wedge A_0 \wedge \overline{D_0}) \vee (A_1 \wedge A_0 \wedge D_0)$

Finalmente, se puede diseñar un circuito a partir de estas expresiones.



Es importante notar que se evaluará la correctitud del circuito diseñado, no que sea igual al de esta pauta. Se otorga **1 pto.** por el circuito correctamente diseñado de cada segmento. En cada caso se otorga la mitad del puntaje si existe, como máximo, una combinación de señales de entrada que generen una salida incorrecta.

(b) (3 ptos.) Para aplicar los contenidos del curso en su hogar, decide implementar a través de *hardware* una versión reducida del minijuego "*Bowser's Big Blast*" de Mario Party 2 ™.



Para su implementación debe diseñar, utilizando **solo** compuertas lógicas, *enablers*, multiplexores y flip-flops, un contador secuencial de 2 bits que decrementa en cada flanco de subida a partir de una señal *Boom* según las siguientes condiciones:

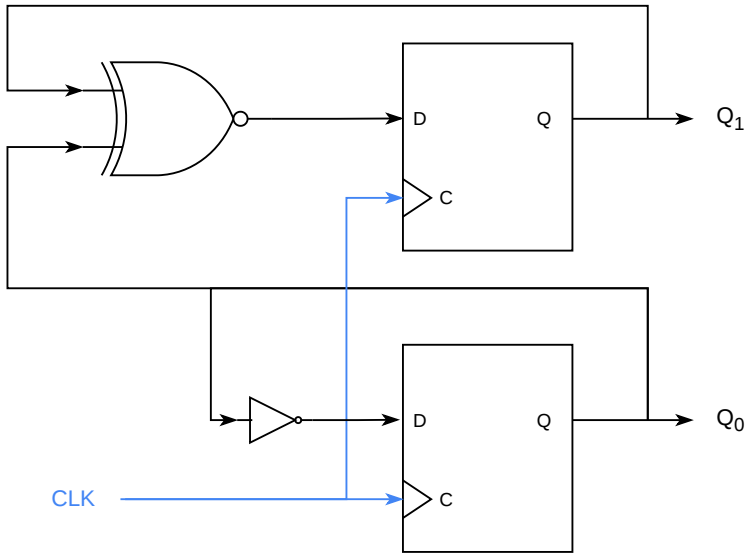
- El contador parte en 3 ($Q_1Q_0 = 11$). Asuma que los flip-flops del contador parten en este estado.
- El contador decrementa en el flanco de subida si, y solo si $Boom = 1$.
- El contador vuelve a 3 ($Q_1Q_0 = 11$) si $Boom = 0$, independiente de su valor actual.

Puede asumir que su contador hace *overflow* en caso de decrementar estando en su mínimo valor, *i.e.* asuma que Q_1Q_0 pasa de 00 a 11 en caso de decrementar con $Boom = 1$.

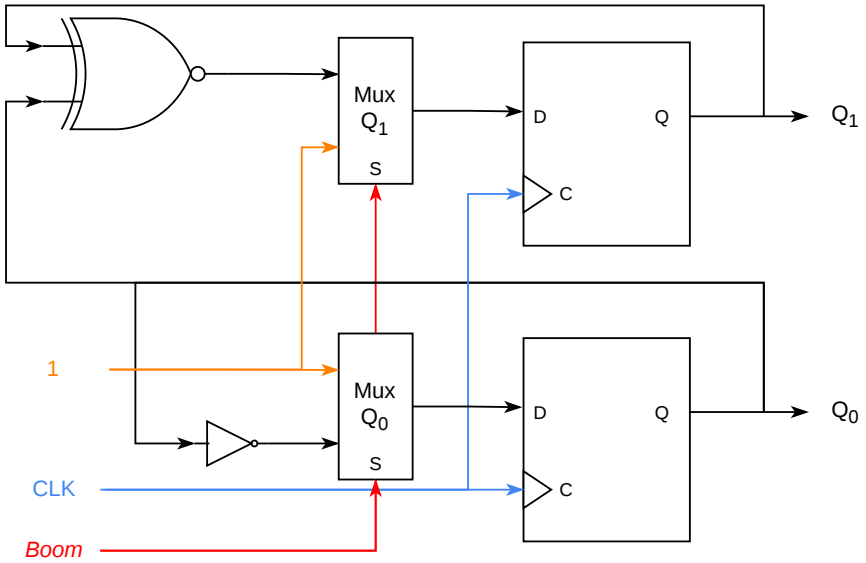
Solución: Si denotamos por $Q_1^tQ_0^t$ el estado del contador en un ciclo t de la señal de control `CLK` y $Q_1^{t+1}Q_0^{t+1}$ su estado en el próximo ciclo, se puede elaborar la siguiente tabla de verdad:

Q_1^t	Q_0^t	Q_1^{t+1}	Q_0^{t+1}
0	0	1	1
0	1	0	0
1	0	0	1
1	1	1	0

Se deduce que $Q_0^{t+1} = \overline{Q_0^t}$ y $Q_1^{t+1} = \overline{(Q_0^t \oplus Q_1^t)}$. Es decir, Q_0 será igual a `NOT` Q_0, mientras que Q_1 será igual a Q_0 `XNOR` Q_1. Haciendo uso de flip-flops y las compuertas antes señaladas, se llega al siguiente diagrama:



Ahora, esta es una solución intermedia porque no contempla el caso en el que $Boom = 0$, donde el contador debería actualizarse a $Q_1Q_0 = 11$. Para ello, agregaremos un multiplexor para cada señal Q_i que seleccionará una entrada constante igual a 1 si $Boom = 0$ y la salida de las compuertas descritas anteriormente si $Boom = 1$:



Es importante notar que se evaluará la correctitud del contador diseñado, no que sea igual al de esta pauta. Se otorgan **1.5 ptos.** por diseñar correctamente un contador de 2 bits decremental, y **1.5 ptos.** por actualizar su valor correctamente a partir de la señal *Boom*. Por cada criterio, se otorga la mitad del puntaje si existe, como máximo, un error de implementación.

Pregunta 3: Computador básico con saltos (6 ptos.)

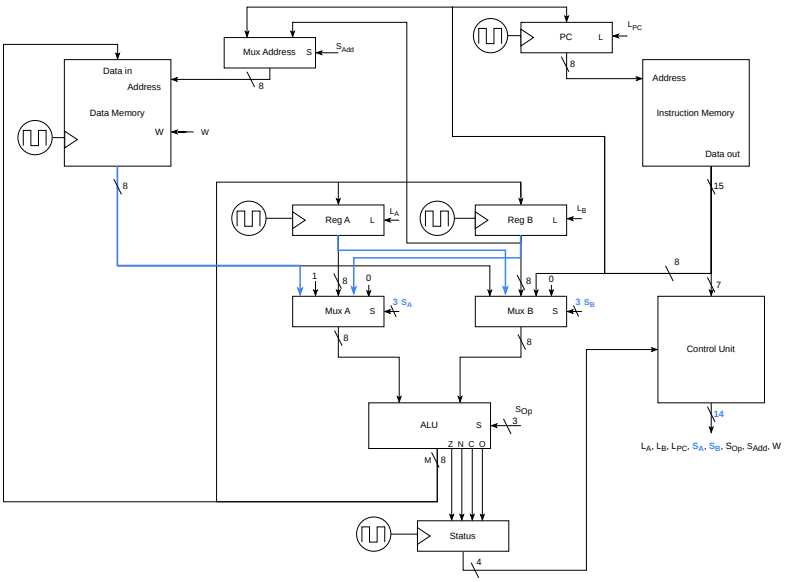
En los ítems **(a)**, **(b)** y **(c)**, se le describirán instrucciones que deberá implementar en el computador básico a partir del diagrama adjunto. Para cada una de ellas, debe incluir la combinación **completa** de señales que las ejecutan. En las señales de carga/escritura, debe indicar si se activan (1) o no (0); en las señales de selección, debe indicar el **nombre** de la entrada escogida ("-" si no afecta). Debe realizar todas las modificaciones de *hardware* en un solo diagrama. Además, puede implementar instrucciones sin modificaciones si es que la arquitectura se lo permite, indicando las señales de control necesarias. **Cada instrucción debe ejecutarse en un ciclo**.

