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

IIC2343 - Arquitectura de Computadores (II/2023)

Interrogación 1

Pauta de evaluación

Consideraciones

- Respuestas sin desarrollo o justificación no tendrán puntaje.
- Cada pregunta podría tener más de un desarrollo válido. La pauta evalúa eso y este documento solo muestra una alternativa de solución.

Pregunta 1: Preguntas conceptuales (8 ptos.)

(a) (2 ptos.) Un estudiante implementa un *script* para computar el inverso aditivo de un número en base binaria representado en complemento de 2. Para verificar su correctitud, imprime la suma entre un número y su inverso aditivo. No obstante, el resultado, interpretado como número entero en base binaria, es igual a -1. Indique qué error cometió el estudiante y cómo se puede resolver.

Solución: En este caso, si -1 se interpretó como número entero en base binaria, este corresponde a la secuencia binaria que solo posee bits iguales a uno. Esto ocurre en particular al realizar la suma de un número con su inverso computado con **complemento de 1** (la negación de todos los bits del número original). Para resolverlo, basta con incrementar en una unidad al inverso aditivo computado para que la suma resultante sea igual a cero (descartando el bit de *carry* generado) y el *script* imprima el cero esperado. Se otorga **1 pto.** por señalar correctamente el error y **1 pto.** por explicar cómo se resuelve.

(b) Suponga que se crea el tipo de dato `arquiFloat` de 32 bits a partir del nuevo estándar "IIC2343" para representar números de punto flotante. A diferencia del tipo de dato `float` del estándar IEEE754, este posee el siguiente formato:

- Un bit de signo de significante.
- Un bit de signo de exponente.
- 10 bits de exponente **no desfasado**.
- 20 bits de significante **normalizado** con bit implícito.

Compare los dos estándares respondiendo las siguientes preguntas:

1. (1 pto.) Señale una ventaja y una desventaja del tipo de dato `arquiFloat` respecto del tipo de dato `float`.

   Solución: Una ventaja es que `arquiFloat` posee un rango mayor respecto a los números que puede representar dada su cantidad de bits de exponente. En este caso, el mayor exponente representable es $1111111110b = 1022$ (si se asume la reserva para los valores infinitos y con bit de signo positivo), que es significativamente mayor al exponente máximo de `float` igual a 127. Por otra parte, una desventaja es la pérdida de precisión. En este caso, si incluimos el bit implícito, `arquiFloat` puede representar números de hasta 21 bits en comparación a `float` que puede representar números de hasta 24 bits. Se otorgan **0.5 ptos.** por indicar justificadamente una ventaja y **0.5 ptos.** por indicar justificadamente una desventaja.

2. (3 ptos.) Señale cómo se representa el resultado de la operación $2^{22} + 7$ en formato `float` y `arquiFloat`. ¿Existe pérdida de precisión en alguno de los casos?

   Solución: El número que se busca representar, haciendo la suma entre 2^{22} y 7 con notación científica, es igual a $1{,}00000000000000000111 \times 10^{00010110}$. A continuación veremos cómo se almacena en cada representación:

   - `float`: El número es positivo (bit de signo igual a cero) y el exponente desfasado es igual a $22 + 127 = 149$. Por lo tanto, se almacena la secuencia $0_{\text{signo}}10010101_{\text{exp.}}00000000000000000001110_{\text{sig.}}$. No existe pérdida de precisión. Se otorgan **0.75 ptos.** por la representación y **0.75 ptos.** por señalar que no se pierde precisión.

   - `arquiFloat`: Tanto el número como su exponente son positivos (bits de signo iguales a cero). Por lo tanto, se almacena la secuencia $0_{\text{signo sig.}}0_{\text{signo exp.}}0000010110_{\text{exp.}}00000000000000000001_{\text{sig.}}$. Se pierden dos bits de precisión, por lo que se pierde el valor exacto del número: se redondea a $2^{22} + 4$. Se otorgan **0.75 ptos.** por la representación y **0.75 ptos.** por señalar que se pierde precisión. En este caso, dado que no se explicita, no importa el orden en el que se guarden las secuencias de bits, pero sí que cada una de ellas sea correcta.

(c) (2 ptos.) Indique el valor final de los registros `A` y `B` luego de ejecutar el siguiente fragmento de código escrito en el Assembly del computador básico.

```
MOV A,7
MOV B,1
PUSH B
PUSH A
RET
MOV A,0
MOV B,0
JMP end
end:
```

