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

IIC2343 - Arquitectura de Computadores (II/2024)

Interrogación 1

Pauta de evaluación

Pregunta 1: Representación de Números (6 ptos.)

(a) (1 pto.) Escriba la representación en decimal del número 0xCAFE, si este se interpreta como binario en complemento de dos.

Solución: En primer lugar, se representará el número en binario a través de la concatenación de la representación binaria de cada dígito hexadecimal:

- 0xC = 1100b
- 0xA = 1010b
- 0xF = 1111b
- 0xE = 1110b
- 0xCAFE = 1100101011111110b

Si el número resultante se interpreta como binario con signo, y el bit más significativo es igual a 1, su valor es negativo y se requiere la aplicación del complemento de 2 para obtener su magnitud:

$$\begin{aligned} 1100101011111110b &= -C_2(1100101011111110b) \\ &= -0011010100000010b \\ &= -(2^{13} + 2^{12} + 2^{10} + 2^8 + 2^1) \\ &= -13570 \end{aligned}$$

Se otorgan **0.5 ptos.** por determinar correctamente el signo del número, y **0.5 ptos.** por determinar correctamente su magnitud. Se considera válido expresar la magnitud como la suma de potencias de 2.

(b) (1 pto.) ¿Cuál es el problema del *shift right* lógico con los números binarios en complemento de dos? ¿Es posible solucionarlo?

Solución: Como todos los bits del número se mueven una posición hacia la derecha, esto incluye al "bit de signo" que surge naturalmente del complemento de dos. Luego, como se inserta un bit igual a 0 en la posición más significativa, cambia el signo del número durante la operación. Es posible solucionar este problema mediante el uso del *shift right* aritmético: se mueven todos los bits una posición hacia la derecha, pero en vez de insertar un 0 en la posición más significativa, se inserta el bit más significativo del número original. A continuación, un ejemplo:

- $1010 \rightarrow^{\texttt{SHR}} 0101$
- $1010 \rightarrow^{\texttt{SHR-Ar}} 1101$

Esto permite que se mantenga el signo y, mejor aún, que la operación se siga comportando como una división entera por dos sobre el número original. Se otorgan **0.5 ptos.** por describir correctamente el problema, y **0.5 ptos.** por indicar de forma correcta la solución.

(c) (2 ptos.) Al utilizar complemento de 2 para representaciones posicionales binarias, se genera una representación no equilibrada donde existen más elementos negativos que positivos. Modifique el algoritmo de transformación tal que ahora existan más números positivos que negativos.

Solución: Una modificación posible consiste en añadir un paso previo al algoritmo de complemento de 2, como se muestra a continuación:

- "Anteponer" un 1 al número (*i.e.* cambiar su bit más significativo a 1).
- Negar todos los bits.
- Sumar 1.

Con este cambio, se tendrá una mayor cantidad de números positivos que de negativos, además de respetar el resultado de la suma entre un número y su inverso aditivo. Se otorga **1 pto.** por proponer una modificación del algoritmo, y **1 pto.** si la propuesta es correcta, *i.e.* genera más positivos que negativos.

(d) (2 ptos.) Suponga que se hace uso del computador básico del curso para realizar las siguientes operaciones:

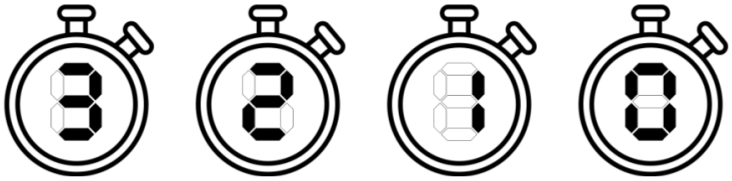
1. Multiplicación, a través de sumas iterativas, de los valores de los registros A y B, con $A = 30$ y $B = 10$, almacenando el resultado en A.
2. Aplicación de la operación *shift right* sobre el registro A, almacenando el resultado en A.

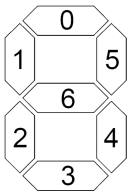
¿Cuál es el resultado, en representación decimal, del registro A luego de ejecutar las operaciones anteriores? Responda considerando que los registros almacenan valores de 8 bits sin signo.

