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

IIC2343 - Arquitectura de Computadores (I/2026)

Ayudantía 6: Repaso I1

Ayudantes: Alberto Maturana (alberto.maturana@uc.cl), Mario Rojas (mario.denzel@estudiante.uc.cl, Nicolás Romo Aubele (nroma@uc.cl)

Pregunta 1: Representaciones Numéricas

(a) Escriba la representación en decimal del resultado de la operación 0xF0CA + 0x0F30. Este resultado debe interpretarse como binario en complemento de dos.

Solución: En primer lugar, se tiene que 0xF0CA + 0x0F30 es igual a 0xFFFA. Luego, en base binaria este resultado equivale a 1111111111111010b. Si se interpreta como binario en complemento de dos, este valor es **negativo** e igual a $-C_2(1111111111111010\text{b}) = -0000000000000110\text{b} = -6$. Se otorgan **0.75 ptos.** por obtener correctamente el resultado de la operación, ya sea en base hexadecimal o binaria; y **0.75 ptos.** por obtener correctamente su representación en base decimal.
Nota: En la evaluación, se explicita que el resultado se representa con 16 bits.

(b) Considere una función $C_B(x)$ definida para un número x en base B, tal que $x + C_B(x) = 0$, donde la suma se realiza dígito a dígito en base B y sobre una cantidad fija de cifras. Esta función representa el **complemento en base** B de x. Sabiendo que $C_2(x)$ corresponde al complemento a dos de x (estudiado en clases), defina paso a paso el procedimiento para calcular el complemento en base 4 de un número x, es decir, $C_4(x)$. Además, considere que la representación del cero debe ser única, formada exclusivamente por ceros en todos sus dígitos, evitando representaciones alternativas (como ocurre con el complemento a uno). Justifique cómo su procedimiento garantiza que se cumple $x + C_4(x) = 0$ en base 4, utilizando una cantidad fija de dígitos.

Solución: El procedimiento, en este caso, no es más que la generalización del complemento de 2:

1. Para cada dígito x_i de un número $x = x_{N-1}...x_0$ en base B de N dígitos, definir $x'_i = (B - 1 - x_i)_B$
2. Construir el complemento en base B de x como $C_B(x) = x'_{N-1}...x'_0 + 1$

Este funciona porque, al sumar x_0 con x'_0, se tendrá que:

$$x_0 + x'_0 = (B-1)_B$$

Luego, al sumar la unidad extra definida en el paso 2:

$$x_0 + x'_0 + 1 = (B-1+1)_B = (B)_B = 10_B$$

Entonces, la suma resultará en un valor igual a B que, al no tener un dígito asociado en su propia base, será igual a 10_B. Esto **equivale a una suma igual a cero con acarreo de salida igual a 1**, por lo que $(x + C_B(x))_0 = 0$. Finalmente, esto derivará en el siguiente resultado para el resto de los dígitos de $x + C_B(x)$:

$$(x+C_B(x))_i = (x_i+x'_i+carry_{i-1}) = (x_i+x'_i+1) = (B-1+1)_B = B_B = 10_B = 0, \quad i > 0$$

Así, se evidencia que $x + C_B(x)$ será igual a una secuencia de N dígitos iguales a cero con un acarreo de salida final igual a 1, que es el resultado esperado. Para computar $C_4(x)$, simplemente se reemplaza B por 4 en el procedimiento planteado. Un ejemplo a continuación para computar el complemento en base 4 de 123_4:

$$C_4(123_4) = 210_4 + 1 = 211_4$$

$$\begin{array}{r} carry \rightarrow 110 \\ 123_4 \\ +\ 211_4 \\ \hline 1000_4 \end{array}$$

Otra opción válida es definir $C_B(x)$ de N dígitos como:

$$C_B(x) = (B_B)^N - x$$

Como la representación de B en su propia base es igual a 10_B, se tiene:

$$C_B(x) = 10_B^N - x$$

Podemos evaluar su correctitud reordenando la igualdad de la siguiente forma:

$$C_B(x) + x = 10_B^N$$

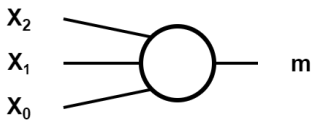
10_B^N, independiente de la base, tendrá un total de $N+1$ dígitos, por lo que al truncar el resultado en los N dígitos menos significativos, se descartará naturalmente el dígito igual a 1 que interpretamos como el acarreo de salida. Si aplicamos esta definición sobre el mismo ejemplo anterior, vemos que se llega al mismo resultado:

$$C_4(123) = 1000_4 - 123_4 = 211_4$$

Se otorga **1 pto.** por definir un procedimiento correcto; y **1 pto.** por justificación. No es necesario proponer un método para $C_B(x)$, pero sí para $C_4(x)$.