(a) (1 pto.) `SUB B,B,A`. Almacena en el registro `B` el resultado de la operación `B - A`.

(b) (1 pto.) `ADD (B),Lit`. Almacena en `Mem[B]` el resultado de la operación `Mem[B] + Lit`.

(c) (1 pto.) `RESTART`. Carga en el registro `PC` el valor 0.

Solución: A continuación, se muestra el diagrama con las conexiones necesarias para ejecutar todas las instrucciones solicitadas, incluyendo una descripción de cada conexión añadida y lo que habilita.



- Para la instrucción `SUB B,B,A`, se conecta la salida del registro `B` con el multiplexor `Mux A`. Esto permite poder seleccionar `B` como primer operando de la ALU en la operación de resta.
- Para la instrucción `SUB B,B,A`, se conecta la salida del registro `A` con el multiplexor `Mux B`. Esto permite poder seleccionar `A` como segundo operando de la ALU en la operación de resta. Ahora, como este multiplexor tiene 5 entradas, requiere que su señal de selección S_B incremente a 3 bits y la salida de la `Control Unit` incremente a 13 bits.

- Para la instrucción `ADD (B),Lit`, se conecta la salida de la `Data Memory` con el multiplexor `Mux A`. Esto permite poder seleccionar `DOUT` como primer operando de la ALU, lo que a su vez permite que este valor pueda ser operado con un literal seleccionado desde el `Mux B`. Ahora, como este multiplexor tiene 5 entradas, requiere que su señal de selección S_A incremente a 3 bits y la salida de la `Control Unit` incremente a 14 bits.
- Para la instrucción `RESTART`, no es necesario hacer modificaciones. No obstante, es necesario explicitar que, para la combinación de señales a utilizar, es necesario que **el literal sea igual a cero**.

A continuación, la tabla de señales resultante:

Instrucción	L_A	L_B	L_{PC}	W	S_A	S_B	S_{OP}	S_{Add}
`SUB B,B,A`	0	1	0	0	B	A	SUB	-
`ADD (B),Lit`	0	0	0	1	DOUT	LIT	ADD	B
`RESTART`	0	0	1	0	-	-	-	-

Por cada instrucción, se otorgan:

- **0.5 ptos.** por *hardware* correctamente modificado; **0.3 ptos.** si existe **un solo error** de implementación; **0.1 ptos.** si existen **solo dos errores**; y **0 ptos.** en otro caso.
- **0.5 ptos.** por entregar una combinación de señales correcta; **0.3 ptos.** si existe **un solo error**; **0.1 ptos.** si existen **solo dos errores**; y **0 ptos.** en otro caso. Si una instrucción se define en más de un ciclo, tampoco se otorga puntaje en este criterio. Si la instrucción se implementa correctamente sin modificación de *hardware*, se otorga el puntaje completo.

Nota: A continuación, se describen otras posibles formas de implementar la instrucción `RESTART`:

- Añadir un `Mux PC` que reciba tanto el valor literal como una señal de valor 0 de 8 bits y seleccionar este último con S_{PC} = `ZERO`.
- Añadir como salida de la `Control Unit` una nueva señal R_{PC} y conectarla con la entrada *reset* del contador `PC`, de forma que si $R_{PC} = 1$ su valor sea reseteado.

(d) (3 ptos.) Los programas **(1)** y **(2)** están escritos en el Assembly del computador básico del curso, con la diferencia de que **(2)** usa las instrucciones implementadas en los ítems anteriores. Para cada programa, indique los valores de los registros `A` y `B` al finalizar la ejecución. Si no incluye desarrollo o no justifica cómo llegó a los valores obtenidos, **no se otorgará puntaje**.

