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

IIC2343 - Arquitectura de Computadores (II/2025)

Interrogación 1

Pauta de evaluación

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

(a) (1.5 ptos.) Explique por qué la representación de complemento de 2 es **desbalanceada** e indique su consecuencia sobre el rango de números positivos y negativos representables.

> **Solución:** Se considera una representación desbalanceada porque se representan más números negativos que positivos. Esto ocurre porque la representación del valor 0 ocupa una de las combinaciones de bits de números positivos. Su consecuencia sobre el rango de números representable es que este siempre tendrá un número negativo sin su representación positiva. Por ejemplo, para 8 bits, el rango de números representable corresponde a $[-128, 127]$. Se otorgan **0.75 ptos.** por explicar el desbalance de la representación, y **0.75 ptos.** por indicar su consecuencia sobre el rango de números representable.

(b) (1.5 ptos.) 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.**

(c) (3 ptos.) Describa un algoritmo para transformar números naturales codificados en base 8 a base 4 **sin hacer uso de la fórmula de transformación de bases** $\sum_{k=0}^{n-1} s_k \times b^k$. Es decir, su algoritmo no puede transformar su número a base decimal como paso intermedio. Ilustre su funcionamiento transformando el número 712_8 a base 4.

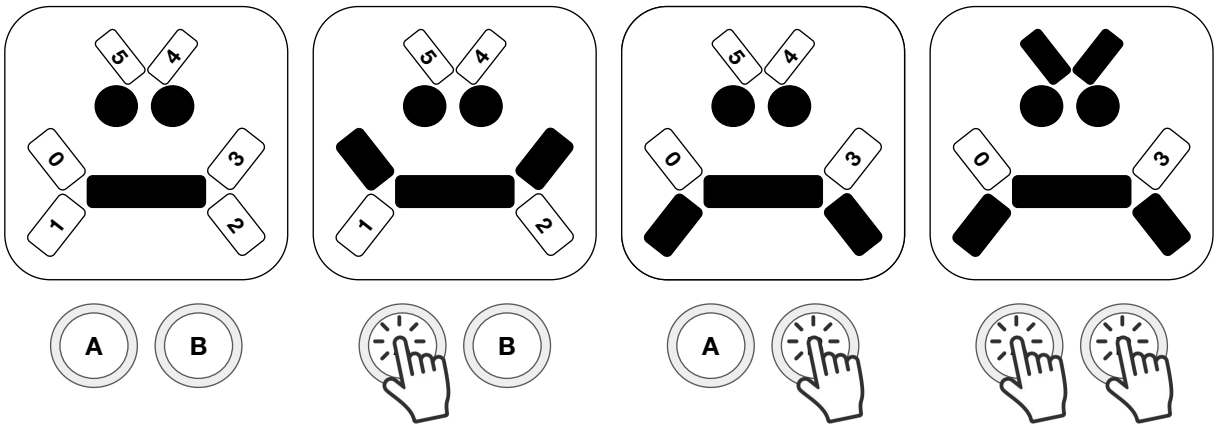
Solución: A continuación, se describe un algoritmo que cumple con lo solicitado:

1. Transformar cada dígito del número en base 8 en tres bits que lo representen, manteniendo el orden correlativo entre ellos. Por ejemplo, $712_8 = 111|001|010\text{b}$.
2. Si el total de bits no es múltiplo de 2, agregar un 0 a la izquierda. Por ejemplo, $111001010\text{b} = 0111001010\text{b}$
3. Agrupar los bits en grupos de 2 y transformar cada grupo a su número equivalente en base 4, manteniendo el orden correlativo entre grupos. Por ejemplo, $01|11|00|10|10\text{b} = 1_4|3_4|0_4|2_4|2_4 = 13022_4$.

Este algoritmo funciona ya que tanto la base 8 como la base 4 son **potencias de 2**, por lo que podemos definir un procedimiento similar al de la transformación de base hexadecimal a binaria. Se otorgan **2.5 ptos.** por describir un algoritmo que **funcione correctamente** y **0.5 ptos.** por ilustrarlo. Si describen un algoritmo similar al de la pauta sin el paso 2, solo se descuentan **0.5 ptos.**

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

Su docente, con el fin de comunicar sus emociones sin interrumpir la clase, decide usar un *display* de 6 segmentos para representar su estado de ánimo en torno a la participación del curso:



Como se observa en la figura, este funciona a partir de dos botones A y B: Si presiona solo el botón A, el *display* mostrará que el docente **está contento**; si presiona solo el botón B, el *display* mostrará que el docente **está triste**; y si presiona A y B de forma simultánea, esto señalará que el docente **está molesto**.

A partir de este contexto:

(a) (3 ptos.) Diseñe, para cada segmento S_i del *display*, un circuito con dos señales de entrada A y B de 1 bit, y una señal de salida de 1 bit. Las señales de entrada indican si los botones A y B están presionados (1) o no (0), mientras que la salida indica si el segmento S_i se prende (1) o no (0) según los botones presionados a partir de lo ilustrado en la figura. Puede, si lo desea, diseñar un único circuito para computar la salida de todos los segmentos simultáneamente.