Solución: Considerando que los registros almacenan valores de 8 bits sin signo, el valor máximo que pueden almacenar es de 255 antes de que se genere *overflow*. Por lo tanto, dado que $30 \times 10 = 300$, se tendrá como resultado $300 - 256 = 44$ en A. Luego, si se aplica un *shift right* sobre este valor, resultará en $44//2 = 22$. Se otorga **1 pto.** por indicar correctamente el resultado después de la multiplicación, y **1 pto.** por indicarlo correctamente para el *shift right*.

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

Pronto se llevarán a cabo las DCCarreras de sacos dieciocheras y, para ahorrar recursos, le piden a usted que diseñe un *timer* para marcar el inicio de cada carrera. A continuación, una imagen de referencia del *timer* esperado y del *display* de 7 segmentos que tendrá integrado para desplegar el número:





Para construir el *timer*:

(a) (3 ptos.) Diseñe, para cada segmento S_i del *display*, un circuito con una señal de entrada de 2 bits $I_1 I_0$ y una señal de salida de 1 bit. La entrada corresponderá al número a desplegar en el *timer* (3, 2, 1 o 0), mientras que la salida indica si el segmento S_i se prende (1) o no (0) según el número que se busca desplegar.

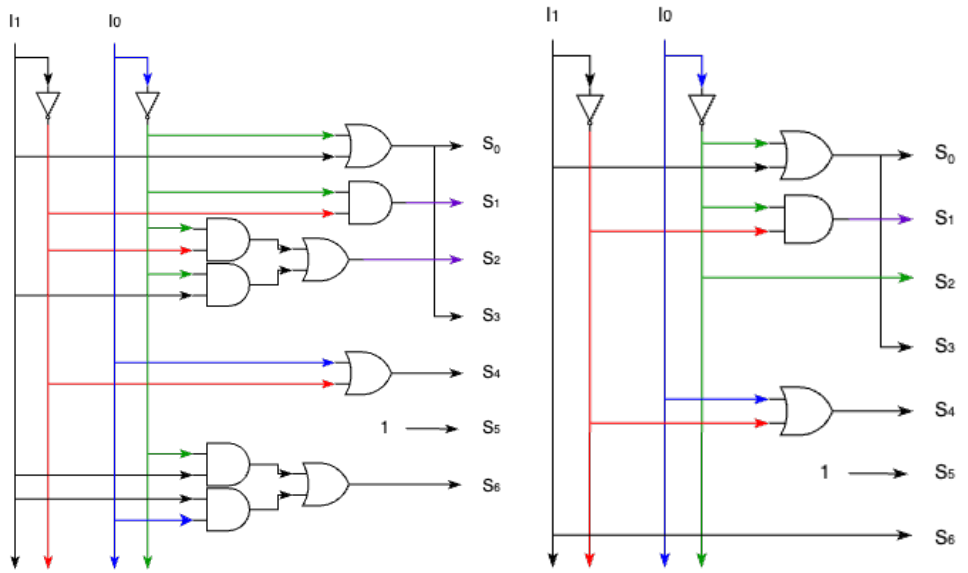
Solución: A continuación, la tabla de verdad para los segmentos S_i a partir de $I_1 I_0$:

I_1	I_0	S_0	S_1	S_2	S_3	S_4	S_5	S_6
0	0	1	1	1	1	1	1	0
0	1	0	0	0	0	1	1	0
1	0	1	0	1	1	0	1	1
1	1	1	0	0	1	1	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.

- $S_0 = S_3 = I_1 \vee \overline{I_0}$
- $S_1 = \overline{I_1} \wedge \overline{I_0}$
- $S_2 = (\overline{I_1} \wedge \overline{I_0}) \vee (I_1 \wedge \overline{I_0}) = \overline{I_0} \wedge (I_1 \vee \overline{I_1}) = \overline{I_0} \wedge 1 = \overline{I_0}$
- $S_4 = \overline{I_1} \vee I_0$
- $S_5 = 1$
- $S_6 = (I_1 \wedge \overline{I_0}) \vee (I_1 \wedge I_0) = I_1 \wedge (\overline{I_0} \vee I_0) = I_1 \wedge 1 = I_1$

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 exactamente igual a los incluidos en esta pauta.