(1)

```
DATA:
  var1 -1
  var2 -3
CODE:
  MOV A,(var1)
  MOV B,(var2)
  ADD A,B
  JLT case_1
  CMP A,B
  JGT case_2
  JMP case_3
  case_1:
    INC B
  case_2:
    INC B
  case_3:
    INC B
  end:
```

(2)

```
DATA:
  arr 9
      6
  index 0
  len 2
  target 7
  diff 127
  res -1
CODE:
  MOV A,(index)
  MOV B,(len)
  CMP A,B
  JEQ end
  MOV B,arr
  ADD B,A
  MOV A,(B)
  MOV B,(target)
  SUB B,B,A
  JGE continue_1
  MOV A,0
  SUB A,B
  MOV B,A
  continue_1:
    MOV A,(diff)
    CMP A,B
    JLE continue_2
    MOV (diff),B
    MOV A,(index)
    MOV (res),A
  continue_2:
    MOV B,index
    ADD (B),1
    RESTART
  end:
    MOV A,(diff)
    MOV B,(res)
```

Solución: El fragmento de código **(1)**, como resultado negativo de la suma entre las variables `var1` y `var2`, genera un salto a `case_1` y, por ende, la ejecución de tres veces consecutivas de `INC B`. El fragmento de código **(2)**, por otra parte, computa el valor absoluto de la resta entre los elementos de `arr` y un `target`, de forma que se guarde en `res` el índice del elemento más cercano a `target` y en `diff` la diferencia entre ambos valores. A continuación, se muestra el resultado de la ejecución para ambos fragmentos:

(1)

```
DATA:
  var1 -1        ; MEM[0] = -1
  var2 -3        ; MEM[1] = -1
CODE:
  MOV A,(var1)   ; A = MEM[0] = -1
  MOV B,(var2)   ; B = MEM[1] = -3
  ADD A,B        ; A = -1 + -3 = -4
  JLT case_1     ; N = 1 -> Salta
  CMP A,B        ; No se ejecuta
  JGT case_2     ; No se ejecuta
  JMP case_3     ; No se ejecuta
  case_1:
    INC B        ; B += 1 -> B = -2
  case_2:
    INC B        ; B += 1 -> B = -1
  case_3:
    INC B        ; B += 1 -> B = 0
  end:           ; A = -4, B = 0
```

(2)

```
DATA:
  arr 9          ; MEM[0] = 9
      6          ; MEM[1] = 6
  index 0        ; MEM[2] = 0
  len 2          ; MEM[3] = 2
  target 7       ; MEM[4] = 7
  diff 127       ; MEM[5] = 127
  res -1         ; MEM[6] = -1
CODE:
  MOV A,(index)     ; A = MEM[0] (index): 0, 1
  MOV B,(len)       ; B = MEM[3] = 2 (len)
  CMP A,B           ; A - B (index - len)
  JEQ end           ; if Z = 1 -> Salta a end (luego de 2 it.)
  MOV B,arr         ; B = 0 (dirección arr)
  ADD B,A           ; B += index (dirección de memoria arr[i])
  MOV A,(B)         ; A = arr[i]
  MOV B,(target)    ; B = 7 (target)
  SUB B,B,A         ; B = target - arr[i]
  JGE continue_1    ; if N = 0 -> Salta a continue_1
  MOV A,0           ; Si no, se computa:
  SUB A,B           ; A = - (target - arr[i]) = arr[i] - target
  MOV B,A           ; B = arr[i] - target (valor absoluto)
  continue_1:
    MOV A,(diff)    ; A = MEM[5] (diff)
    CMP A,B         ; A - B (diff final - diff actual)
    JLE continue_2  ; if N = 1 -> Salta a continue_2
    MOV (diff),B    ; e.o.c. diff = diff actual
    MOV A,(index)   ; A = MEM[0] (index)
    MOV (res),A     ; res = A (arr[i] más cercano a target)
  continue_2:
    MOV B,index     ; B = 2 (dirección index)
    ADD (B),1       ; MEM[2] +=1 (aumenta index)
    RESTART         ; Salto a primera instrucción
  end:
    MOV A,(diff)    ; A = 1 (target = 7 - arr[1] = 6)
    MOV B,(res)     ; B = 1 (index i = 1)
```

Para cada fragmento, se otorgan **0.75 ptos.** por registro correctamente computado siempre que su cómputo esté respaldado por desarrollo o justificación. Si no se llega a ningún resultado final correcto por **error de arrastre**, se otorgan **0.5 ptos.** en el fragmento.