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

IIC2343 - Arquitectura de Computadores (I/2025)

Interrogación 1

Pauta de evaluación

Pregunta 1: Representaciones numéricas (6 ptos.)

(a) (1 pto.) Suponga que se suman dos números de 8 bits con signo, A y B, ambos negativos. Como resultado, se genera un acarreo de salida igual a 1. ¿Implica esto necesariamente la ocurrencia de un *overflow*? Justifique su respuesta.

Solución: No. El *overflow* ocurre cuando el resultado de una operación entre dos números está fuera del rango de su representación. Al usar una representación **con signo**, el acarreo de salida igual a 1 no implica que exista *overflow*. Esto se puede evidenciar con un contraejemplo en una representación de números enteros con 4 bits:

$$-1_2 + -2_2 = 1111 + 1110 = (1)1101 = -3_2$$

Si bien existe un acarreo de salida, no hay *overflow*. Un acarreo de salida solo evidencia *overflow* en una representación **sin signo**. Se otorgan **0.5 ptos.** por responder correctamente; y **0.5 ptos.** por justificación.

(b) (2 ptos.) 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 $C_4(x)$.

(c) (2 ptos.) En los siguientes incisos, se le entregará un par de números reales A, B en formato float del estándar IEEE754 con notación hexadecimal. En cada caso, debe interpretar sus valores y realizar las operaciones de suma o multiplicación solicitadas. Luego, debe indicar tanto el valor teórico (resultado de la operación) como el valor real (almacenado como float del estándar IEEE754) de la operación. Si son iguales, explicite que lo son. Los valores deben presentarse en base binaria con notación científica normalizada.

- (1 pto.) $A + B$. $A =$ 0x4C800000, $B =$ 0x40400000.

Solución:

- $A =$ 01001100100000000000000000000000b.
  - **Signo:** 0b $\rightarrow$ positivo.
  - **Exponente:** 10011001b $\rightarrow$ 153 - 127 = 26.
  - **Significante:** 00000000000000000000000b $\rightarrow$ 1,0b.
  - **Valor:** $(1,0 \times 10^{26})$b = 10000000000000000000000000b = 2^{26}.
- $B =$ 0100000001000000 0000000000000000b.
  - **Signo:** 0b $\rightarrow$ positivo.
  - **Exponente:** 10000000b $\rightarrow$ 128 - 127 = 1.
  - **Significante:** 10000000000000000000000b $\rightarrow$ 1,1b.
  - **Valor:** $(1,1 \times 10^{1})$b = 11b = 3.

De lo anterior, se deduce que la operación a realizar es $2^{26} + 3$. En este caso, no se evaluará la forma en la que se realice la suma; solo que su resultado sea correcto. Aquí se opta por seguir el algoritmo visto en clases: se igualan los exponentes y B se expresa como $0,00000000000000000000000011 \times 10^{26}$. Luego, sumando los significantes, se tiene $A + B = (1,00000000000000000000000011 \times 10^{26})$b. Finalmente, si se almacena como `float`, se tiene el valor 01001100100000000000000000000000b = 0x4C800000 = 2^{26}. **Se pierden los 3 bits menos significantes que aportaban la magnitud de 3 y, por ende, se pierde precisión**.

Se otorgan **0.25 ptos.** por la interpretación correcta de A; **0.25 ptos.** por la interpretación correcta de B; **0.25 ptos.** por señalar el resultado correcto de la operación; y **0.25 ptos.** por indicar correctamente si se pierde precisión o no. Se acepta escribir el valor en potencias de 2.

- (1 pto.) $A \times B$. A = 0x42F60000, B = 0x3F800000.