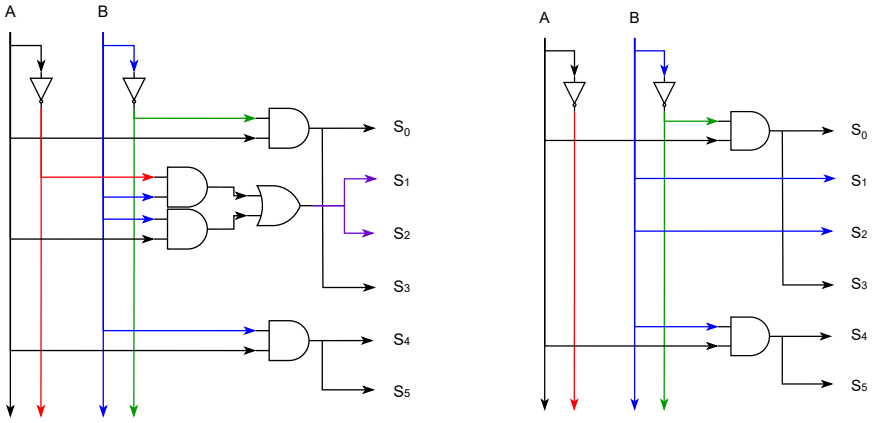
Solución: A continuación, la tabla de verdad para los segmentos S_i a partir de A y B:

A	B	S_0	S_1	S_2	S_3	S_4	S_5
0	0	0	0	0	0	0	0
0	1	0	1	1	0	0	0
1	0	1	0	0	1	0	0
1	1	0	1	1	0	1	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.

- $S_0 = S_3 = A \wedge \overline{B}$
- $S_1 = S_2 = (\overline{A} \wedge B) \vee (A \wedge B) = (\overline{A} \vee A) \wedge B = B$
- $S_4 = S_5 = A \wedge B$

Finalmente, se puede diseñar un circuito a partir de estas expresiones. Se incluye un circuito para las primeras expresiones obtenidas con *minterms* y *maxterms* (izquierda), y otro reducido a partir de equivalencias lógicas (derecha).



Es importante notar que se evaluará la correctitud del circuito diseñado, no que sea igual al de esta pauta. Se otorgan **0.5 ptos.** 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.) Diseñe, utilizando **solo** compuertas lógicas, *enablers*, multiplexores y flip-flops, un contador secuencial de 2 bits que incrementa o decrementa en cada flanco de subida a partir de las señales A y B antes descritas bajo las siguientes condiciones:

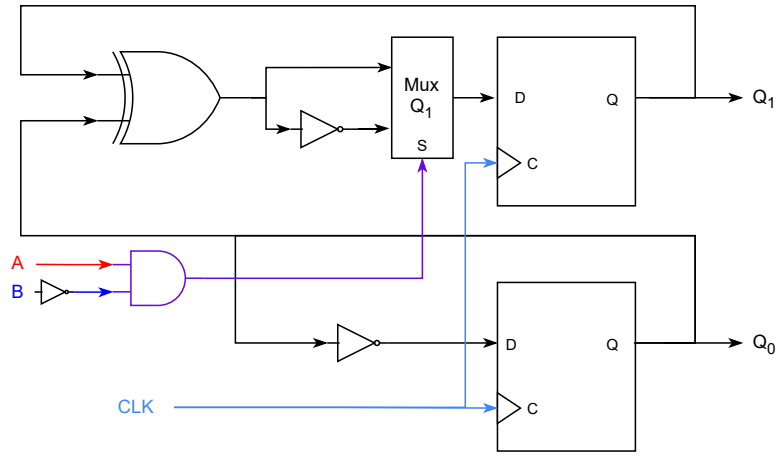
- Incrementa en una unidad si el docente está contento con la participación.
- Decrementa en una unidad si el docente está triste **o** molesto con la participación.
- Mantiene su valor en otro caso.

Puede asumir que su contador hace *overflow* en caso de incrementar estando en su máximo valor o decrementar estando en su mínimo valor.

