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

IIC2343 - Arquitectura de Computadores (I/2024)

Interrogación 2

Pauta de evaluación

Pregunta 1: Representación de Números Reales (6 ptos.)

(a) (1.5 ptos.) Suponga que posee dos representaciones para números reales que puede utilizar para almacenar el número 101,10011b. La primera corresponde a una representación de punto fijo con la siguiente estructura: 1 bit de signo; 3 bits de parte entera; y 4 bits de parte fraccional. La segunda, en contraste, corresponde a una representación de punto flotante **sin bit implícito** con la siguiente distribución: 1 bit de signo de significante; 4 bits de significante normalizado; 1 bit de signo de exponente; y 2 bits de exponente. ¿Cuál de estas representaciones recomendaría para almacenar el número? Justifique su respuesta en base a la precisión de la representación.

Solución: Para este número, es recomendable hacer uso de la representación de **punto fijo**. A través de ella, el número se almacena como: $0_{\text{Signo}}101_{\text{Entero}}1001_{\text{Signo}}$, donde se deduce que se pierde el bit menos significativo de la parte fraccional. En cambio, en el formato de punto flotante, su representación normalizada equivale a $1,0110011 \times 10^{10}$, por lo que se almacena la siguiente secuencia: $0_{\text{Signo sig.}}1011_{\text{Significante}}0_{\text{Signo exp.}}10_{\text{Exponente}}$. Se observa que en esta última representación se pierden los 4 bits menos significativos, por lo que su resultado es menos preciso. Se otorgan **0.75 ptos.** por indicar la representación que implica menor pérdida de precisión; y **0.75 ptos.** por justificar indicando la cantidad de bits perdidos en cada caso.

(b) (1.5 ptos.) Suponga que se quiere utilizar el computador básico visto en el curso para realizar operaciones sobre **números reales** bajo la representación de **punto fijo**. Para ello, los tres bits más significativos de los registros `A`, `B` se reservan para la parte entera; mientras que los cinco bits restantes se utilizan para la parte fraccional. Considerando que **no se realizan modificaciones de *hardware***, ¿existirá algún problema al momento de realizar operaciones con la ALU? Justifique su respuesta solo aludiendo a las operaciones aritméticas (`ADD`, `SUB`, `SHL`, `SHR`) ejecutadas en la arquitectura.

Solución: Si se opera con registros que almacenan números representados por punto fijo con **la misma cantidad de bits fraccionales** (en este caso, 5), entonces la suma y resta se pueden expresar como: $A \pm B = (A' \times 10^{-5} \pm B' \times 10^{-5}) = (A' \pm B') \times 10^{-5}$. Es decir, podemos computar de forma consistente la suma y la resta de los registros, aplicando solo el punto decimal al momento de representar el resultado. Por otra parte, las operaciones de *shifting* seguirán funcionando como la multiplicación o división por 2, solo que manteniendo una cantidad constante de puntos decimales. Por lo tanto, **no existe problema en operar con esta representación a nivel de *hardware***. Se otorgan **0.75 ptos.** por señalar correctamente que no existen problemas en términos aritméticos; y **0.75 ptos.** por justificación. En este último criterio, se aceptan ejemplos para mostrar que los resultados no varían en esta representación.

(c) (3 ptos.) En los siguientes incisos, se le entregará un par de números reales A, B en formato float con notación hexadecimal. En cada caso, debe interpretar sus valores y realizar las operaciones de suma o multiplicación solicitadas. Luego, debe indicar si la representación en formato float del resultado presenta cambios en el valor obtenido.

- (1.5 ptos.) $A + B$. A = 0x4A800000, B = 0x41700000.