Solución:

- A = 01000010111101100000000000000000b.
  - **Signo:** 0b → positivo.
  - **Exponente:** 10000101b → 133 - 127 = 6.
  - **Significante:** 11101100000000000000000b → 1,111011b.
  - **Valor:** $(1,111011 \times 10^6)$b = 1111011b = 123.
- B = 00111111100000000000000000000000.
  - **Signo:** 0b → positivo.
  - **Exponente:** 01111111b → 127 - 127 = 0.
  - **Significante:** 00000000000000000000000b → 1,0b.
  - **Valor:** $(1,0 \times 10^0)$b = 1b = 1.

De lo anterior, se deduce que la operación a realizar es 123×1. En este caso, no se evaluará la forma en la que se realice la multiplicación; solo que su resultado sea correcto. Aquí se opta por seguir el algoritmo visto en clases: se suman los exponentes sin desfase y se llega al valor $6+0 = 6 = 110$b. Luego, se multiplican los significantes: $1,111011 \times 1,0 = (1111011 \times 10^{-1}) \times (1 \times 10^0) = (1111011 \times 1) \times 10^{-1} = 1111011 \times 10^{-1} = 1,111011$. Finalmente, se tiene como resultado $1,111011 \times 10^{110}$b = 1111011b = 123, que en formato float se almacena como 01000010111101100000000000000000b = 0x42F60000. **No existe pérdida de precisión**.

Se otorgan **0.25 ptos.** por la interpretación correcta de A; **0.25 ptos.** por la interpretación correcta de B; **0.25 ptos.** por señalar el resultado correcto de la operación; y **0.25 ptos.** por indicar correctamente si se pierde precisión o no. Se acepta escribir el valor en potencias de 2. Además, en este caso se acepta no realizar la operación si se deduce que la multiplicación es por 1, manteniendo el valor final de A.

(d) (1 pto.) ¿Cuál de los siguientes tipos de datos puede representar una mayor cantidad de valores **distintos** en base binaria: un número entero de 32 bits o un número en formato `float` de 32 bits según el estándar IEEE-754? Justifique su respuesta.

Solución: Se pueden representar una cantidad mayor de valores distintos en **una representación de números enteros de 32 bits**. El motivo es sencillo: En esta representación, cada secuencia de bits tendrá un valor distinto, por lo que se tienen 2^{32} valores. En el estándar IEEE-754, en cambio, existen combinaciones de bits que **representan el mismo valor**, en particular para `NaN`. Por ende, existirán menos de 2^{32} valores distintos. Se otorgan **0.5 ptos.** por indicar correctamente la representación con mayor cantidad de valores distintos; y **0.5 ptos.** por justificación. No es necesario indicar cuántos valores menos presenta la representación `float`.

Pregunta 2: Circuitos digitales y Almacenamiento de datos (6 ptos.)

El Museo Nacional ha instalado un sistema de seguridad que utiliza sensores láser distribuidos a lo largo de un pasillo. Existen cuatro sensores, denominados L_1, L_2, L_3 y L_4, los que entregan una señal binaria de acuerdo al siguiente criterio:

- $L_i = 1$ si el haz del sensor i fue interrumpido.
- $L_i = 0$ si el haz permanece intacto.

Un estudio de seguridad reveló que una persona sospechosa, potencialmente un ladrón, interrumpe **exactamente 3** de los 4 sensores al desplazarse por el pasillo. En cambio, un visitante común puede llegar a interrumpir 1, 2 o los 4, pero **nunca exactamente 3**. Por lo tanto, se desea diseñar una alarma mediante un circuito combinacional que emita una señal binaria de salida A, tal que:

$$A = 1 \quad \text{si, y solo si, exactamente 3 de los sensores fueron interrumpidos.}$$

(a) (3 ptos.) Construya la tabla de verdad correspondiente a la salida A y, a partir de ella, diseñe un circuito que la calcule utilizando **únicamente** compuertas lógicas básicas: `AND`, `OR`, `XOR` y `NOT`.