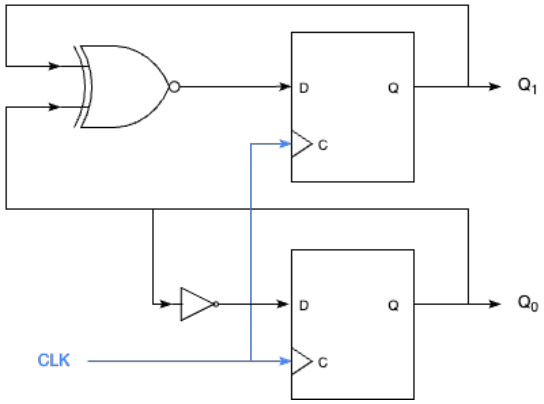
Se otorgan **0.4 ptos.** por el circuito correctamente diseñado de cada segmento, y **0.2 ptos.** si todos los circuitos son correctos. Por cada segmento, se otorga la mitad del puntaje si existe, como máximo, una combinación de señales de entrada que generen una salida incorrecta.

(b) (3 ptos.) Diseñe un contador secuencial de 2 bits que se decrementa con cada flanco de subida de la señal de control. Este contador, además, debe recibir una señal de entrada B de 1 bit, correspondiente al valor del botón del *timer*. Si el valor de B es igual a 1 durante el flanco de subida de la señal de control (*i.e.*, el botón está presionado), entonces el contador debe actualizar su valor a 3 en vez de decrementarse en una unidad.

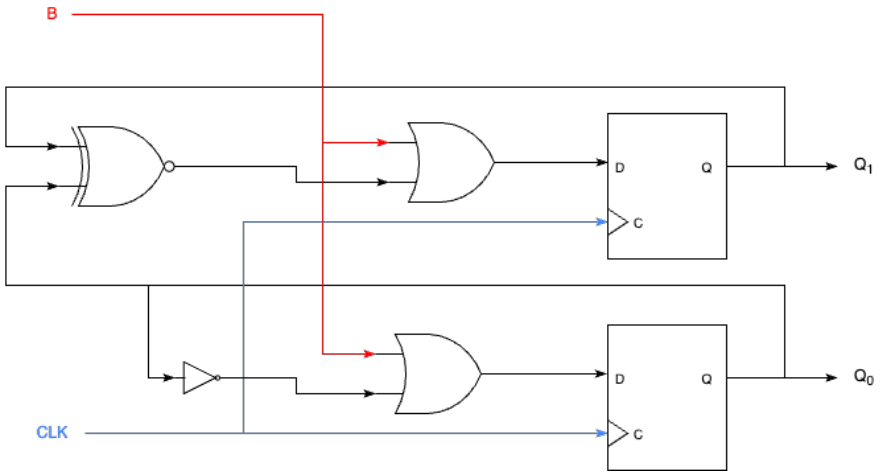
Solución: Si denotamos por $Q_1^t Q_0^t$ el estado del contador en un ciclo t de la señal de control `CLK`, y $Q_1^{t+1} Q_0^{t+1}$ su estado en el próximo ciclo, se puede elaborar la siguiente tabla de verdad:

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

Se deduce que $Q_0^{t+1} = \overline{Q_0^t}$, mientras que $Q_1^{t+1} = \overline{(Q_0^t \oplus Q_1^t)}$. 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, falta considerar la señal B. Si $B^t = 1$, entonces $Q_1^{t+1}Q_0^{t+1} = 11$. Por lo tanto, haciendo uso de la ley de dominación, se pueden ajustar las expresiones de los estados del contador de la siguiente forma: $Q_0^{t+1} = B^t \vee \overline{Q_0^t}$; y $Q_1^{t+1} = B^t \vee \overline{(Q_0^t \oplus Q_1^t)}$. Finalmente, el siguiente diagrama representa el contador modificado conectado a la señal B:



Es importante notar que se evaluará la correctitud del contador diseñado, no que sea exactamente igual al incluido en 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 B. Por cada criterio, se otorga la mitad del puntaje si existe, como máximo, un error de implementación.

Pregunta 3: Programabilidad (6 ptos.)