Pregunta 2: Lógica Digital I1-2024-1

Definiremos una nueva compuerta lógica llamada `Mayoria`. Esta recibirá tres señales de entrada de 1 bit y su salida, de 1 bit, corresponderá al valor que más se repite. Por ejemplo, si tenemos $X_0 = 0$, $X_1 = 0$, $X_2 = 1$, la salida de la compuerta `Mayoria` es 0.



A partir de esta compuerta, responda los incisos (a) y (b).

(a) Diseñe la compuerta lógica `Mayoria`, de 3 entradas, a partir de las siguientes compuertas: `AND`, `OR`, `NOT`, `XOR`, `NAND` o `NOR`.

Solución: Si tenemos tres entradas X_0, X_1 y X_2 solo pueden darse dos casos:

- Que el valor que más se repite esté en las tres entradas (3 a 0).
- Que el valor que más se repite esté en dos de las tres entradas (2 a 1).

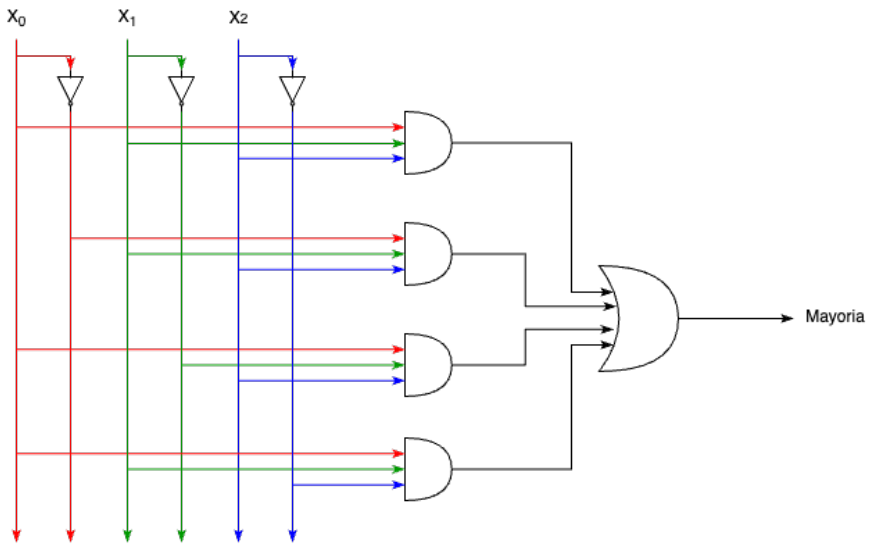
X_0	X_1	X_2	m
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

De esta manera, usamos *minterms* o *maxterms* (*minterms* en este caso) para obtener la expresión que representa la salida.

$$m = (X_0 \wedge X_1 \wedge X_2) \vee (\overline{X_0} \wedge X_1 \wedge X_2) \vee (X_0 \wedge \overline{X_1} \wedge X_2) \vee (X_0 \wedge X_1 \wedge \overline{X_2})$$

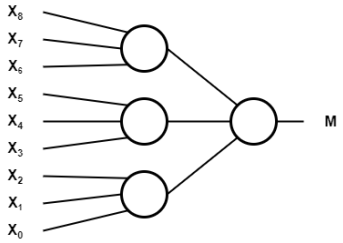
.

Finalmente, diseñamos el circuito a partir de esta expresión.



Es importante aclarar que se evaluará la correctitud del circuito diseñado, no que sea exactamente igual al incluido en la pauta.

(b) Haciendo uso de la compuerta `Mayoria`, se propone el siguiente circuito para determinar el valor que más se repite entre 9 *inputs*:



Indique si el circuito propuesto resuelve correctamente el problema planteado. Si lo resuelve, justifique por qué; si no lo hace, indique en qué casos no funciona.

Solución: Hay solo dos casos que analizar:

- Si la mayoría fue de 6 a 3.
- Si la mayoría fue de 5 a 4.