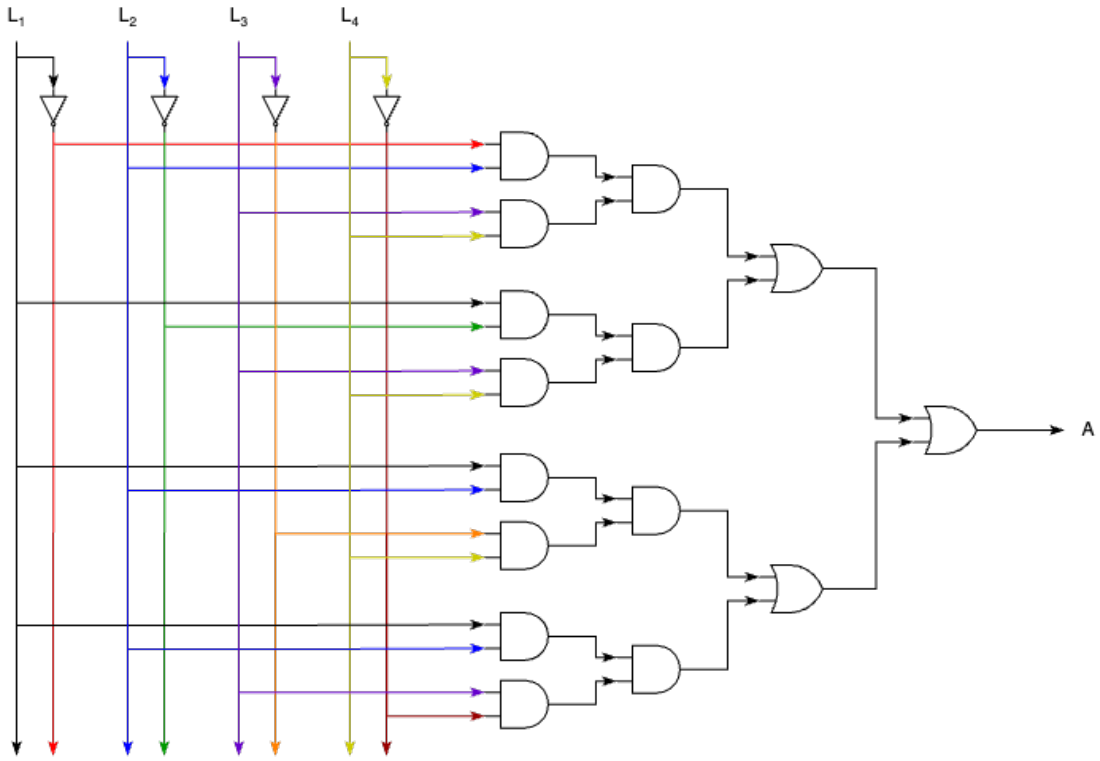
Solución: A continuación, la tabla de verdad para A a partir de las señales L_i:

L_1	L_2	L_3	L_4	A
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0

A partir de ella, se puede hacer uso de *minterms* o *maxterms* para obtener la expresión que representa la salida. En este caso, el uso de *minterms* asegura una expresión concisa:

$$\begin{aligned} A = \quad & (\overline{L_1} \wedge L_2 \wedge L_3 \wedge L_4) \\ & \vee (L_1 \wedge \overline{L_2} \wedge L_3 \wedge L_4) \\ & \vee (L_1 \wedge L_2 \wedge \overline{L_3} \wedge L_4) \\ & \vee (L_1 \wedge L_2 \wedge L_3 \wedge \overline{L_4}) \end{aligned}$$

Finalmente, se puede diseñar un circuito a partir de esta expresión.



Es importante notar que se evaluará la correctitud del circuito diseñado, no que sea exactamente igual al de esta pauta. Además, se permite el uso de compuertas `AND` y `OR` con más de dos entradas.

Se otorgan **0.6 ptos.** por el circuito correctamente diseñado, y **0.15 ptos.** por cada combinación de señales de entrada que compute el valor correcto de A. Se otorgan **0.3 ptos.** si el circuito presenta, como máximo, un error de diseño. No hay puntaje parcial para una combinación de señales de entrada que compute incorrectamente el valor de A.