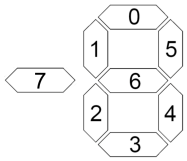
Solución: Cuando se ejecuta el fragmento de código anterior, `RET` gatilla la carga del valor numérico del tope del *stack* en el registro `PC`. En este caso, el tope del `stack` posee el valor 7 y por ende el programa saltará a dicha dirección en la memoria de instrucciones. A continuación, se muestra el mismo fragmento mostrando las direcciones en las que se almacena cada instrucción.

```
MOV A,7    ;Dirección 0x00
MOV B,1    ;Dirección 0x01
PUSH B     ;Dirección 0x02
PUSH A     ;Dirección 0x03
RET        ;Dirección 0x04-0x05
MOV A,0    ;Dirección 0x06
MOV B,0    ;Dirección 0x07       <- RET gatilla un salto aquí.
JMP end    ;Dirección 0x08
end:
```

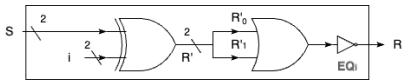
Por lo tanto, el programa terminará ejecutando las instrucciones `MOV B,0` y `JMP end`, saltándose `MOV A,0`. De esta forma, el valor final del registro `A` es igual a 7 y el del registro `B` es igual a 0. Se otorga **1 pto.** por cada registro donde se indique su valor final correcto.

Pregunta 2: Circuitos digitales (8 ptos.)

Se busca implementar el *display* de un ascensor cuyos pisos van del -2 al 1 (siendo 0 el piso base). Construya, a partir de compuertas lógicas, el controlador de este *display*. Este recibe como *input* una señal de 2 bits que representa un número entero con signo equivalente al número del piso del ascensor. Puede basarse en la figura para señalar los segmentos del *display* que deben encenderse (1) o apagarse (0) para cada *input*.



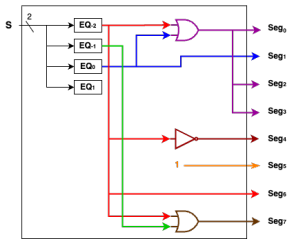
Solución: Para construir este circuito, en primer lugar definimos el componente $\texttt{EQ}_i$:



Su salida es 1 si S es igual al valor i. Lo usaremos con los valores $i = -2, -1, 0, 1$ y veremos las condiciones para que se encienda cada segmento del *display*:

$$\texttt{Seg}_0 = \texttt{EQ}_{-2} \vee \texttt{EQ}_0$$
$$\texttt{Seg}_1 = \texttt{EQ}_0$$
$$\texttt{Seg}_2 = \texttt{EQ}_{-2} \vee \texttt{EQ}_0$$
$$\texttt{Seg}_3 = \texttt{EQ}_{-2} \vee \texttt{EQ}_0$$
$$\texttt{Seg}_4 = \texttt{EQ}_{-1} \vee \texttt{EQ}_0 \vee \texttt{EQ}_1 \equiv \neg\texttt{EQ}_{-2}$$
$$\texttt{Seg}_5 = \texttt{EQ}_{-2} \vee \texttt{EQ}_{-1} \vee \texttt{EQ}_0 \vee \texttt{EQ}_1 \equiv 1$$
$$\texttt{Seg}_6 = \texttt{EQ}_{-2}$$
$$\texttt{Seg}_7 = \texttt{EQ}_{-2} \vee \texttt{EQ}_{-1}$$

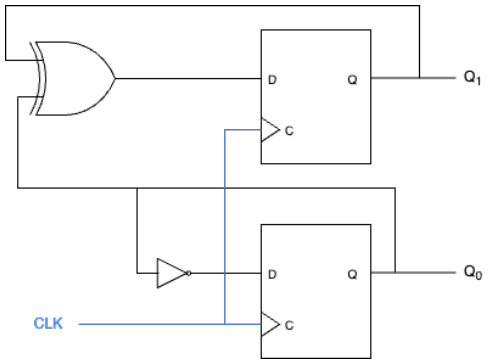
Por lo tanto, el diagrama interno del controlador se construye de la siguiente forma:



Se otorga **1 pto.** por segmento bien computado y **0.5 ptos.** si el control del segmento falla como máximo en un caso.

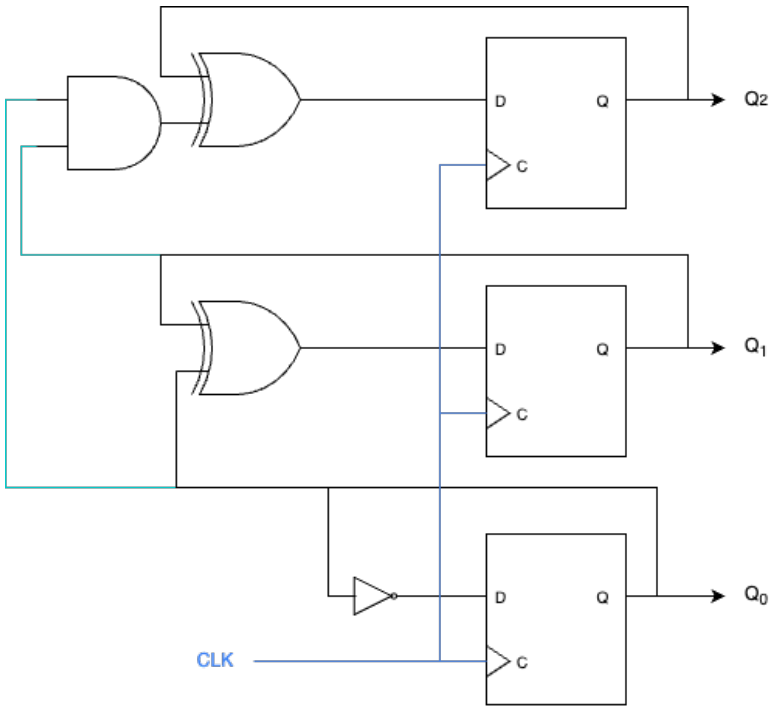
Pregunta 3: Almacenamiento de datos (8 ptos.)