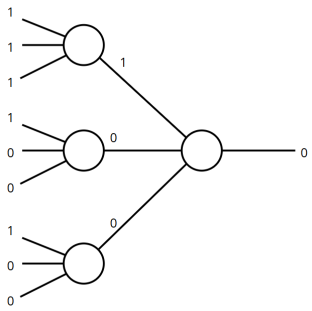
Si fue de **6 a 3**, el circuito propuesto determina correctamente el valor que más se repite. Para comprobarlo, hay que ver cómo pueden dividirse los 6 valores de mayoría entre las tres compuertas de la primera "fase" del circuito; solo hay tres posibilidades:

- 3, 3, 0
- 2, 2, 2

- 3, 2, 1

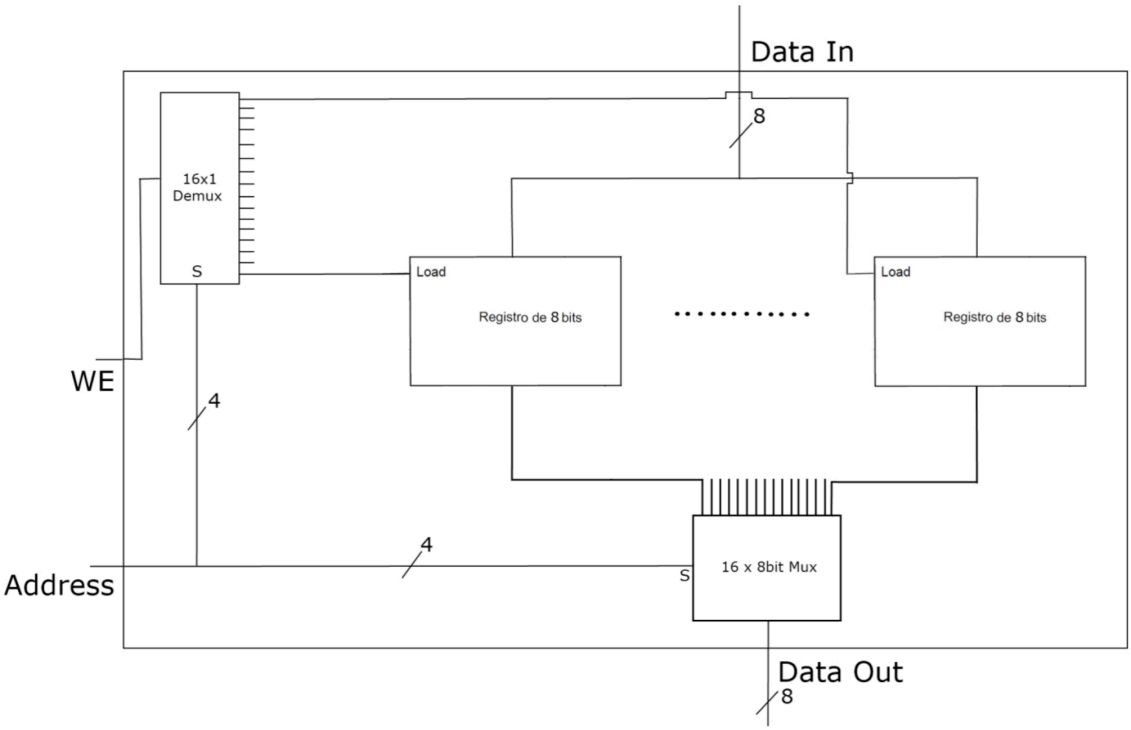
En los tres casos, la cuarta compuerta (la segunda "fase") siempre produce el resultado correcto.

Si fue de **5 a 4**, el circuito propuesto puede equivocarse. Por ejemplo: si los *inputs* se dividen en 3-0, 1-2 y 1-2, el circuito dice que la opción minoritaria, de 4 valores, es la mayoritaria.



Pregunta 3: Almacenamiento: I1 2023-2

A continuación, se adjunta un posible diagrama interno de una RAM de 16 direcciones:



Modifique este diagrama para que reciba entradas $\texttt{DataIn}_1$; $\texttt{DataIn}_2$; $\texttt{Address}_1$; $\texttt{Address}_2$; $\texttt{WE}_1$; $\texttt{WE}_2$, y salidas $\texttt{DataOut}_1$ y $\texttt{DataOut}_2$ para leer y escribir datos en dos direcciones de memoria distintas de forma simultánea.