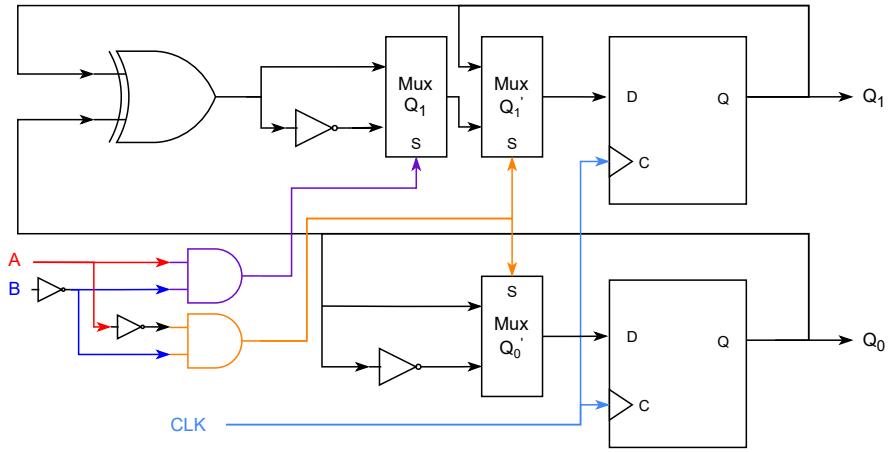
Solución: Si denotamos por $Q_1^t Q_0^t$ el estado del contador en un ciclo t de la señal de control `CLK`; $(Q_1^{t+1}Q_0^{t+1})^+$ su estado en el próximo ciclo si incrementa; y $(Q_1^{t+1}Q_0^{t+1})^-$ si decrementa, se puede elaborar la siguiente tabla de verdad:

Q_1^t	Q_0^t	$(Q_1^{t+1})^+$	$(Q_0^{t+1})^+$	$(Q_1^{t+1})^-$	$(Q_0^{t+1})^-$
0	0	0	1	1	1
0	1	1	0	0	0
1	0	1	1	0	1
1	1	0	0	1	0

Se deduce que $(Q_0^{t+1})^+ = (Q_0^{t+1})^- = \overline{Q_0^t}$, $(Q_1^{t+1})^+ = Q_0^t \oplus Q_1^t$ y $(Q_1^{t+1})^- = \overline{(Q_0^t \oplus Q_1^t)}$. Es decir, Q_0 será igual a `NOT` Q_0 independiente de que el contador incremente o decremente, mientras que Q_1 será igual a Q_0 `XOR` Q_1 en caso de incrementar y Q_0 `XNOR` Q_1 en caso de decrementar. Haciendo uso de flip-flops, multiplexores y las compuertas antes señaladas, se llega al siguiente diagrama:



Se añade el multiplexor `Mux Q1` para seleccionar el valor final de Q_1 dependiendo de si el docente está contento ($A = 1, B = 0$) o no. Esta elección es arbitraria y se pudo haber realizado la selección con la condición de decremento. Ahora, esta es una solución intermedia porque no contempla el caso en el que $A = 0$ y $B = 0$, donde se debe mantener el estado del contador y no decrementarlo como lo haría la implementación actual. Para ello, agregaremos un multiplexor para cada señal Q_i que seleccionará su valor sin modificar para la combinación de A y B antes señalada:



Se añaden los multiplexores `Mux Q0'` y `Mux Q1'` para seleccionar los valores originales de Q_0 y Q_1 respectivamente si se cumple que $A = 0$ y $B = 0$. En otro caso, se almacenará el valor actualizado dependiendo de si el contador incrementa o decrementa.

Es importante notar que se evaluará la correctitud del contador diseñado, no que sea igual al de esta pauta. Se otorga **1 pto.** por diseñar correctamente un contador de 2 bits incremental, **1 pto.** por diseñar correctamente un contador de 2 bits decremental, y **1 pto.** por actualizar su valor correctamente a partir de las señales A y B. Por cada criterio, se otorga la mitad del puntaje si existe, como máximo, un error de implementación.

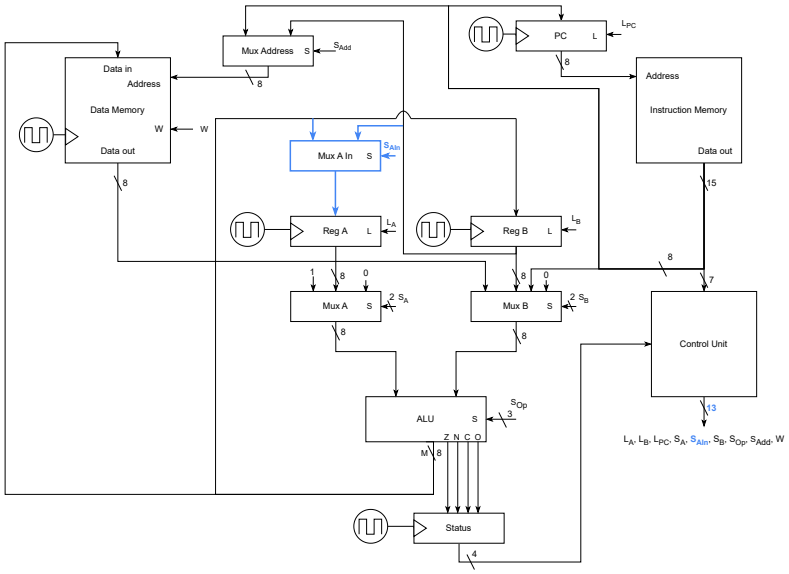
Pregunta 3: Computador básico con saltos (6 ptos.)

En los ítems **(a)** y **(b)**, 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**.

(a) (1.5 ptos.) `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.

(b) (1.5 ptos.) `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.

(c) (3 ptos.) 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.