(a) (2 ptos.) Describa qué hace el fragmento **(a)**, escrito en el Assembly del computador básico del curso. Indique los valores de los registros `A` y `B` al finalizar la ejecución del código.

> **Solución:** El fragmento computa el inverso aditivo de la variable almacenada en el registro `A` y luego le suma el valor de la variable almacenada en el registro `B`. En otras palabras, a la variable almacenada en `B` se le resta la variable almacenada en `A`. Al finalizar, el programa ejecuta infinitamente la instrucción `JMP end`.
>
> Dado que `A` y `B` almacenan la misma variable (en este caso, `var1`), entonces se tiene que $\texttt{A} = \texttt{var1} - \texttt{var1} = 0$ y $\texttt{B} = \texttt{var1} = 255$. Esto, no obstante, responde a un *typo* en el enunciado, dado que se buscaba hacer uso de `var2`. Por lo tanto, también se considera correcto si computan $\texttt{A} = \texttt{var1} - \texttt{var2} = 78$ y $\texttt{B} = \texttt{var1} = 255$; o bien $\texttt{A} = \texttt{var2} - \texttt{var1} = -78$ y $\texttt{B} = \texttt{var2} = 177$.
>
> Se otorga **1 pto.** por la descripción correcta del programa, y **0.5 ptos.** por el valor correcto de cada registro.

(b) (2 ptos.) Describa qué hace el fragmento **(b)**, escrito en el Assembly del computador básico del curso. Indique los valores de los registros `A` y `B` al finalizar la ejecución del código.

> **Solución:** El fragmento obtiene el valor `NOT B` y lo utiliza como dirección de memoria para guardar el dato almacenado en ella en el registro `A`.
>
> Dado que `B` almacena el valor 254 de 8 bits, se tiene que $\texttt{NOT B} = 1$. Luego, como las variables se almacenan desde la dirección 0 de la memoria de datos, se tiene que $\texttt{A} = \texttt{Mem[1]} = \texttt{var2}$. Por lo tanto, se tiene que $A = 5$ y $B = 1$.
>
> Se otorga **1 pto.** por la descripción correcta del programa, y **0.5 ptos.** por el valor correcto de cada registro.

(c) (2 ptos.) El fragmento **(c)**, escrito en el Assembly del computador básico del curso, se encarga de revisar si un arreglo `arr1` es igual al inverso de otro arreglo `arr2`, es decir, que el primer elemento de `arr1` sea igual al último elemento de `arr2`; que el segundo elemento de `arr1` sea igual al penúltimo elemento de `arr2`, y así sucesivamente. En caso de cumplirse, se tendrá que `res == 1` al finalizar la ejecución, y `res == 0` en otro caso. No obstante, el cómputo final para los datos ingresados no es correcto. Identifique qué instrucción(es) está(n) errada(s) y explique justificadamente cómo corregirla(s) para que el código actualice `res` correctamente.

Solución: Este fragmento cuenta con dos errores:

1. La dirección de memoria del último elemento de `arr2` se obtiene como la suma entre su dirección base y su largo. No obstante, esta suma genera la dirección de la variable `iter`, ya que se excede en una unidad. Esto se arregla insertando `SUB A, 1` antes de `MOV (i2), A`, quinta instrucción del fragmento original.

2. Luego de realizar la comparación `CMP A, (len)`, se ejecuta `JEQ end`. Es decir, se termina la ejecución si `A` es igual al largo de los arreglos. En caso de no cumplirse, se quiere seguir iterando a través del salto a `loop`. No obstante, la instrucción `JNE loop` hará uso de las *flags* condicionales computadas por `JEQ end` y no las de `CMP A, (len)`, dado que estas se actualizan en cada ciclo. Por lo tanto, no se evaluará correctamente la condición de salto. Esto se puede resolver de dos formas: cambiando la instrucción `JNE loop` por `JMP loop`; o insertando nuevamente la instrucción `CMP A, (len)` posterior a `JEQ end`.

Por cada error, se otorgan **0.5 ptos.** por señalar correctamente el problema y **0.5 ptos.** por proponer una corrección correcta.