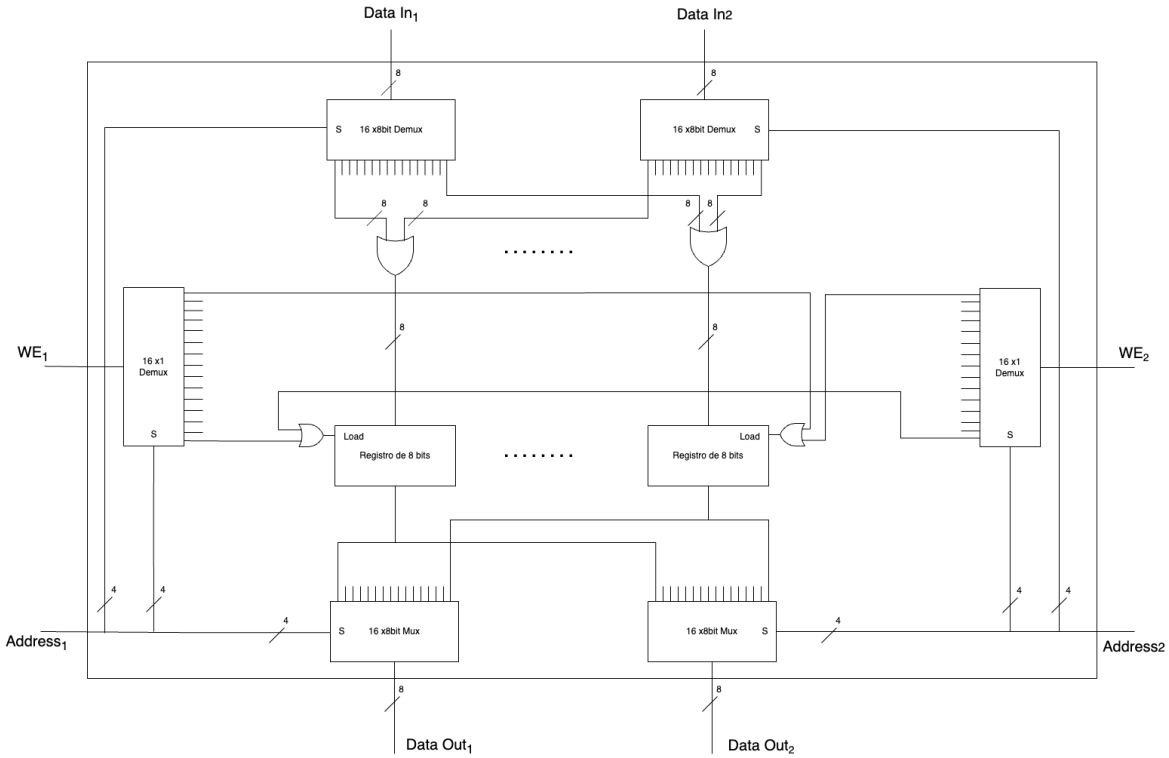
Solución: Lo que se pide, en resumen, es crear una RAM que permita el acceso simultáneo a dos direcciones de memoria distintas. Para ello, se deben realizar los siguientes cambios:

- Se agrega un segundo MUX cuya señal de selección es igual a Address2 y recibe como entradas los valores de todos los registros internos. Esto permite escoger una segunda salida DataOut2.
- Se agrega un segundo DEMUX cuya señal de selección es igual a $\texttt{Address}_2$ y recibe como entrada la señal $\texttt{WE}_2$. Esto permite transmitir solo a uno de los 16 registros la segunda señal de escritura.
- Como cada registro recibe dos señales de escritura, se conectan ambas a una compuerta `OR`. Se habilitará la escritura sobre un registro si, y solo si $\texttt{WE}_2$ está activo para su dirección $\texttt{Address}_1$ o si $\texttt{WE}_2$ está activo para su dirección $\texttt{Address}_2$.

- Para manejar las dos entradas $\texttt{DataIn}_1$ y $\texttt{DataIn}_2$, se conectan cada una de ellas a un DEMUX para transmitir su valor solo a uno de los 16 registros.