Solución:

- A = 01001010100000000000000000000000b.
  - **Signo:** 0b → positivo.
  - **Exponente:** 10010101b → 149 - 127 = 22.
  - **Significante:** 00000000000000000000000b → 1,0b.
  - **Valor:** $(1,0 \times 10^{10110})$b = 4194304.
- B = 01000001011100000000000000000000b.
  - **Signo:** 0b → positivo.
  - **Exponente:** 10000010b → 130 - 127 = 3.
  - **Significante:** 11100000000000000000000b → 1,111b.
  - **Valor:** $(1,111 \times 10^{11})$b = 1111b = 15.

De lo anterior, se deduce que la operación a realizar es 4194304+15. 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 $0,00000000000000000111 \times 10^{10110}$. Luego, sumando los significantes, se tiene $A + B = 1,00000000000000000111 \times 10^{10110}$. Finalmente, si se almacena como `float`, se tiene el valor 01001010100000000000000000011110b = 0x4A80001E = 4194319. **No existe pérdida de precisión**.
Se otorgan **0.375 ptos.** por la desagregación e interpretación correcta de cada número; y **0.75 ptos.** por señalar correctamente el resultado de la representación en formato `float`.

- (1.5 ptos.) $A \times B$. A = 0x7F800000, B = 0x40000000.

Solución:

- A = 01111111100000000000000000000000b.
  - **Signo:** 0b → positivo.
  - **Exponente:** 11111111b → 255 - 127 = 128.
  - **Significante:** 00000000000000000000000b → 1,0b.
  - **Valor:** + ínf.
- B = 01000000000000000000000000000000b.
  - **Signo:** 0b → positivo.
  - **Exponente:** 10000000b → 128 - 127 = 1.
  - **Significante:** 00000000000000000000000b → 1,0b.
  - **Valor:** $1,0 \times 10^1 = 10\text{b} = 2$.

De lo anterior, se deduce que la operación a realizar es igual a $+\infty \times 2$, lo que en formato float se traduce como $+\infty$ = 0x7F800000. Se otorgan **0.375 ptos.** por la desagregación e interpretación correcta de cada número; y **0.75 ptos.** por señalar correctamente el resultado de la representación en formato `float`.

Pregunta 2: Comunicación con dispositivos I/O (6 ptos.)

(a) (1 pto.) Explique el significado y uso del identificador que envía el controlador de interrupciones luego de recibir la señal INTA por parte de la CPU.

> **Solución:** Al recibir la señal INTA, la CPU espera obtener el identificador asociado al **dispositivo cuya interrupción será atendida**. Este será enviado por el controlador de interrupciones. Luego, la CPU hace uso de este valor como índice del vector de interrupciones para obtener el puntero de la ISR del dispositivo atendido, permitiendo así la ejecución de su subrutina. Se otorgan **0.5 ptos.** por indicar la asociación entre el identificador y los dispositivos I/O; y **0.5 ptos.** por indicar su uso en la ejecución de la ISR deseada.

(b) (1 pto.) Indique cómo se lleva a cabo la transferencia de datos entre dispositivos I/O y la memoria RAM en arquitecturas que **carecen** de un controlador de DMA.

> **Solución:** En este caso se lleva a cabo *Programmed* I/O; *i.e.* la transferencia manual de los datos hacia los registros de la CPU, los que luego son transferidos a destino (ya sea la RAM o dispositivos I/O de almacenamiento). Se otorga **1 pto.** si se describe correctamente del rol de la CPU en este tipo de interacción. No se otorga puntaje parcial.

(c) (4 ptos.) Para evitar que las plantas de interior de su hogar se sequen, usted implementa un sistema de regado automatizado conectado a un mini-computador con ISA RISC-V. A cada una de sus plantas le añade un sensor de humedad y un regador, cuyos registros de 32 bits son mapeados en memoria desde la dirección 0xABADBABE:

Offset	Nombre	Descripción
0x00	`humidity_sensor_cactus`	Sensor humedad cactus
0x04	`humidity_sensor_monstera`	Sensor humedad monstera
0x08	`irrigator_status_cactus`	Registro estado regador cactus
0x0C	`irrigator_status_monstera`	Registro estado regador monstera
0x10	`irrigator_command_cactus`	Registro comando regador cactus
0x14	`irrigator_command_monstera`	Registro comando regador monstera

