



IIC2343 - Arquitectura de Computadores (II/2025)

Interrogación 2

Pauta de evaluación

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

- (a) (2 ptos.) Compare la representación de punto fijo con la representación de punto flotante en términos de rango y precisión. ¿Cuál es más conveniente en cada caso y por qué?

Solución: La representación de punto fijo es más conveniente si lo que se busca es **precisión**, dado que sus valores se distribuyen de manera uniforme en su rango gracias a la porción de bits de parte fraccional. No obstante, tiene un rango muy acotado que depende de los bits restantes para representar la parte entera del número. La representación de punto flotante, en cambio, es más conveniente si lo que se busca es un **rango mayor**, ya que el uso de bits para representar el exponente permite “flotar” el punto y representar valores mucho más grandes o pequeños. Hay que considerar, de todas formas, que mientras más grande o pequeño sea el valor a representar, su precisión disminuye significativamente al depender de los bits restantes para el significante. Se otorga **1 pto.** por describir correctamente en qué caso se prefiere una representación de punto fijo, y **1 pto.** por describir correctamente en qué caso se prefiere una representación de punto flotante.

- (b) (1 pto.) Suponga que imprime en la consola de su computador un valor que se encuentra almacenado en formato float del estándar IEEE754. Al imprimir, en la consola ve que aparece el valor “NaN”. ¿Es posible, a partir de este resultado, obtener la secuencia de 32 bits exacta que representa este valor? Justifique.

Solución: No es posible. La representación del valor NaN del formato float tiene las siguientes características:

- Su exponente es igual a 11111111.
- Al menos uno de sus bits de significante es distinto de cero.

El último punto es relevante. Dado que cualquiera de los bits del significante es distinto de cero, al leer en consola el valor “NaN” no tenemos cómo saber cuál de los 23 bits es igual a 1. Se otorgan **0.5 ptos.** por responder correctamente, y **0.5 ptos.** por justificación correcta. Si solo responde que no es posible sin ninguna justificación, no se otorga puntaje.

- (c) (3 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 el valor del resultado de la operación y señalar si se pierde precisión al almacenarlo de vuelta como float del estándar IEEE754, indicando además cómo queda almacenado en este formato. Los valores deben presentarse en base binaria con notación científica normalizada.

- (1.5 ptos.) $A + B$. $A = 0x40400000$, $B = 0x80000000$.

Solución:

- $A = 0\ 10000000\ 10000000000000000000000b$.
 - **Signo:** $0b \rightarrow$ positivo.
 - **Exponente:** $10000000b \rightarrow 128 - 127 = 1$.
 - **Significante:** $10000000000000000000000b \rightarrow 1,1b$.
 - **Valor:** $(1,1 \times 10^1)b = 11b = 3$.
- $B = 1\ 00000000\ 00000000000000000000000b$.
 - **Signo:** $1b \rightarrow$ negativo.
 - **Exponente:** $00000000b \rightarrow$ reservado para el cero.
 - **Significante:** $00000000000000000000000b \rightarrow 1,0b$.
 - **Valor:** -0 (valor reservado).

De lo anterior, se deduce que la operación a realizar es $3 + (-0)$. En este caso, no se evaluará la forma en la que se realice la suma; solo que su resultado sea correcto. No obstante, **no es necesario realizarla para deducir que el resultado es igual a 3** y, por ende, no hay pérdida de precisión al almacenar el valor de vuelta como float.

Se otorgan **0.375 ptos.** por la interpretación correcta de A ; **0.375 ptos.** por la interpretación correcta de B ; **0.375 ptos.** por señalar el resultado correcto de la operación; y **0.375 ptos.** por indicar correctamente si se pierde precisión o no. Además, en este caso se acepta no realizar la operación si se deduce que la suma es por 0, manteniendo el valor final de A .

- (1.5 ptos.) $A \times B$. $A = 0x40000000$, $B = 0xC0A00000$.

Solución:

- $A = 0\ 10000000\ 0000000000000000000000b$.
 - **Signo:** $0b \rightarrow$ positivo.
 - **Exponente:** $10000000b \rightarrow 128 - 127 = 1$.
 - **Significante:** $00000000000000000000000b \rightarrow 1,0b$.
 - **Valor:** $(1,0 \times 10^1)b = 10b = 2$.
- $B = 1\ 10000001\ 01000000000000000000000b$.
 - **Signo:** $1b \rightarrow$ negativo.
 - **Exponente:** $10000001b \rightarrow 129 - 127 = 2$.
 - **Significante:** $010000000000000000000000b \rightarrow 1,01b$.
 - **Valor:** $-(1,01 \times 10^2)b = -101b = -5$.