(a)

```
DATA:
  var1 255
  var2 177

CODE:
  MOV A, (var1)
  MOV B, (var1)
  JMP mysterium

  end:
  JMP end

  mysterium:
  NOT A, A
  ADD A, 1
  ADD A, B
  JMP end
```

(b)

```
DATA:
  var1 3
  var2 5

CODE:
  MOV B,254
  MOV A,B
  NOT A,A
  MOV B,A
  MOV A,(B)
```

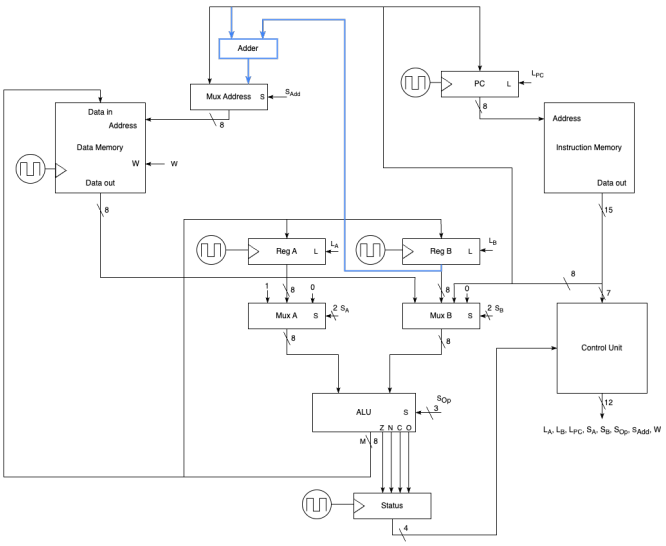
(c)

```
DATA:
  res   1
  len   4
  arr1  3
        5
        8
        2
  arr2  2
        8
        5
        3
  iter  0
  i1    0
  i2    0
  temp  0

CODE:
  MOV A, arr1
  MOV (i1), A
  MOV A, arr2
  ADD A, (len)
  MOV (i2), A

  loop:
  MOV B, (i1)
  MOV A, (B)
  MOV (temp), A

  MOV B, (i2)
  MOV A, (B)
  CMP A, (temp)
  JNE no_son_iguales

  INC (iter)
  INC (i1)
  MOV A, (i2)
  SUB A, 1
  MOV (i2), A

  MOV A, (iter)
  CMP A, (len)
  JEQ end
  JNE loop

  no_son_iguales:
  MOV A, 0
  MOV (res), A

  end:
```

Pregunta 4: Computador básico con saltos (6 ptos.)

(a) (3 ptos.) Modifique la arquitectura del computador básico para implementar las instrucciones `MOV A,(B+offset)`, `MOV (B+offset),A`, `MOV B,(B+offset)` y `MOV (B+offset),B`, *i.e.* instrucciones de direccionamiento indirecto con registro `B` y *offset*, siendo este último un literal. Para cada instrucción, deberá incluir la combinación completa de señales que la ejecutan. Por cada señal de carga y escritura, 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).

Solución: Para la implementación de estas instrucciones, se puede modificar la conexión entre el registro `B` y el componente `Mux Address` para que ahora reciba el resultado de un sumador que sume el valor de `B` y el literal de la instrucción:



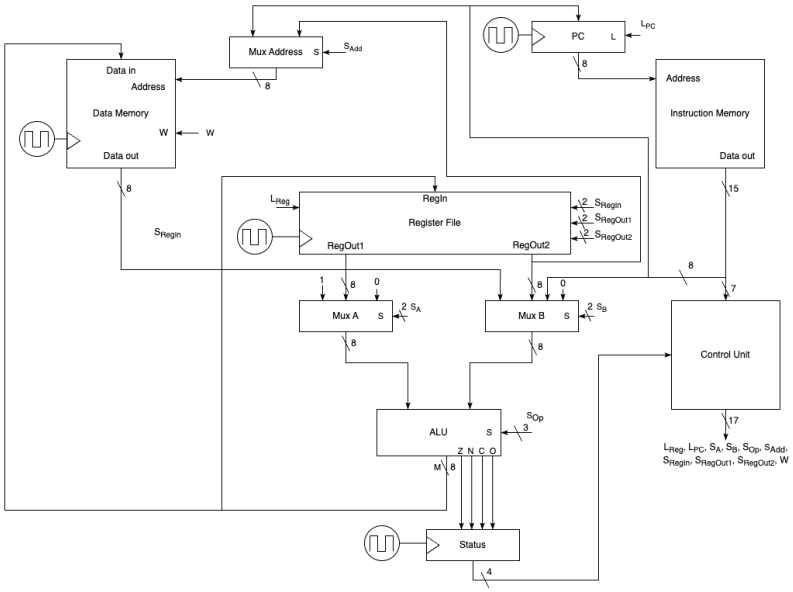
De esta forma, no es necesario agregar ni modificar señales, pero sí asegurar que el compilador haga uso del literal 0 para las instrucciones `MOV A,(B)`, `MOV B,(B)`, `MOV (B),A` y `MOV (B),B`. La tabla de señales es como sigue para las instrucciones implementadas:

Instrucción	L_A	L_B	L_{PC}	W	S_A	S_B	S_{OP}	S_{Add}
`MOV A,(B+offset)`	1	0	0	0	ZERO	DOUT	ADD	B+offset
`MOV B,(B+offset)`	0	1	0	0	ZERO	DOUT	ADD	B+offset
`MOV (B+offset),A`	0	0	0	1	A	ZERO	ADD	B+offset
`MOV (B+offset),B`	0	0	0	1	ZERO	B	ADD	B+offset

También es válido dejar como una tercera conexión la salida del sumador agregado. No obstante, en sería necesario señalar que la cantidad de bits de la señal S_{Add} incrementa a 2.

Se otorgan **1.5 ptos.** por la modificación correcta del diagrama y **0.375 ptos.** por cada instrucción correctamente implementada según su combinación de señales. Se otorga la mitad del puntaje en cada ítem si existe, como máximo, un error de implementación.

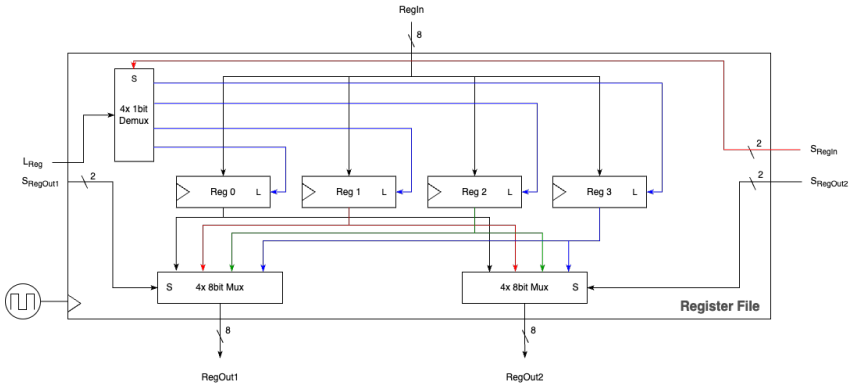
(b) (3 ptos.) Una limitante del computador básico estudiado en clases es que solo hace uso de dos registros y, además, no se puede usar cualquiera de estos sobre todas las operaciones. En la práctica, las arquitecturas hacen uso de un *Register File*, componente con múltiples registros en su interior. A continuación, se muestra el diagrama del computador básico modificado para hacer uso de este componente, el que cuenta con 4 registros en su interior:



Se puede observar que, a través de las señales de control $\texttt{S}_{\text{RegOut1}}$ y $\texttt{S}_{\text{RegOut2}}$, se pueden seleccionar los registros a ser utilizados como operandos de la ALU; mientras que $\texttt{S}_{\text{RegIn}}$ permite seleccionar el registro a ser sobreescrito si $\texttt{L}_{\text{Reg}}$ es igual a 1. A partir de esta versión del computador básico, desarrolle las siguientes preguntas:

1. (1.5 ptos.) Elabore el diagrama interno del *Register File* para que cumpla con el comportamiento descrito. Puede agregar las conexiones y componentes que estime convenientes.

Solución: A continuación, se muestra una posible implementación del *Register File*:



En resumen:

- Se agrega el componente `4x 1bit Demux` para transmitir la señal de carga $\texttt{L}_{\texttt{Reg}}$ a solo uno de los cuatro registros internos. Se conecta a la señal $\texttt{S}_{\texttt{RegIn}}$ para realizar la selección.