(a) (4 ptos.) Suponga que tiene el siguiente contador de dos bits:



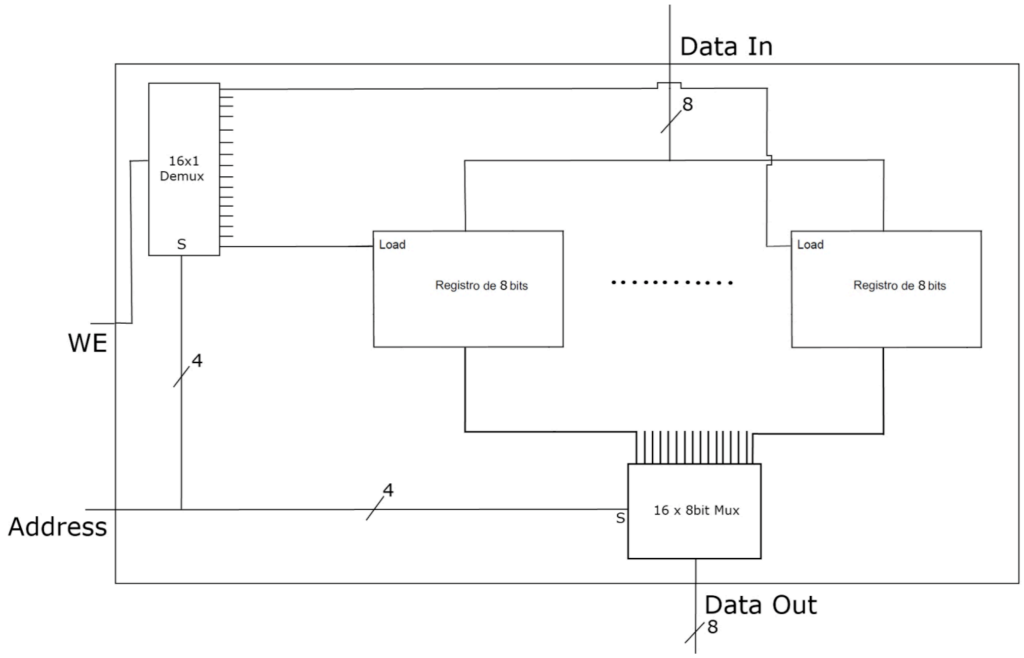
Este parte en 00 e incrementa una unidad por cada flanco de subida, volviendo a 00 por *overflow* una vez que llega a su valor máximo. Agregue a este contador un tercer flip-flop que represente el bit más significativo de un nuevo contador de tres bits cuyo valor incrementa en una unidad por cada flanco de subida.

Solución: Para este contador, es importante notar que el bit a añadir (Q_2) **alterna** su valor si, y solo si $Q_1 = Q_0 = 1$. Por lo tanto, se puede definir como $Q_2^{t+1} = (Q_1^t \wedge Q_0^t) \oplus Q_2$, como se muestra a continuación:



Se otorgan **2 ptos.** si el estado Q_2 se actualiza correctamente de 0 a 1; y **2 ptos.** si el estado Q_2 se actualiza correctamente de 1 a 0 según los valores de Q_1 y Q_0.

(b) (4 ptos.) A continuación, se adjunta un posible diagrama interno de una RAM de 16 direcciones:

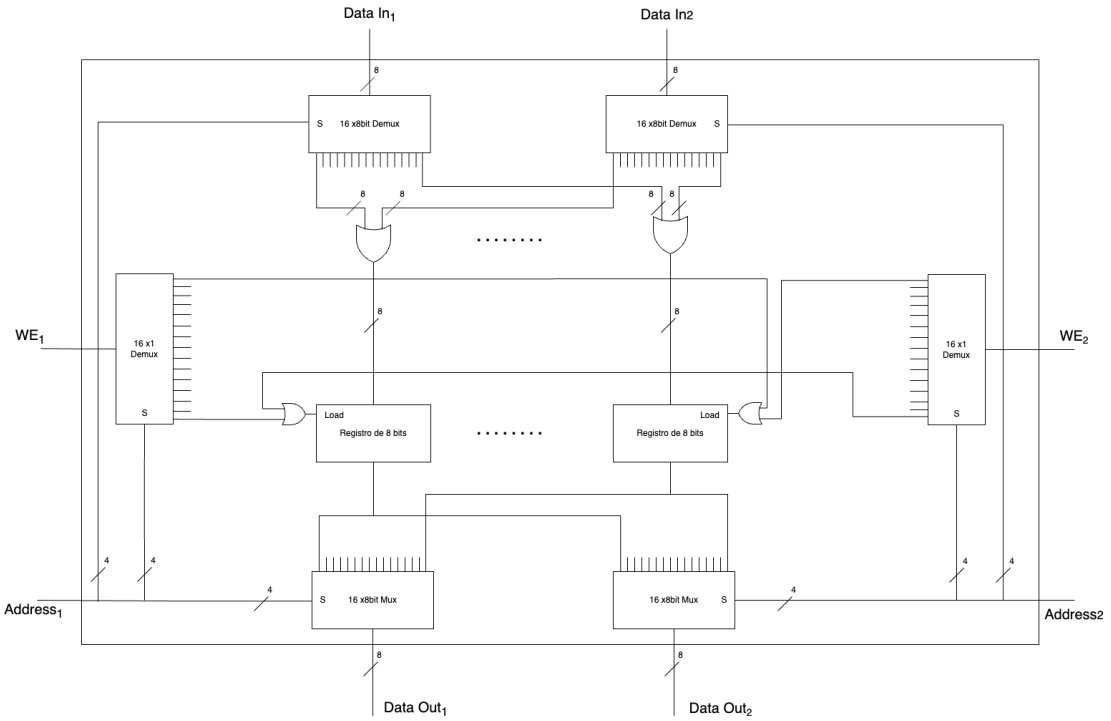


Modifique este diagrama para que reciba entradas `DataIn_1`; `DataIn_2`; `Address_1`; `Address_2`; `WE_1`; `WE_2`, y salidas `DataOut_1` y `DataOut_2` para leer y escribir datos en dos direcciones de memoria **distintas** de forma **simultánea**.

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

- Agregamos un segundo multiplexor cuya señal de selección es igual a `Address_2` y recibe como entradas los valores de todos los registros internos. Esto permite escoger una segunda salida `DataOut_2`.
- Agregamos un segundo demultiplexor cuya señal de selección es igual a `Address_2` y recibe como entrada la señal `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 `WE_1` está activo para su dirección `Address_1` o si `WE_2` está activo para su dirección `Address_2`.
- Para manejar las dos entradas `DataIn_1` y `DataIn_2`, conectamos cada una de ellas a un demultiplexor 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 plasmadas en el siguiente diagrama:



El puntaje se distribuye de la siguiente forma:

- **1 pto.** por computar correctamente las salidas `DataOut_1` y `DataOut_2`. Se otorgan **0.5 ptos.** si existe como máximo un error de implementación.
- **1.5 ptos.** por transmitir correctamente la segunda señal de escritura `WE_2`. Se otorgan **0.75 ptos.** si existe como máximo un error de implementación.
- **1.5 ptos.** por transmitir correctamente el segundo dato de entrada `DataIn_2`. Se otorgan **0.75 ptos.** si existe como máximo un error de implementación.

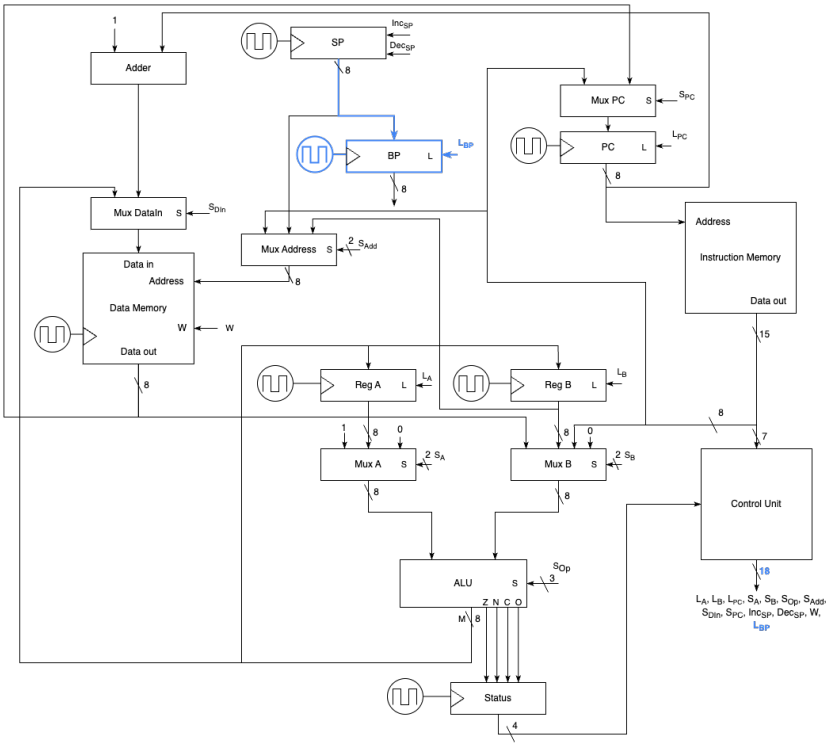
Si el problema se resolvió duplicando la cantidad de registros, se otorga como máximo **1 pto.** del total dado que esto no es lo que se pide por enunciado.