(b) (3 ptos.) Adicionalmente, el museo desea que este sistema se conecte a un contador que registre la cantidad de veces que se ha activado la alarma. Como los administradores son optimistas, suponen que **no se registrarán más de 7 activaciones** en el año, por lo que ese será el valor máximo requerido. Diseñe este contador secuencial utilizando flip-flops tipo D y compuertas lógicas, de forma que se incremente en una unidad con cada flanco de subida **si, y solo si,** A **es igual a 1**. En su diseño se permite el uso de multiplexores, en caso de que los necesite.

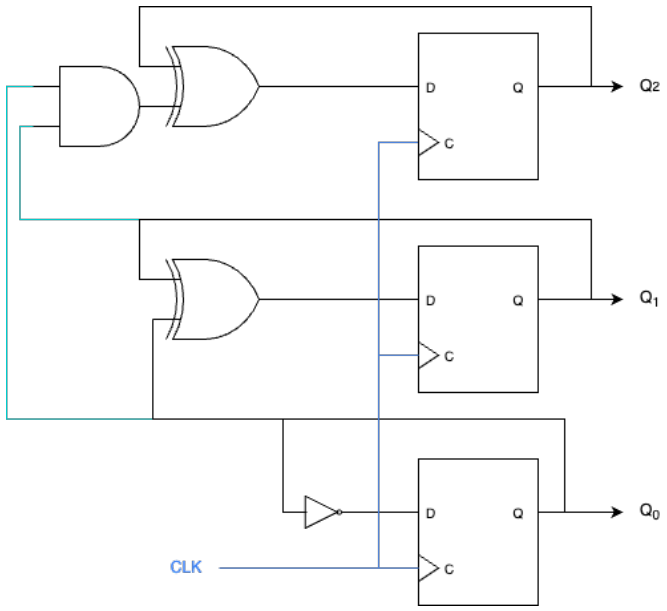
Solución: Si el contador tiene un valor máximo igual a 7, debe ser de 3 bits. Denotaremos sus salidas como $Q_2Q_1Q_0$; $Q_2^tQ_1^tQ_0^t$ como su estado en el ciclo t; y $Q_2^{t+1}Q_1^{t+1}Q_0^{t+1}$ como su estado en el siguiente ciclo. Así, se tiene la siguiente tabla de verdad:

Q_2^t	Q_1^t	Q_0^t	Q_2^{t+1}	Q_1^{t+1}	Q_0^{t+1}
0	0	0	0	0	1
0	0	1	0	1	0
0	1	0	0	1	1
0	1	1	1	0	0
1	0	0	1	0	1
1	0	1	1	1	0
1	1	0	1	1	1
1	1	1	0	0	0

Si se revisa la tabla de verdad de cada salida Q_i^{t+1}, se puede determinar que:

$$Q_0^{t+1} = \overline{Q_0^t} \qquad Q_1^{t+1} = Q_0^t \veebar Q_1^t \qquad Q_2^{t+1} = (Q_0^t \wedge Q_1^t) \veebar Q_2^t$$

Por lo tanto, se diseña el circuito a partir de las expresiones anteriores:

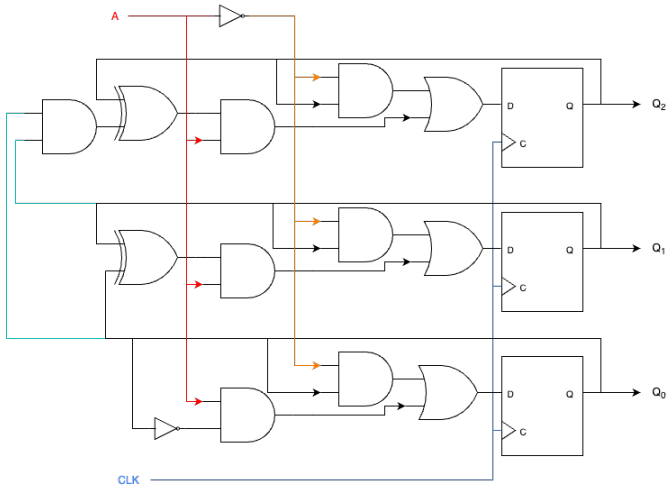


Ahora, lo anterior corresponde a un circuito secuencial que incrementa con cada flanco de subida. Se requiere, como paso final, que este se realice solo si la señal de entrada A es igual a 1. Entonces, se deben modificar las salidas de la siguiente forma:

$$Q_0^{t+1} = (A \wedge \overline{Q_0^t}) \vee (\overline{A} \wedge Q_0^t) \quad Q_1^{t+1} = (A \wedge (Q_0^t \veebar Q_1^t)) \vee (\overline{A} \wedge Q_1^t)$$

$$Q_2^{t+1} = (A \wedge ((Q_0^t \wedge Q_1^t) \veebar Q_2^t)) \vee (\overline{A} \wedge Q_2^t)$$

En palabras simples: si A es igual a 1, la salida será igual al incremento en una unidad del contador; en otro caso, mantendrá su estado actual. El circuito final, entonces, es el siguiente:



Esto es equivalente a **la implementación de un multiplexor** para cada salida Q_i^{t+1}, donde la señal A se usa para seleccionar entre el valor incrementado o el actual.

Otras implementaciones que se consideran igual de correctas son las siguientes:

- Uso de la señal A como *enabler* de `CLK` (*i.e.* $A \wedge$ `CLK`), evitando los flancos de subida de los flip-flops si no hay alarma.
- Mantención del estado $Q_2Q_1Q_0 = 111$ una vez que se llega al valor máximo, en vez de dejar que por *overflow* vuelva a ser 000.

Se otorgan **0.5 ptos.** por cada salida correctamente computada en un flanco de subida; y **0.5 ptos.** si, además, se actualiza correctamente según el valor de la señal A. Se considera válido el uso de multiplexores en el circuito. El uso de sumadores, en cambio, **se considera válido si, y solo si se incluyen los diagramas internos de los elementos que lo componen** (*half-adder*, *full-adder* y sus conexiones dentro del sumador). Si se ocupa como caja negra, se otorga como máximo la mitad del puntaje si la implementación es correcta.

Pregunta 3: Computador básico (6 ptos.)

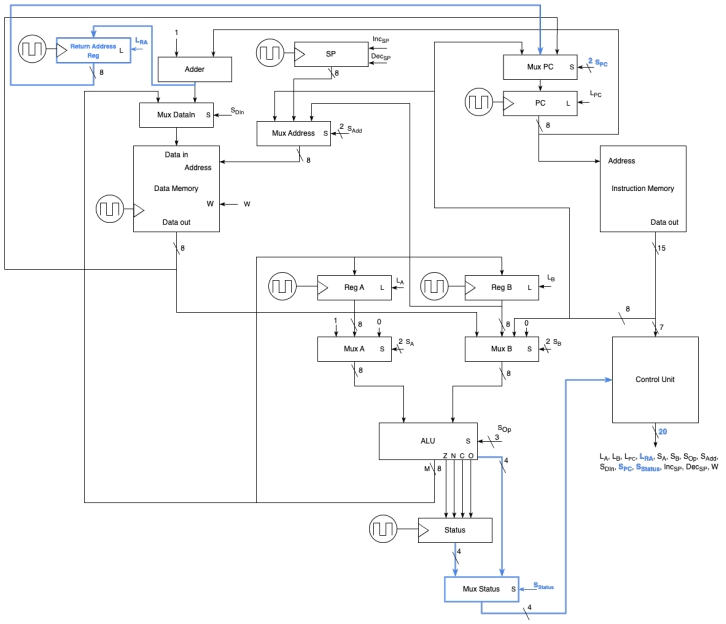
Ha sido contratado/a como programador/a en una bolsa de valores internacional. Al revisar la infraestructura tecnológica, nota que todos los procesadores se basan en el computador básico del curso, lo que limita el rendimiento necesario para reaccionar ante cambios del mercado. Su misión es diseñar una versión mejorada, llamada *FastBasicComputer*, que permita ejecutar programas con mayor velocidad y eficiencia. Para ello, deberá incorporar **instrucciones nuevas que disminuyan los ciclos de ejecución**. Deberá modificar la microarquitectura del computador básico para implementar las instrucciones nuevas y su funcionamiento. Para cada instrucción nueva, deberá incluir la combinación **completa** de señales que la ejecutan. Por cada señal de carga/escritura/incremento/decremento, deberá indicar si se activan (1) o no (0); en las señales de selección, deberá indicar el **nombre** de la entrada escogida ("-" si no afecta). Puede realizar todas las modificaciones en un solo diagrama.