- Se conecta la señal de entrada `RegIn` a la entrada de todos los registros internos. Esto no es un problema, dado que su valor se actualizará solo si es que se les transmite la señal de carga habilitada según el valor de las señales $\texttt{L}_{\texttt{Reg}}$ y $\texttt{S}_{\texttt{RegIn}}$.
- Se agregan dos componentes `4x 8bit Mux` para seleccionar uno de los cuatro registros y transmitirlos a las salidas `RegOut1` y `RegOut2`. Para realizar la selección, se conectan a las señales $\texttt{S}_{\texttt{RegOut1}}$ y $\texttt{S}_{\texttt{RegOut2}}$ respectivamente.
- Se asume que todos los registros se conectan a la misma señal de *clock*, no es necesario explicitar la conexión.

Se otorgan **0.5 ptos.** por transmitir correctamente la señal de carga $\texttt{L}_{\texttt{Reg}}$ a los registros; **0.5 ptos.** por transmitir correctamente la señal `RegIn` a la entrada de los registros; y **0.5 ptos.** por transmitir correctamente el registro seleccionado para cada salida `RegOut1` y `RegOut2` según las señales de selección $\texttt{S}_{\texttt{RegOut1}}$ y $\texttt{S}_{\texttt{RegOut2}}$. Por cada criterio, se otorga la mitad del puntaje si existe, como máximo, un error de implementación.

2. (1.5 ptos.) Indique la combinación de señales completa para llevar a cabo la ejecución de las siguientes instrucciones:
   - (0.3 ptos.) `MOV Reg0, Reg3`. Almacena en `Reg0` el valor `Reg3`.
   - (0.3 ptos.) `MOV Reg2, Lit`: Almacena en `Reg2` el literal de la instrucción.
   - (0.3 ptos.) `MOV (Reg0), Reg1`. Almacena en `Mem[Reg0]` el valor `Reg1`.
   - (0.3 ptos.) `XOR Reg0, Reg1, Reg2`. Almacena en `Reg0` el valor `Reg1 XOR Reg2`.
   - (0.3 ptos.) `NOT Reg1, Reg1`. Almacena en `Reg1` el valor `NOT Reg1`.

Solución: A continuación, la combinación de señales por instrucción:

Instrucción	$\texttt{L}_{\texttt{RegIn}}$	$\texttt{L}_{\texttt{PC}}$	W	$\texttt{S}_{\texttt{RegIn}}$	$\texttt{S}_{\texttt{RegOut1}}$	$\texttt{S}_{\texttt{RegOut2}}$	$\texttt{S}_{\texttt{A}}$	$\texttt{S}_{\texttt{B}}$	$\texttt{S}_{\texttt{OP}}$	$\texttt{S}_{\texttt{Add}}$
`MOV Reg0, Reg3`	1	0	0	`Reg0`	-	`Reg3`	`ZERO`	`RegOut2`	`ADD`	-
`MOV Reg2, Lit`	1	0	0	`Reg2`	-	-	`ZERO`	`LIT`	`ADD`	-
`MOV (Reg0), Reg1`	0	0	1	-	`Reg1`	`Reg0`	`Reg1`	`ZERO`	`ADD`	`RegOut2`
`XOR Reg0, Reg1, Reg2`	1	0	0	`Reg0`	`Reg1`	`Reg2`	`RegOut1`	`RegOut2`	`XOR`	-
`NOT Reg1, Reg1`	1	0	0	`Reg1`	`Reg1`	-	`RegOut1`	-	`NOT`	-

Cabe destacar que la instrucción `MOV (Reg0), Reg1` tenía un *typo* en la descripción del enunciado, señalando erróneamente que `Reg1` se almacena en `Mem[Reg3]`. Por este motivo, se considera también correcto que se indique $\texttt{S}_{\texttt{RegOut2}} = \texttt{Reg3}$.

Se otorgan **0.3 ptos.** por cada combinación correcta de señales y, en caso de existir como máximo una señal errónea, se otorga la mitad del puntaje.