De lo anterior, se deduce que la operación a realizar es 2×-5 . 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 $1 + 2 = 3 = 11b$. Luego, se multiplican los significantes: $1,0 \times -1,01 = -1,01$. Finalmente, se tiene como resultado $-1,01 \times 10^{11}b = -1010b = -10$, que en formato float se almacena como $1100000100100000000000000000000b = 0xC1200000$. **No existe pérdida de precisión.**

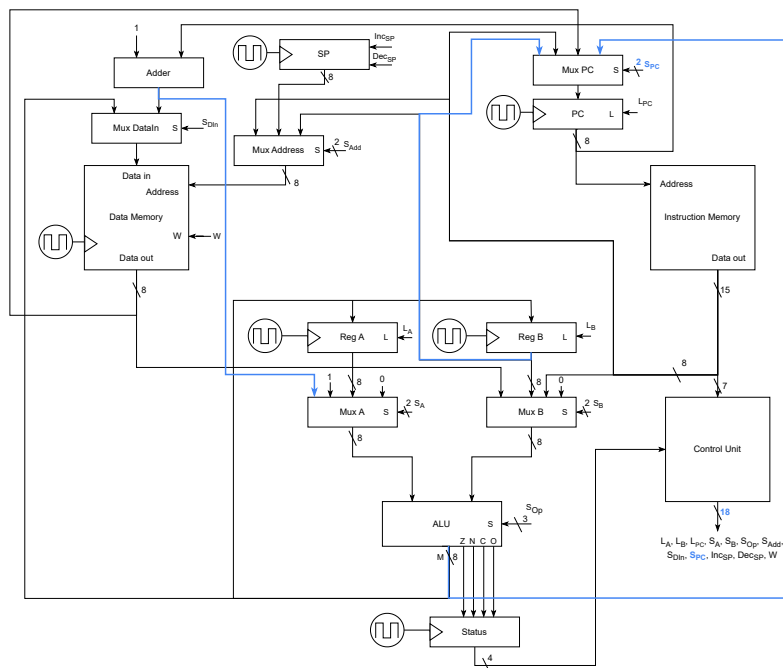
Se otorgan **0.375 ptos.** por la interpretación correcta de A ; **0.375 ptos.** por la interpretación correcta de B ; **0.375 ptos.** por señalar el resultado correcto de la operación; y **0.375 ptos.** por indicar correctamente si se pierde precisión o no. Se acepta escribir el valor en potencias de 2.

Pregunta 2: Computador básico con subrutinas (6 pts.)

En los ítems **(a)** y **(b)**, se le describirán instrucciones que deberá implementar en el computador básico con subrutinas 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.**

- (0.75 ptos.) **CALL A,B**. Almacena en el registro A la dirección de retorno PC+1 y salta a la dirección de memoria almacenada en el registro B.
- (0.75 ptos.) **CALL B,Dir**. Almacena en el registro B la dirección de retorno PC+1 y salta a la dirección de memoria equivalente al literal **Dir**.
- (0.75 ptos.) **RET A**. Salta a la dirección de retorno almacenada en el registro A.
- (0.75 ptos.) **RET B**. Salta a la dirección de retorno almacenada en el registro B.

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 conecta la salida del *Adder* que computa $PC+1$ con la entrada restante del componente **Mux A**, de forma que se pueda tener este valor como salida de la ALU para su almacenamiento en los registros A o B.
- Se conecta la salida de la ALU con una de las entradas del componente **Mux PC**, de forma que se pueda seleccionar el valor del registro A o del registro B como direcciones de memoria a saltar.
- Se conecta la salida del registro B con una de las entradas del componente **Mux PC**, de forma que se pueda seleccionar el valor de este como dirección de memoria a saltar mientras que al mismo tiempo se compute $PC+1$ desde la ALU.
- Finalmente, incrementa a 2 bits la señal S_{PC} para poder soportar las dos nuevas entradas del multiplexor **Mux PC**.

A continuación, la tabla de señales resultante:

Instrucción	L_A	L_B	L_{PC}	Inc_{SP}	Dec_{SP}	W	S_A	S_B	S_{OP}	S_{Add}	S_{DIn}	S_{PC}
CALL A,B	1	0	1	0	0	0	PC+1	ZERO	ADD	-	-	B
CALL B,Dir	0	1	1	0	0	0	PC+1	ZERO	ADD	-	-	LIT
RET A	0	0	1	0	0	0	A	ZERO	ADD	-	-	ALU
RET B	0	0	1	0	0	0	ZERO	B	ADD	-	-	ALU

Por cada instrucción, se otorgan:

- **0.375 ptos.** por *hardware* correctamente modificado; **0.25 ptos.** si existe **un solo error** de implementación; **0.1 ptos.** si existen **solo dos errores**; y **0 ptos.** en otro caso.
- **0.375 ptos.** por entregar una combinación de señales correcta; **0.25 ptos.** si existe **un solo error**; **0.1 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.