(a) (1 pto.) Implemente la instrucción `CALLFAST label`: Almacena `PC+1` en un **nuevo** registro llamado *Return Address Register* y **no modifica** el *stack*. Además, salta a la dirección `label`.

(b) (1 pto.) Implemente la instrucción `RETFAST`: Carga en el contador `PC` el valor almacenado en el registro *Return Address Register* y **no modifica** el *stack*.

(c) (1 pto.) Implemente la instrucción `JEQFAST A, Lit, label`: Ejecuta la operación `A - Lit` en la ALU y salta a la dirección asociada a `label` si, y solo si la *flag* `Z` **computada en el mismo ciclo** es igual a 1.

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 registro *Return Address Register* con su señal de carga L_{RA}.
- La conexión entre la salida del *Adder* que computa `PC + 1` y la entrada de datos de *Return Address Register* permite cargar el valor en el registro con la señal L_{RA}.
- La conexión entre la salida de *Return Address Register* y una de las entradas del multiplexor `Mux PC` permite seleccionar el valor del registro como dato a cargar en el contador `PC`. Es importante destacar que, al tener tres valores a seleccionar en el multiplexor, es necesario incrementar a 2 bits la señal de selección S_{PC}.
- Se añade el multiplexor `Mux Status` y su señal de selección S_{Status} para seleccionar las *flags* a ser evaluadas por la Unidad de Control. Si provienen del registro `Status`, entonces se utilizan las *flags* computadas en el ciclo anterior; en otro caso, se utilizan las computadas en **el ciclo actual**.

A continuación, la tabla de señales resultante:

Instrucción	L_A	L_B	L_{PC}	L_{RA}	Inc_{SP}	Dec_{SP}	W	S_A	S_B	S_{OP}	S_{Add}	S_{DIn}	S_{PC}	S_{Status}
`CALLFAST label`	0	0	1	1	0	0	0	-	-	-	-	-	LIT	-
`RETFAST`	0	0	1	0	0	0	0	-	-	-	-	-	RA	-
`JEQFAST A, Lit, label`	0	0	`Z == 1`	0	0	0	0	A	LIT	SUB	-	-	LIT	ALU

Algunas consideraciones importantes:

- En la columna S_{PC}, `RA` representa la selección de la salida de *Return Address Register*.
- En la columna S_{Status}, `ALU` representa la selección de las *flags* proveniente de ella y no del registro `Status`.
- En la columna L_{PC}, no importa la notación, pero sí debe quedar claro que su valor no es por defecto 1, sino que depende de la *flag* `Z`.
- En la instrucción `JEQFAST A, Lit, label` existe una limitante importante: se debe usar el mismo literal tanto para la comparación como para el *label* dado que en la memoria de instrucciones solo se define uno. A modo de simplificación, se considerará correcto "asumir" que pueden ser distintos sin modificaciones adicionales de *hardware*, pero en la práctica su implementación requiere de la definición de dos literales por instrucción, lo que implicaría más cambios en la microarquitectura.