Pregunta 4: Computador básico (8 ptos.)

En varias arquitecturas se hace uso del registro *base pointer* (`BP`). Este funciona como punto de referencia para obtener los parámetros y dirección de retorno de cada llamada de una subrutina, lo que facilita el manejo de llamados anidados. Deberá modificar la microarquitectura del computador básico para implementar el registro `BP` de 8 bits y su funcionamiento. Se le listarán las instrucciones que deben ser implementadas. 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). Puede realizar todas las modificaciones en un solo diagrama.

(a) (1 pto.) `MOV BP,SP`. Almacena en el registro `BP` el valor del registro `SP`.

Solución: Para esta modificación, agregamos el registro `BP` a nuestra arquitectura con una nueva señal de carga L_{BP} y con valor de entrada igual a la salida de `SP`. De momento, no conectamos la salida de `BP` con ninguna compuerta (se asume que va a tierra).



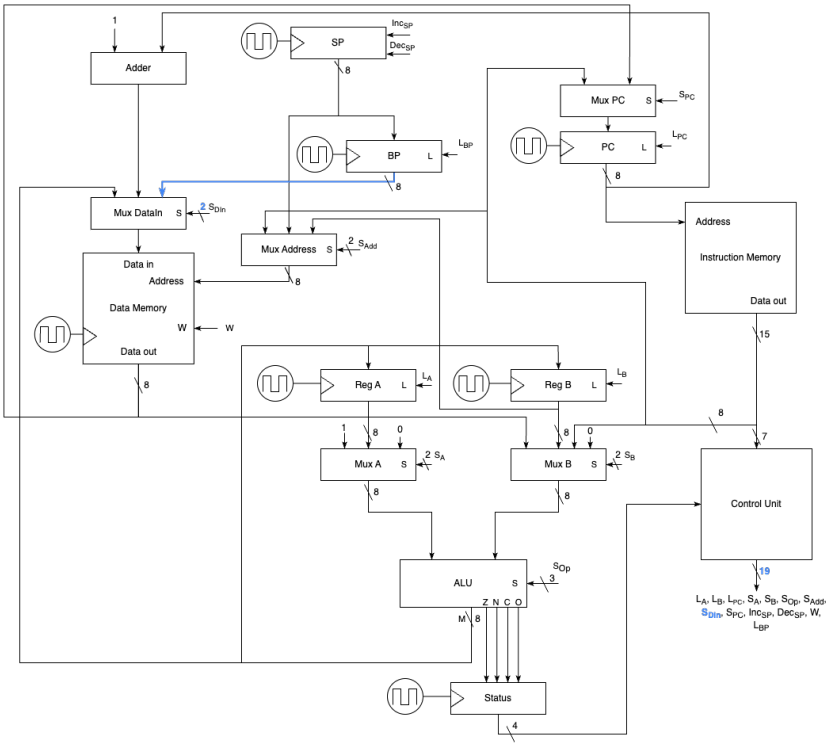
Para ejecutar la instrucción, se utiliza la siguiente combinación de señales:

Instrucción	L_A	L_B	L_{PC}	W	Inc_{SP}	Dec_{SP}	S_A	S_B	S_{OP}	S_{Add}	S_{DIn}	S_{PC}	L_{BP}
`MOV BP,SP`	0	0	0	0	0	0	-	-	-	-	-	-	1

Se otorgan **0.5 ptos.** por la correctitud de la modificación y **0.5 ptos.** por entregar una combinación de señales correcta para la instrucción.

(b) (1 pto.) `PUSH BP`. Almacena en `Mem[SP]` el valor de `BP` y decrementa `SP` en una unidad.

Solución: Para esta modificación, conectamos la salida del registro `BP` con una de las entradas del componente `Mux DataIn`. Para ello, es necesario incrementar la cantidad de bits de la señal S_{DIn} a dos.