- Como cada registro recibe dos valores de entrada, se conectan ambos a una compuerta `OR` de 8 bits. Como las direcciones de memoria son **distintas**, nunca se tendrán dos valores de entrada distintos de 0. Por lo tanto, la compuerta cumple con seleccionar solo el valor de escritura transmitido si corresponde.

Las modificaciones anteriores se ven reflejadas en el siguiente diagrama:



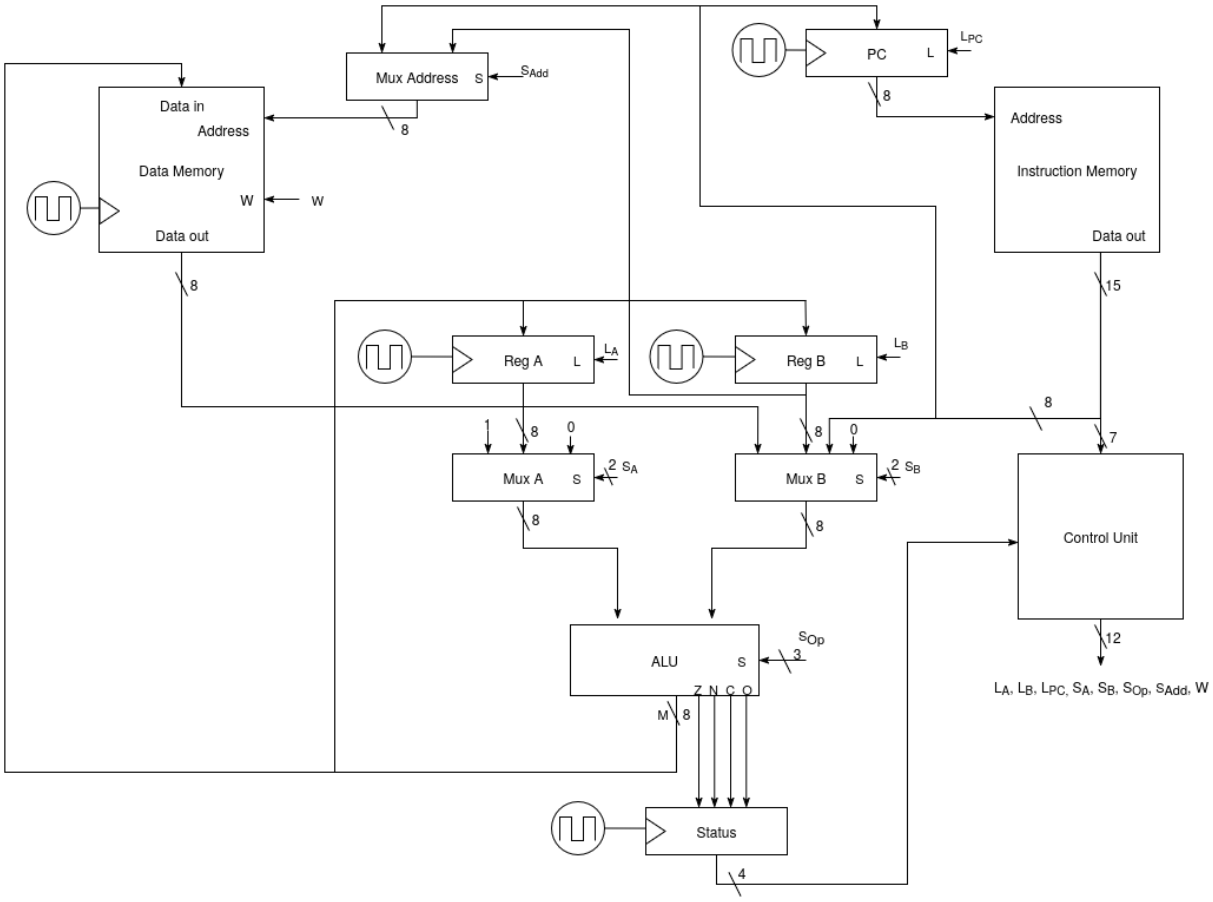
Pregunta 4: Computador Básico con saltos

(a) Modifique la arquitectura del computador básico para que el registro `STATUS` se actualice solo después de la ejecución de una instrucción `CMP`.

> **Solución:** La modificación más simple que se puede hacer para lograr el objetivo consiste en crear una nueva señal (llamémosla $\texttt{L}_{stat}$). $\texttt{L}_{stat}$ será la señal que habilita la carga de los datos en el registro `STATUS`. Finalmente, basta con que la `Control Unit` se encargue de transmitir $\texttt{L}_{stat} = 0$ para todo *opcode* que no corresponda a la instrucción `CMP`.