Por cada instrucción, se otorgan:

- **0.5 ptos.** por *hardware* correctamente modificado; **0.25 ptos.** si existe **un solo error** de implementación; y **0 ptos.** en otro caso.
- **0.5 ptos.** por por entregar una combinación de señales correcta; **0.3 ptos.** si existe **un solo error**; **0.2 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 bien no se indica explícitamente en el enunciado, sí se indicó en la hoja de respuestas entregada (además de alinearse con el contexto entregado).

(d) (2 ptos.) Para evaluar la velocidad de *FastBasicComputer*, se realiza una comparativa entre un programa escrito en el *Assembly* del computador básico `(a)` y otro equivalente en la nueva arquitectura `(b)`. Indique los valores de los registros `A` y `B` al finalizar la ejecución del código en cada caso junto con la cantidad de ciclos de ejecución. Asuma que no se agregan instrucciones para inicializar la memoria RAM.

(a)

```
DATA:
  n         8
  is_even   0
CODE:
    MOV A, (n)
    CALL check_is_even
    JMP end
    check_is_even:
        MOV B, 1
        AND A, B
        CMP A, 0
        JEQ set_is_even
        end_check_is_even:
            RET
        set_is_even:
            MOV (is_even), B
            JMP end_check_is_even
    end:
        MOV A, (is_even)
```

(b)

```
DATA:
  n         8
  is_even   0
CODE:
    MOV A, (n)
    CALLFAST check_is_even
    JMP end
    check_is_even:
        MOV B, 1
        AND A, B
        JEQFAST A, 0, set_is_even
        end_check_is_even:
            RETFAST
        set_is_even:
            MOV (is_even), B
            JMP end_check_is_even

    end:
        MOV A, (is_even)
```

Solución: Los programas `(a)` y `(b)` determinan si la variable `n` es par (`is_even` = 1) o no (`is_even` = 0) a partir de la revisión de su bit menos significativo (`AND n, 1`). En ambos casos, el valor final de `A` será igual a 1, ya que es correspondiente al resultado de `is_even` para `n` igual a 8; mientras que el valor de `B` también será 1, dado que se le asigna este literal y no se modifica de nuevo. En términos de ejecución, el programa `(a)` toma un total de 12 ciclos ya que se debe considerar que se realiza el salto condicional `JEQ set_is_even` y que `RET` ejecuta en dos ciclos; `(b)`, en cambio, toma 10 ciclos al no requerir el uso de `CMP` y al tener un retorno de un solo ciclo con `RETFAST`.

Para cada programa, se otorgan **0.25 ptos.** por indicar el valor correcto de `A`; **0.25 ptos.** por indicar el valor correcto de `B`; y **0.5 ptos.** por indicar la cantidad correcta de ciclos de ejecución. Excepcionalmente, se otorgarán **0.25 ptos.** por programa si se evidencia un desarrollo parcialmente correcto a pesar de no llegar a las respuestas esperadas.

(e) (1 pto.) Las instrucciones `CALLFAST label` y `RETFAST` presentan un problema al ejecutar llamados anidados, situación que **no ocurre** con `CALL` y `RET`. En particular, se observa que no se realiza de forma correcta el retorno. Explique por qué ocurre esto.

Solución: Si se tienen llamados anidados, existirá más de una dirección de retorno (una por llamado). Al usar `CALL` y `RET`, las direcciones se respaldan en el *stack*, que simplemente incrementa su tamaño. Al usar `CALLFAST` y `RETFAST`, en cambio, se tiene un único registro que se **sobreescribe con cada llamado**, manteniendo así solo la última dirección de retorno. Se otorgan **0.5 ptos.** por explicar por qué fallan las llamadas anidadas con `CALLFAST` y `RETFAST`; y **0.5 ptos.** por explicar por qué funcionan con `CALL` y `RET`.