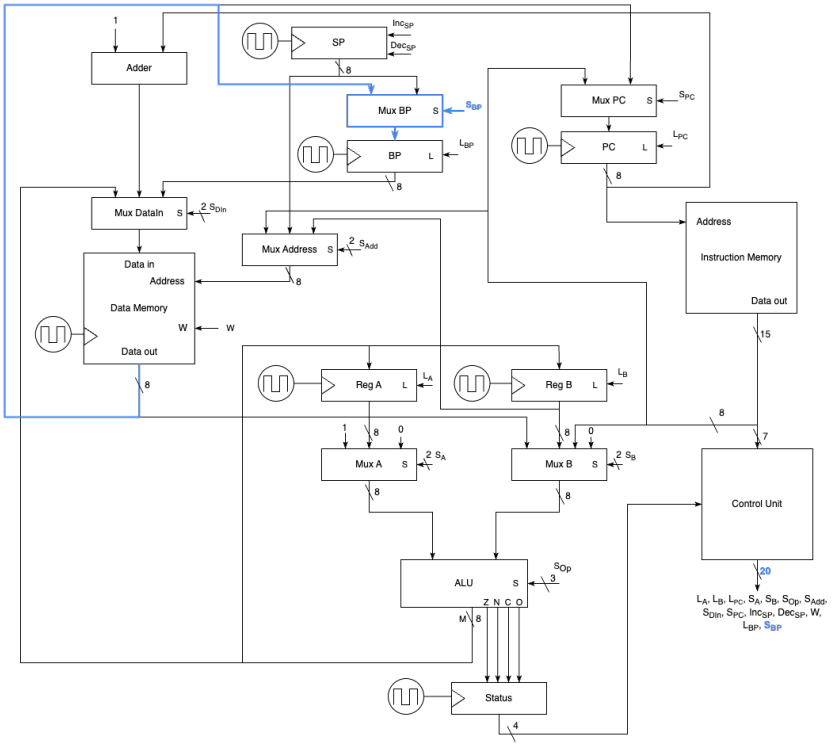
Para ejecutar la instrucción, se utiliza la siguiente combinación de señales:

Instrucción	L_A	L_B	L_{PC}	W	Inc_{SP}	Dec_{SP}	S_A	S_B	S_{OP}	S_{Add}	S_{DIn}	S_{PC}	L_{BP}
PUSH BP	0	0	0	1	0	1	-	-	-	SP	BP	-	0

Se otorgan **0.5 ptos.** por la correctitud de la modificación y **0.5 ptos.** por entregar una combinación de señales correcta para la instrucción.

(c) (1 pto.) `POP BP`. Incrementa `SP` en una unidad y almacena en `BP` el valor `Mem[SP]`.

Solución: Para esta modificación, conectamos la salida de la memoria de datos con la entrada de `BP`. Dado que ya existe una entrada para `SP`, se agrega un nuevo componente `Mux BP` con señal de selección S_{BP} para escoger el valor de carga entre `SP` y `DOUT`.



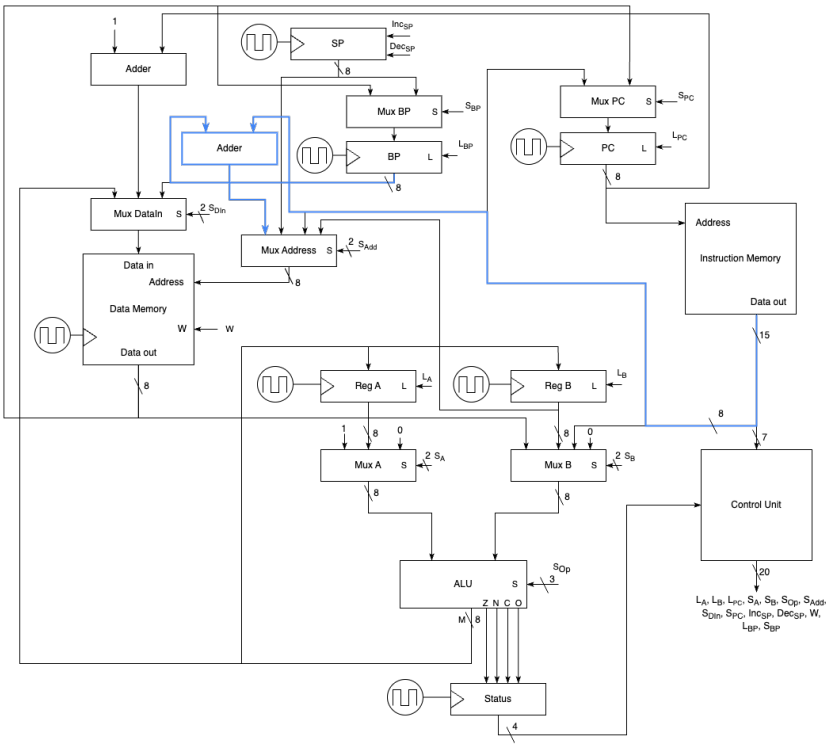
Para ejecutar la instrucción, se utiliza la siguiente combinación de señales:

Instrucción	L_A	L_B	L_{PC}	W	Inc_{SP}	Dec_{SP}	S_A	S_B	S_{OP}	S_{Add}	S_{DIn}	S_{PC}	L_{BP}	S_{BP}
POP BP	0	0	0	0	1	0	-	-	-	-	-	-	0	-
	0	0	0	0	0	0	-	-	-	SP	-	-	1	DOUT

Se otorgan **0.5 ptos.** por la correctitud de la modificación y **0.5 ptos.** por entregar una combinación de señales correcta para la instrucción.