Nombre	Descripción	Valor
`humidity_sensor_*`	Porcentaje humedad	0-100
`irrigator_status_*`	Apagado	0
`irrigator_status_*`	Encendido	1
`irrigator_status_*`	Riego suave	2
`irrigator_status_*`	Riego medio	4
`irrigator_status_*`	Riego alto	8
`irrigator_command_*`	Encender	1
`irrigator_command_*`	Apagar	255
`irrigator_command_*`	Detener riego	128
`irrigator_command_*`	Regar (por defecto suave)	2
`irrigator_command_*`	Incrementar riego	3
`irrigator_command_*`	Decrementar riego	4

Luego de varias pruebas, usted desarrolla el siguiente programa para llevar a cabo el riego:

```
irrigation_system:
  li t0, 0xABADBABE            # Carga en el registro t0 el valor del literal 0xABADBABE
  li s0, 15
  li s1, 20
  li s2, 0
  li s3, 1
  step_one:
    lw t1, 8(t0)
    beq t1, s2, step_two
    bne t1, s3, step_three
    beq zero, zero, step_four                # Salto incondicional a step_four
  step_two:
    li t2, 1
    sw t2, 16(t0)
    beq zero, zero, step_four                # Salto incondicional a step_four
  step_three:
    li t2, 128
    sw t2, 16(t0)
  step_four:
    li t2, 2
    sw t2, 16(t0)
    li t2, 3
    sw t2, 16(t0)
    while_one:
        lw t3, 0(t0)
        blt t3, s0, while_one
    li t2, 4
    sw t2, 16(t0)
    while_two:
        lw t3, 0(t0)
        blt t3, s1, while_two
    li t2, 128
    sw t2, 16(t0)
    li t2, 255
    sw t2, 16(t0)
  end:                         # Término del programa.
```

En base a este programa, conteste las siguientes preguntas:

1. (1 pto.) ¿Qué modo de comunicación existe entre el computador y el sistema de riego?

Solución: El modo de comunicación existente consiste en *polling*, dado que en el *label* `step_four` se evidencia la consulta constante sobre el valor del sensor de humedad para gestionar el riego de la planta. No obstante lo anterior, también se aceptan las siguientes respuestas:

- Comunicación en base a interrupciones, si se justifica que el código anterior consiste en la ISR ejecutada por el computador para llevar a cabo el riego de una planta.
- *Memory mapped* I/O. Si bien esto alude a la forma en la que se accede a los datos de los registros asociados a los dispositivos, dada la ambigüedad del enunciado se acepta también como respuesta **siempre** que se señale como argumento una de dos opciones: (1) la única forma de acceso a dispositivos a RISC-V; (2) el acceso a los registros de los dispositivos con instrucciones de lectura y escritura de memoria.

En cualquiera de los tres casos, se otorga **1 pto.** siempre y cuando **exista una justificación donde se apliquen correctamente los conceptos de I/O**. No se otorgará puntaje parcial.

2. (3 ptos.) Describa, basándose en las tablas provistas, las tareas que realiza el programa en cada *label* `step_*`. Haga una descripción según los estados y comandos; no una explicación por línea de código.

Solución:

- `step_one`: Se revisa el estado del regador del cactus. Si está apagado, se ejecuta el código del *label* `step_two`. Si está regando, independiente de la intensidad, se ejecuta el código del *label* `step_three`. En cualquier otro caso (*i.e.* está encendido), se ejecuta el código del *label* `step_four`. Se otorgan **0.5 ptos.** si se describe correctamente; **0.25 ptos.** si existe como máximo un error; y no se otorga puntaje si existen dos o más errores.

- `step_two`: Se enciende el regador del cactus y se procede a ejecutar el código del *label* `step_four`. Se otorgan **0.5 ptos.** si se describe correctamente; **0.25 ptos.** si existe como máximo un error; y no se otorga puntaje si existen dos o más errores.