- (e) (3 ptos.) A continuación, se adjuntan dos fragmentos de código (1) y (2) escritos en el Assembly del computador básico, 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 12
  var2 4
CODE:
  MOV A,5 ; Dir. memoria 0x00
  MOV B,3 ; Dir. memoria 0x01
  RET ; Dir. memoria 0x02-0x03
case_1:
  ADD A,B ; Dir. memoria 0x04
  PUSH A ; Dir. memoria 0x05
  MOV A,B ; Dir. memoria 0x06
  SHL A,A ; Dir. memoria 0x07
  MOV B,A ; Dir. memoria 0x08
  POP A ; Dir. memoria 0x09-0x0A
  JMP end ; Dir. memoria 0x0B
case_2:
  SUB A,B ; Dir. memoria 0x0C
  PUSH A ; Dir. memoria 0x0D
  MOV A,B ; Dir. memoria 0x0E
  SHR A,A ; Dir. memoria 0x0F
  MOV B,A ; Dir. memoria 0x10
  POP A ; Dir. memoria 0x11-0x12
end:
```

(2)

```
DATA:
  n 2
  res 0
CODE:
  MOV A,(n) ; Dir. memoria 0x00
  CALL B,fun ; Dir. memoria 0x01
  JMP end ; Dir. memoria 0x02
fun:
  CMP A,0 ; Dir. memoria 0x03
  JLE fun_end ; Dir. memoria 0x04
  PUSH B ; Dir. memoria 0x05
  MOV B,A ; Dir. memoria 0x06
  ADD A,(res) ; Dir. memoria 0x07
  MOV (res),A ; Dir. memoria 0x08
  MOV A,B ; Dir. memoria 0x09
  SUB A,1 ; Dir. memoria 0x0A
  CALL B,fun ; Dir. memoria 0x0B
  POP B ; Dir. memoria 0x0C-0x0D
fun_end:
  RET B ; Dir. memoria 0x0E
end:
  MOV A,(res) ; Dir. memoria 0x0F
```

Solución: El fragmento de código (1) accede, por *overflow* del registro SP, a la dirección de memoria 0 (equivalente a `var1`) para cargar su valor en el registro PC, por lo que la ejecución de `RET` genera el salto al *label* `case_2`. El fragmento de código (2), por otra parte, computa de forma recursiva la suma de la variable `n` con todos sus antecesores hasta llegar a cero, almacenando el resultado final en `res`. A continuación, se muestra el resultado de la ejecución para ambos fragmentos:

(1)

```
DATA:
var1 12      ; MEM[0] = 12
var2 4       ; MEM[1] = 4
CODE:
MOV A,5      ; A = 5
MOV B,3      ; B = 3
RET          ; SP++ => 255 + 1 = 0, PC = MEM[0] = 12
case_1:
  ADD A,B    ; NO SE EJECUTA
  PUSH A     ; NO SE EJECUTA
  MOV A,B    ; NO SE EJECUTA
  SHL A,A    ; NO SE EJECUTA
  MOV B,A    ; NO SE EJECUTA
  POP A      ; NO SE EJECUTA
  JMP end    ; NO SE EJECUTA
case_2:
  SUB A,B    ; A = 2
  PUSH A     ; MEM[SP] = A => MEM[0] = 2, SP-- => 0 - 1 = 255
  MOV A,B    ; A = 3
  SHR A,A    ; A = 1
  MOV B,A    ; B = 1
  POP A      ; SP++ => 255 + 1 = 0, A = MEM[0] = 2
end: ; A = 2, B = 1
```

(2)