(b) ¿Cómo se podría implementar en el computador básico una señal para que esta indique luego de realizar una operación cuando el resultado es par o impar?

> **Solución:** Esto se puede hacer de forma muy sencilla si nos damos cuenta del siguiente hecho: Todo número par en representación binaria termina en 0 y todo número impar termina en 1. Esto, ya que todo bit representa una potencia de 2, salvo el último que representa un 1. Por lo tanto, bastaría con añadir a la salida de la `ALU` una señal P equivalente al último bit del resultado (o su negación) y que fuera almacenada dentro del registro `STATUS` de forma directa. Solo faltaría una nueva instrucción asociada a un nuevo *opcode* (que podríamos llamar JEN -*Jump Even Number*- o JON -*Jump Odd Number*-, por ejemplo).

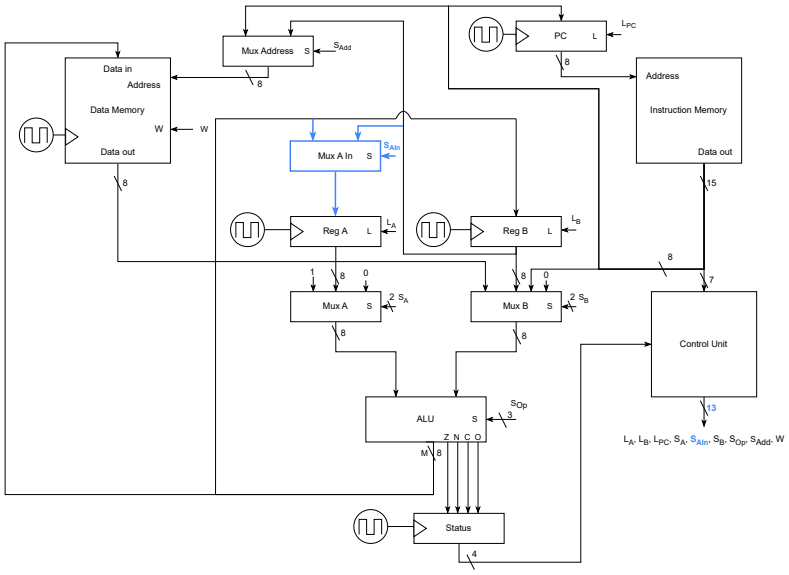

Mux Address
S
SAdd
8
PC
L
LPC
8
Data in
Address
Data Memory
W
W
Data out
Address
Instruction Memory
Data out
8
15
Reg A
L
LA
Reg B
L
LB
8
1
8
0
8
0
7
Mux A
S
2
SA
Mux B
S
2
SB
8
8
Control Unit
ALU
S
SOp
3
Z N C O
M
8
12
LA, LB, LPC, SA, SB, SOp, SAdd, W
Status
4

(c) En los ítems **(I)** y **(II)**, 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/incremento/-decremento, 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* necesarias en un solo diagrama. Además, puede implementar instrucciones sin modificar el *hardware* si es que la arquitectura se lo permite, indicando la forma de hacerlo a partir de las señales de control. **Cada instrucción debe ejecutarse en un ciclo**.

I. `SWAP A,B`. Almacena en el registro `A` el valor del registro `B`; y almacena en el registro `B` el valor del registro `A`, *i.e.* intercambia sus valores.

II. `JGTCMP A,B,label`. Salta a la dirección asociada a `label` si, y solo si las *flags* `Z` y `N` provenientes del registro `Status` son iguales a 0 y, al mismo tiempo, ejecuta la operación `A - B` en la ALU sin actualizar los registros `A` y `B`, independiente de si hay salto o no.

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.



- Se añade el multiplexor `Mux A In` y su señal de selección S_{AIn} para seleccionar el valor de entrada del registro `A`. Para la instrucción `SWAP A,B`, se seleccionará el valor del registro `B`; en cualquier otro caso, se seleccionará el valor proveniente de la ALU. Cabe destacar que se pudo haber implementado un cambio similar para el registro `B`, pero no es necesario para implementar la instrucción.
- Para la instrucción `JGTCMP A,B,label`, no se necesitan modificaciones.