(d) (2 ptos.) `MOV Reg,(BP + Lit)`. Almacena en `Reg` (`A` o `B`) el valor `Mem[BP + Lit]`.

Solución: Para esta modificación, agregamos un `Adder` que recibirá como entradas el valor del registro `BP` y el literal `LIT` de la memoria de instrucciones, para luego conectar su salida al componente `Mux Address`.

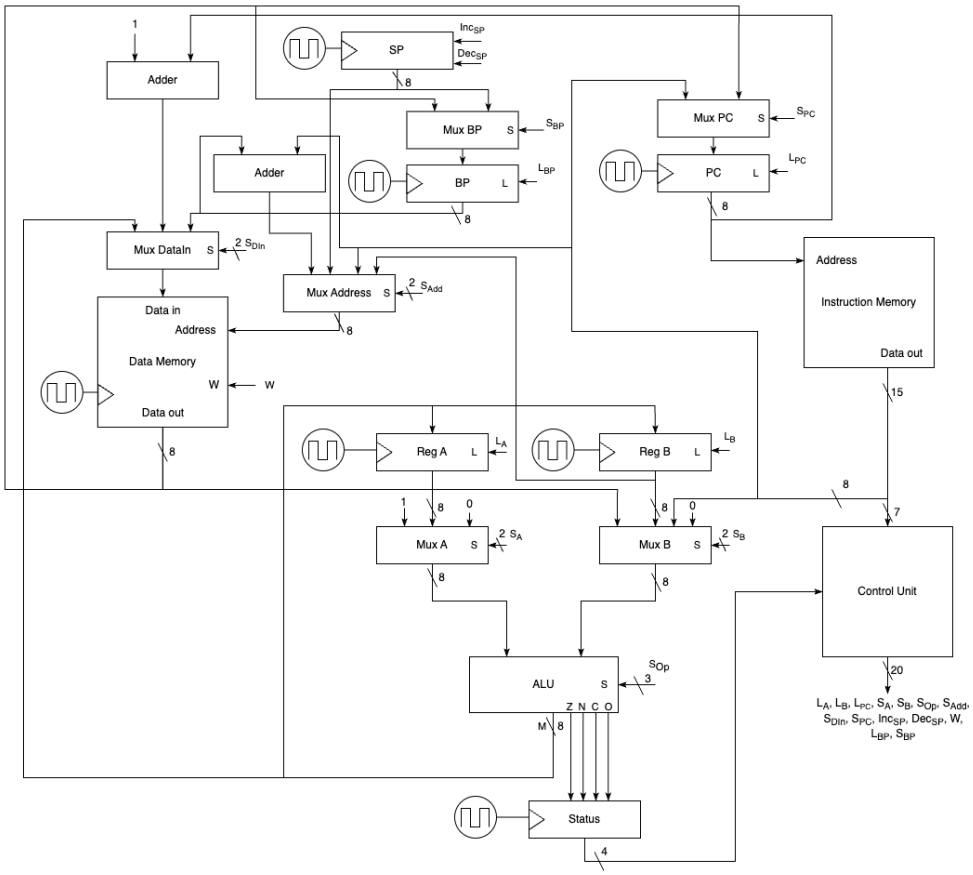


Para ejecutar las instrucciones, se utilizan la siguientes combinaciones de señales:

Instrucción	L_A	L_B	L_{PC}	W	Inc_{SP}	Dec_{SP}	S_A	S_B	S_{OP}	S_{Add}	S_{DIn}	S_{PC}	L_{BP}	S_{BP}
`MOV A,(BP + Lit)`	1	0	0	0	0	0	0	DOUT	ADD	BP+LIT	-	-	0	-
`MOV B,(BP + Lit)`	0	1	0	0	0	0	0	DOUT	ADD	BP+LIT	-	-	0	-

Se otorgan **1.5 ptos.** por la correctitud de la modificación y **0.5 ptos.** por entregar una combinación de señales correcta para una o ambas instrucciones. Se otorgan **0.75 ptos.** por la modificación si esta presenta como máximo un error de implementación.

En la siguiente página se muestra el diagrama completo de la arquitectura esperada y su tabla de instrucciones, asumiendo que se implementan todas las modificaciones juntas.