- `step_three`: Se detiene el riego del regador del cactus y se procede a ejecutar el código del *label* `step_four`. Se otorgan **0.5 ptos.** si se describe correctamente; **0.25 ptos.** si existe como máximo un error; y no se otorga puntaje si existen dos o más errores. En este caso, no es necesario explicitar la ejecución de `step_four`.

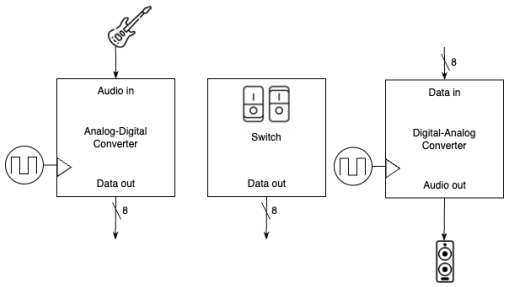
- `step_four`: Se activa el riego del regador del cactus y se incrementa para que realice riego medio. Luego, en `while_one` se consulta el valor del sensor de humedad del cactus constantemente hasta que sea mayor o igual a un 15 %. Posteriormente, se decrementa el nivel de riego para que tenga una intensidad suave. Finalmente, en `while_two` se consulta el valor del sensor de humedad del cactus hasta que sea mayor o igual a un 20 %, momento en el que se detiene el regador y se apaga. Se otorgan **1.5 ptos.** si se describe correctamente; y se descuentan **0.5 ptos.** por cada detalle faltante en la descripción.

Importante: Se considera como error en cada *step* si se toma en cuenta la monstera: en el código nunca se accede a los dispositivos conectados a ella.

Pregunta 3: Computador básico e I/O (6 ptos.)

Dado su interés por la música y la tecnología, ha empezado a explorar el funcionamiento de los pedales de efecto haciendo uso del computador básico para modificar las señales de sonido emitidas por una guitarra. Para esto, habilita la conexión con los siguientes dispositivos *port-mapped*:

Dispositivo	Tipo	Puerto
Conversor análogo-digital	Entrada	0
Switch de efecto	Entrada	1
Conversor digital-análogo	Salida	2



El conversor análogo-digital envía la digitalización de la señal generada por la guitarra; mientras que el conversor digital-análogo recibe la señal digital con efectos aplicados por el computador, los que son finalmente transmitidos a un amplificador. Adicionalmente, añade un *switch* con señal de salida de 8 bits, pero solo con dos valores posibles: 0 si está apagado y 1 si está prendido. El efecto del pedal se aplica solo si el *switch* está encendido. Para lograr este comportamiento, debe modificar la microarquitectura del computador básico implementando las siguientes instrucciones:

Instrucción	Operandos	Operación
`IN`	`A,Lit`	`A` = Dispositivo asociado a puerto `Lit`
	`B,Lit`	`B` = Dispositivo asociado a puerto `Lit`
`OUT`	`Lit,A`	Dispositivo asociado a puerto `Lit` = `A`
	`Lit,B`	Dispositivo asociado a puerto `Lit` = `B`