A continuación, la tabla de señales resultante:

Instrucción	L_A	L_B	L_{PC}	W	S_A	S_B	S_{OP}	S_{Add}	S_{AIn}
SWAP A,B	1	1	0	0	A	ZERO	ADD	-	B
JGTCMP A,B,label	0	0	Z = 0 & N = 0	0	A	B	SUB	-	-

Por cada instrucción, se otorgan:

- **0.75 ptos.** por *hardware* correctamente modificado; **0.5 ptos.** si existe **un solo error** de implementación; **0.25 ptos.** si existen **solo dos errores**; y **0 ptos.** en otro caso.
- **0.75 ptos.** por entregar una combinación de señales correcta; **0.5 ptos.** si existe **un solo error**; **0.25 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.

(d) A continuación, se adjuntan dos fragmentos de código **(1)** y **(2)** escritos en el Assembly del computador básico del curso, con la diferencia de que **(2)** incorpora las instrucciones implementadas en los ítems anteriores. Para cada fragmento, indique los valores de los registros `A` y `B` al finalizar la ejecución del código. Si no incluye desarrollo o no justifica cómo llegó a los valores obtenidos, **no se otorgará puntaje**.

(1)

```
DATA:
  var1 -1
  var2 1
  var3 -1
  var4 1
CODE:
  MOV A,125  ; 125 = 01111101b
  NOT A,A
  SHL A,A
  SHR A,A
  MOV B,A
  MOV A,(B)
  JGT case_1
  JMP case_2
  case_1:
    MOV A,-1
    JMP end
  case_2:
    MOV A,1
  end:
```

(2)

```
DATA:
  var1 5
  var2 16
CODE:
  init:
    MOV A,(var1)
    MOV B,(var2)
    CMP A,B
    JGTCMP A,B,case_1
    JLT case_2
    case_1:
      JMP continue
    case_2:
      SWAP A,B
    continue:
      SWAP A,B
      NOT A,A
      SWAP A,B
      INC B
      ADD A,B
```

Solución:
El fragmento de código **(1)** accede, a través de direccionamiento indirecto, a una de las variables definidas en `DATA`. Si el valor leído es positivo, se guarda en el registro `A` un

-1; en otro caso, se guarda un 1. El fragmento de código **(2)**, por otra parte, computa el valor absoluto de la resta entre `var1` y `var2` y guarda el resultado en el registro `A`. A continuación, se muestra el resultado de la ejecución para ambos fragmentos:

(1)

```
DATA:
  var1 -1    ; MEM[0] = -1
  var2 1     ; MEM[1] = 1
  var3 -1    ; MEM[2] = -1
  var4 1     ; MEM[3] = 1
CODE:
  MOV A,125  ; A = 01111101b = 125
  NOT A,A    ; A = 10000010b = 130
  SHL A,A    ; A = 00000100b = 4
  SHR A,A    ; A = 00000010b = 2
  MOV B,A    ; B = 00000010b = 2
  MOV A,(B)  ; A = MEM[2] = -1
  JGT case_1 ; N = 1 -> No salta
  JMP case_2 ; Salta a case_2
  case_1:
    MOV A,-1 ; No se ejecuta
    JMP end  ; No se ejecuta
  case_2:
    MOV A,1  ; A = 1
  end:       ; A = 1, B = 2
```

(2)

```
DATA:
  var1 5               ; MEM[0] = 5
  var2 16              ; MEM[1] = 16
CODE:
  init:
    MOV A,(var1)       ; A = MEM[0] = 5
    MOV B,(var2)       ; B = MEM[1] = 16
    CMP A,B            ; A - B -> Z = 0, N = 1
    JGTCMP A,B,case_1  ; N = 1 -> No salta, A - B -> Z = 0, N =
     1
    JLT case_2         ; N = 1 -> Salta a case_2
    case_1:
      JMP continue     ; No se ejecuta
    case_2:
      SWAP A,B         ; A = 16, B = 5
    continue:
      SWAP A,B         ; A = 5, B = 16
      NOT A,A          ; A = NOT(5)
      SWAP A,B         ; A = 16, B = NOT(5)
      INC B            ; A = 16, B = NOT(5)+1 = C2(5) = -5
      ADD A,B          ; A = 11, B = -5
```

Para cada fragmento de código, 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.

Feedback ayudantía

Escanee el QR para entregar feedback sobre la ayudantía.