Instrucción	L_A	L_B	L_{PC}	W	Inc_{SP}	Dec_{SP}	S_A	S_B	S_{OP}	S_{Add}	S_{DIn}	S_{PC}	L_{BP}	S_{BP}
MOV BP,SP	0	0	0	0	0	0	-	-	-	-	-	-	1	SP
PUSH BP	0	0	0	1	0	1	-	-	-	SP	BP	-	0	-
POP BP	0	0	0	0	1	0	-	-	-	-	-	-	0	-
	0	0	0	0	0	0	-	-	-	SP	-	-	1	DOUT
MOV A,(BP + Lit)	1	0	0	0	0	0	0	DOUT	ADD	BP+LIT	-	-	0	-
MOV B,(BP + Lit)	0	1	0	0	0	0	0	DOUT	ADD	BP+LIT	-	-	0	-

(e) (3 ptos.) Asuma que se agregan las instrucciones anteriores a la ISA del computador básico y que se ejecuta el programa adjunto. Explique, a grandes rasgos, lo que realiza la subrutina `func` y señale el valor final de la variable `result`. Puede asumir que `BP` parte en 255.

```
DATA:
  n      3
  result 0
CODE:
  MOV A,(n)          // Dirección Mem. Instr.: 0x00
  PUSH A             // Dirección Mem. Instr.: 0x01
  CALL func          // Dirección Mem. Instr.: 0x02
  POP A              // Dirección Mem. Instr.: 0x03-0x04
  JMP end            // Dirección Mem. Instr.: 0x05
  func:
    PUSH BP          // Dirección Mem. Instr.: 0x06
    MOV BP,SP        // Dirección Mem. Instr.: 0x07
    MOV A,(BP + 3)   // Dirección Mem. Instr.: 0x08
    CMP A,0          // Dirección Mem. Instr.: 0x09
    JEQ base_end_0   // Dirección Mem. Instr.: 0x0A
    CMP A,1          // Dirección Mem. Instr.: 0x0B
    JEQ base_end_1   // Dirección Mem. Instr.: 0x0C
    SUB A,1          // Dirección Mem. Instr.: 0x0D
    PUSH A           // Dirección Mem. Instr.: 0x0E
    CALL func        // Dirección Mem. Instr.: 0x0F
    POP A            // Dirección Mem. Instr.: 0x10-0x11
    MOV A,(BP + 3)   // Dirección Mem. Instr.: 0x12
    SUB A,2          // Dirección Mem. Instr.: 0x13
    PUSH A           // Dirección Mem. Instr.: 0x14
    CALL func        // Dirección Mem. Instr.: 0x15
    POP A            // Dirección Mem. Instr.: 0x16-0x17
    JMP end_func     // Dirección Mem. Instr.: 0x18
    base_end_1:
      INC (result)   // Dirección Mem. Instr.: 0x19
    base_end_0:
      NOP            // Dirección Mem. Instr.: 0x1A
    end_func:
      POP BP         // Dirección Mem. Instr.: 0x1B-0x1C
      RET            // Dirección Mem. Instr.: 0x1D-0x1E
  end:
```

Solución: La subrutina `func` corresponde a la función de recurrencia de Fibonacci, *i.e.*:

$$F_N = \begin{cases} F_{N-1} + F_{N-2} & N > 1 \\ N & N \leq 1 \end{cases}$$

Esto se deduce de la siguiente forma:

- Todo `PUSH A` se realiza antes de un llamado a `func`, siendo este el parámetro `n`.
- Mediante `MOV A,(BP + 3)` se recupera el argumento respaldado. Por cada llamado se guardan en el *stack* los valores `A` (parámetro), `PC+1` (retorno) y `BP`. Como `SP` apunta siempre una dirección sobre el tope del *stack* y `BP` tiene el mismo valor por `MOV BP,SP`, entonces `BP + 3` apunta a la dirección en la que se guardó el parámetro.
- Se realizan dos llamados recursivos al interior de `func`, siendo uno de ellos realizado con el parámetro `n - 1` (proveniente de `SUB A,1`) y el otro con el parámetro `n - 2` (proveniente de `SUB A,2`).

- `base_end_0` y `base_end_1` corresponden a los casos base de la recursión y se llega a ellos si, y solo si `A` es 0 o 1 respectivamente. Interesa sobre todo `base_end_1`, dado que en dicho caso se incrementa el valor de la variable `result`.
- Al terminar un llamado de la subrutina en `end_func`, se reestablece el *stack* al ejecutar `POP BP`, `RET` y `POP A` luego de retornar. Esto asegura que siempre se utilice el valor correcto de `BP` para acceder al parámetro de cada llamado recursivo.

A partir de lo anterior, se deduce entonces que el valor de `result` es igual a:

$$\begin{aligned} F_3 &= F_2 + F_1 \\ &= F_1 + F_0 + F_1 \\ &= 1 + 0 + 1 \\ &= 2 \end{aligned}$$

Se otorgan **2 ptos.** por señalar correctamente el resultado de la ejecución correctamente y **1 pto.** por describir `func` a partir de Fibonacci. Si no se cumple ninguno de los dos criterios anteriores, se otorga **1 pto.** si se muestra un desarrollo parcial pero correcto de la ejecución de `func`.