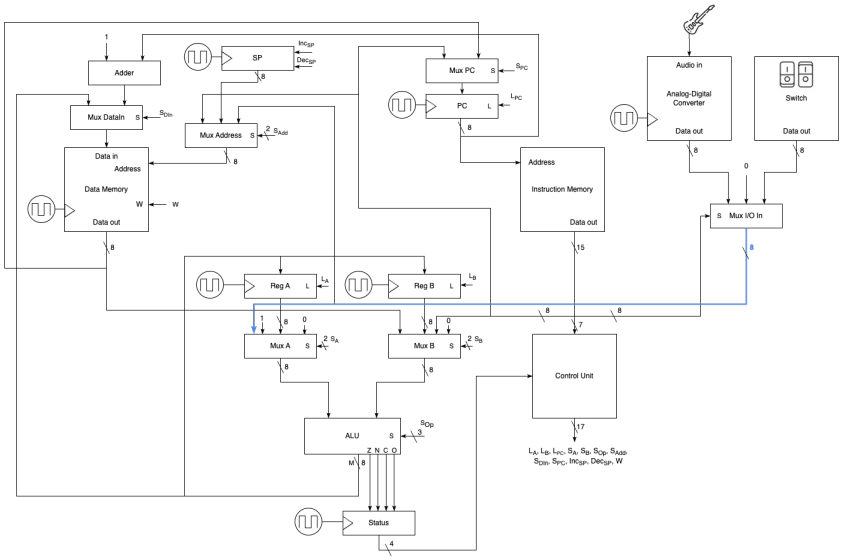
Para cada instrucción nueva, deberá incluir la combinación **completa** de señales que la ejecutan. Por cada señal de carga/escritura/incremento/decremento, deberá indicar si se activan (1) o no (0); en las señales de selección, deberá indicar el **nombre** de la entrada escogida ("-" si no afecta). Puede realizar todas las modificaciones en un solo diagrama. Para el conversor digital-análogo, asuma que si no se le envían datos mediante `OUT`, debe recibir un 0 por defecto.

(a) (2.5 ptos.) `IN Reg, Lit`. Guarda en `Reg` (`A`/`B`) el valor de salida del puerto `Lit`.

Solución: A continuación, se describe una posible implementación:

- Se agrega el multiplexor "`Mux I/O In`" con una señal de selección de 8 bits para escoger uno de varios dispositivos I/O. Si bien solo se conectarán dos, es necesario considerar que el puerto de 8 bits requiere funcionar con valores exactos. Para los puertos no definidos, se selecciona una señal constante igual a cero. Para la selección, se utiliza el literal proveniente de la memoria de instrucciones.
- Se conecta la salida del multiplexor anterior con la entrada libre del multiplexor "`Mux A`", lo que permite almacenar el valor de los dispositivos en cualquiera de los registros.

El siguiente diagrama muestra la implementación anterior:



Para ejecutar las instrucciones, 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}
`IN A, Lit`	1	0	0	0	0	0	`I/O`	`ZERO`	`ADD`	-	-	-
`IN B, Lit`	0	1	0	0	0	0	`I/O`	`ZERO`	`ADD`	-	-	-

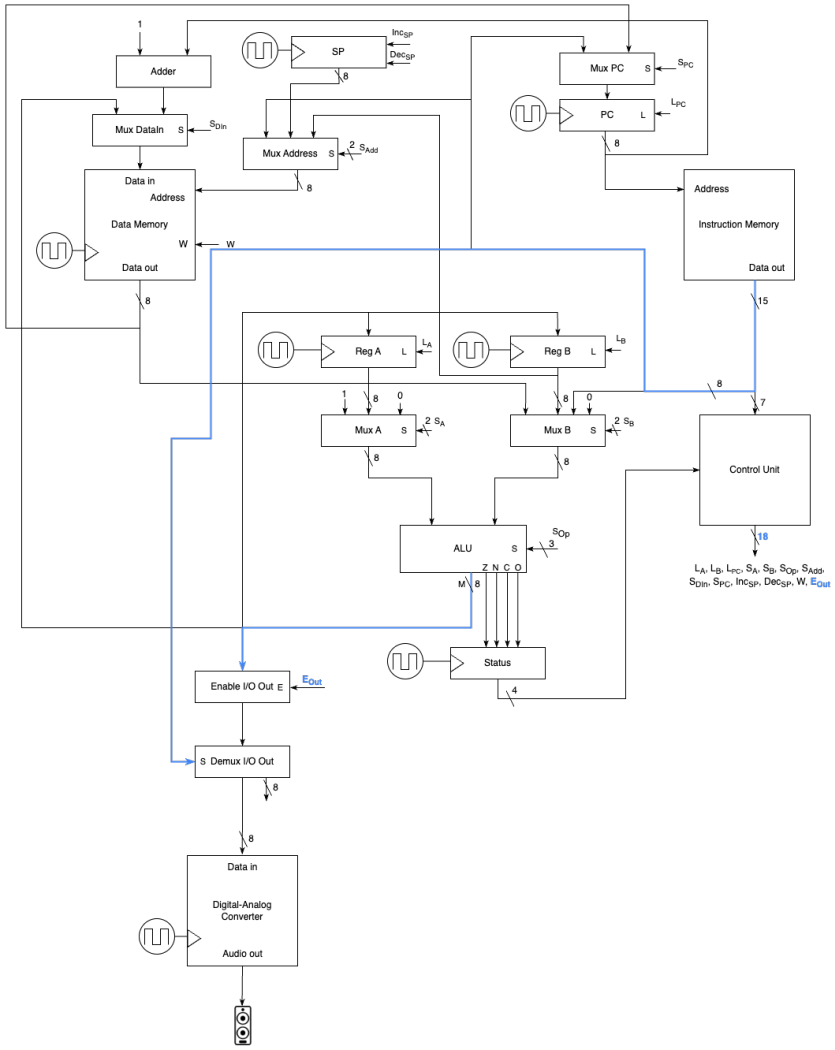
Se otorgan **1.25 ptos.** por la correctitud de la modificación y **1.25 ptos.** por entregar una combinación de señales correcta para la instrucción. En caso de existir como máximo un error de implementación en uno de los criterios antes descritos, se otorga la mitad del puntaje. Si existe más de un error, no se otorga puntaje.

(b) (2.5 ptos.) `OUT Lit, Reg`. Entrega como entrada al puerto `Lit` el valor de `Reg` (`A`/`B`).

Solución: A continuación, se describe una posible implementación:

- Se agrega el demultiplexor "`Demux I/O Out`" con una señal de selección de 8 bits para enviar el valor de entrada a uno de varios dispositivos I/O. Si bien solo habrá un dispositivo disponible para el envío, es necesario considerar que el puerto de 8 bits requiere funcionar con valores exactos. Para los puertos no definidos, simplemente se envía la señal a cables no conectados a otros dispositivos.
- La salida de la ALU se conecta con un *enabler* "`Enable I/O Out`", con señal habilitadora E_{Out} solo activa para las instrucciones de tipo `OUT`. Finalmente, su salida se conecta con la entrada del demultiplexor anterior para el envío correcto del valor de los registros `A` o `B`.

El siguiente diagrama muestra la implementación anterior:

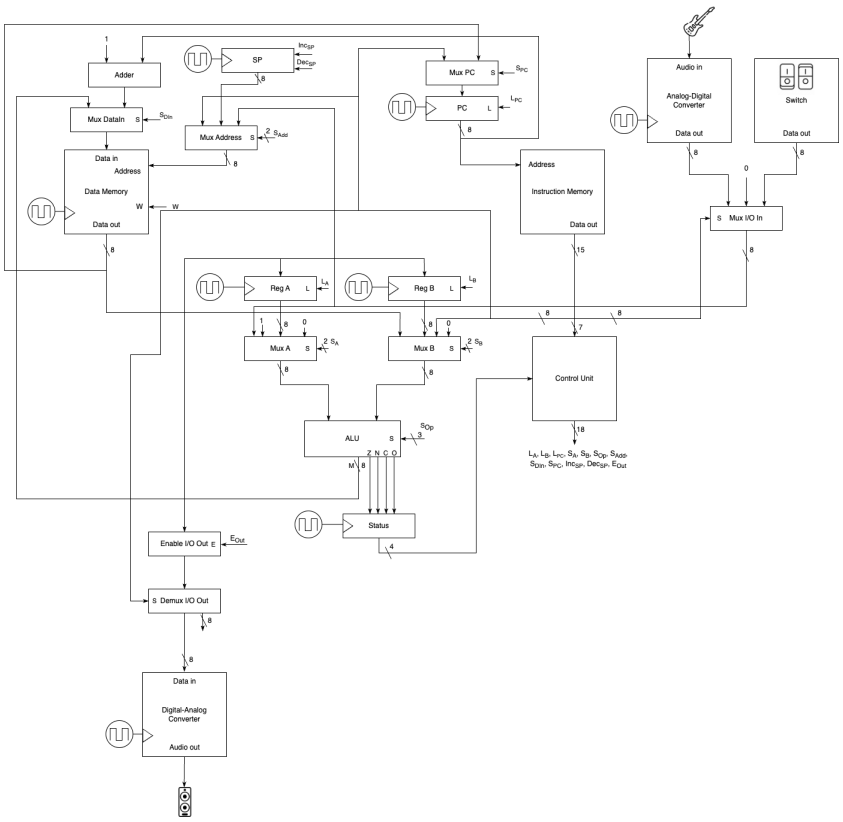


Para ejecutar las instrucciones, se utiliza la siguiente combinación de señales:

Instrucción	L_A	L_B	L_{PC}	W	Inc_{SP}	Dec_{SP}	S_A	S_B	S_{OP}	S_{Add}	S_{DIn}	S_{PC}	E_{Out}
`OUT Lit, A`	0	0	0	0	0	0	A	ZERO	ADD	-	-	-	1
`OUT Lit, B`	0	0	0	0	0	0	ZERO	B	ADD	-	-	-	1

Se otorgan **1.25 ptos.** por la correctitud de la modificación y **1.25 ptos.** por entregar una combinación de señales correcta para la instrucción. En caso de existir como máximo un error de implementación en uno de los criterios antes descritos, se otorga la mitad del puntaje. Si existe más de un error, no se otorga puntaje.

A modo de completitud, se comparte a continuación el diagrama completo con todas las instrucciones solicitadas.



Para ejecutar las instrucciones, 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}	E_{Out}
`IN A, Lit`	1	0	0	0	0	0	`I/O`	`ZERO`	`ADD`	-	-	-	0
`IN B, Lit`	0	1	0	0	0	0	`I/O`	`ZERO`	`ADD`	-	-	-	0
`OUT Lit, A`	0	0	0	0	0	0	`A`	`ZERO`	`ADD`	-	-	-	1
`OUT Lit, B`	0	0	0	0	0	0	`ZERO`	`B`	`ADD`	-	-	-	1

(c) (1 pto.) Asuma que se implementan las instrucciones anteriores. Elabore un fragmento de código Assembly que aplique el efecto del pedal a través del llamado a una subrutina `apply_effect`. Esta asumirá que la señal de sonido se encontrará almacenada en `B` y modificará su valor directamente. El programa debe, entonces:

1. Obtener el valor de la señal del conversor análogo-digital y el estado del *switch*.
2. Aplicar el efecto a partir del llamado a `apply_effect` **solo** si el *switch* está activado.
3. Enviar al conversor digital-análogo la señal resultante.

Debe hacer uso de las instrucciones `IN` y `OUT` para cumplir lo anterior. Puede agregar instrucciones adicionales ya existentes en la ISA del computador básico para completar el programa. **Solo debe llamar a `apply_effect`**, no es necesario que defina su código.

Solución: A continuación, se muestra un fragmento de código que cumple lo solicitado.

```
CODE:
  ; Paso 1: Obtención de la señal del conversor ADC y el switch.
  IN A,1
  IN B,0
  ; Paso 2: Aplicación del efecto solo si el switch está activo.
  CMP A,1
  JNE end
  CALL apply_effect
  ; Paso 3: Envío de la señal resultante a través de B, independiente
  ; de la aplicación del efecto.
  end:
    OUT 2,B
```

Se otorgan **0.25 ptos.** por obtener correctamente el valor de las señales de los dispositivos de entrada; **0.5 ptos.** por aplicar correctamente el efecto según el valor del *switch*; y **0.25 ptos.** por enviar correctamente el resultado al conversor digital-análogo.