```
DATA:
n 2          ; MEM[0] = 2
res 0        ; MEM[1] = 0
CODE:
MOV A,(n)    ; A = n = 2
CALL B,fun   ; B = 2, fun(n)
JMP end
fun:
  CMP A,0
  JLE fun_end ; Termina cuando A = 0
  PUSH B      ; Respalda B en stack (retorno)
  MOV B,A     ; B = A = n
  ADD A,(res) ; A += res
  MOV (res),A ; res = res + n
  MOV A,B     ; A = n
  SUB A,1     ; n--
  CALL B,fun   ; fun(n-1)
  POP B       ; Recupera B (retorno)
fun_end:
  RET B       ; Retorna de la forma esperada
end:
; A = res = fun(2) = 2+fun(1) = 2+1+fun(0) = 2+1+0 = 3
MOV A,(res) ; B = Primer retorno computado = 2
```

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.

Pregunta 3: Comunicación con dispositivos I/O (6 ptos.)

- (a) (1 pto.) Señale una ventaja y una desventaja de mapear los dispositivos I/O de un computador en memoria.

Solución: Una ventaja es que no implica modificaciones en la ISA, lo que permite comunicarse directamente con los dispositivos a través de instrucciones de lectura y escritura de memoria. Una desventaja, en cambio, es el uso significativo de direcciones de memoria a reservar, que en memorias más pequeñas puede implicar problemas de ejecución en los programas (*memory barrier*). Se otorgan **0.5 ptos.** por describir correctamente una ventaja y **0.5 ptos.** por describir correctamente una desventaja.

- (b) (1 pto.) Explique el uso e importancia de las ISRs ejecutadas por la CPU en el contexto de comunicación vía interrupciones.

Solución: Las ISRs o *Interrupt Service Routines* son subrutinas asociadas al manejo que requieren los dispositivos por parte de la CPU una vez que son atendidas por la CPU. Son importantes ya que, a través de su ejecución, se lleva a cabo la comunicación entre la CPU y los dispositivos. Se otorgan **0.5 ptos.** por describir correctamente en qué consiste una ISR y **0.5 ptos.** por destacar su importancia.

- (c) (4 ptos.) Suponga que decide automatizar la disponibilidad de un baño mediante sensores y un emisor Bluetooth conectados a un computador con arquitectura RISC-V. Para la comunicación con los sensores y el emisor, sus registros de 32 bits están mapeados en memoria desde la dirección 0x600DD0D0:

Offset	Nombre	Descripción	Nombre	Descripción	Valor
0x00	door_sensor_status	Registro estado sensor de puerta	door_sensor_status	Puerta cerrada	0
0x04	light_sensor_status	Registro estado sensor de luz	door_sensor_status	Puerta abierta	1
0x08	notifier_command	Registro comando emisor	light_sensor_status	Luz apagada	0
0x0C	notifier_status	Registro estado emisor	light_sensor_status	Luz encendida	1
			notifier_status	Inactivo	0
			notifier_status	Activo	1
			notifier_status	Enviando	2
			notifier_command	Encender emisor	1
			notifier_command	Notificar OPEN	2
			notifier_command	Notificar CLOSED	3
			notifier_command	Apagar emisor	4

El sistema debe enviar una notificación Bluetooth según la disponibilidad del baño: **OPEN** si es que está disponible, y **CLOSED** si es que está ocupado según los estados de los sensores de puerta (`door_sensor_status`) y luz (`light_sensor_status`). Para ello, se desarrolla el siguiente programa:

```

notifier_automation:
    # Carga zero + 0x600DD0D0 en t0
    addi t0, zero, 0x600DD0D0
    addi s0, zero, 0
    addi s1, zero, 1
    addi s2, zero, 2
    addi s3, zero, 3
    addi s4, zero, 4

step_one:
    lw t3, 12(t0) # Carga Mem[t0 + 12] en t3
    # Salta a step_two si t3 != s0
    bne t3, s0, step_two
    sw s1, 8(t0) # Carga s1 en Mem[t0 + 8]

step_two:
    lw t1, 0(t0)
    lw t2, 4(t0)
    bne t1, s1, step_three
    bne t2, s0, step_three
    # Salto incondicional a step_four
    jal zero, step_four

```

```

# => Continuación

step_three:
    sw s3, 8(t0)

    while_step_three:
        lw t3, 12(t0)
        # Salta a while_step_three si t3 == s2
        beq t3, s2, while_step_three

    sw s4, 8(t0)
    jal zero, step_one

step_four:
    sw s2, 8(t0)

    while_step_four:
        lw t3, 12(t0)
        beq t3, s2, while_step_four

    sw s4, 8(t0)
    jal zero, step_one

```

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 cada instrucción del programa.

Solución:

- **step_one (1 pto.):** Se revisa el estado del emisor. Si el emisor está inactivo, se enciende y continúa a **step_two**. En otro caso, salta directamente a **step_two**.
- **step_two (1 pto.):** Se revisa el estado del sensor de puerta y el sensor de luz del baño. Si la puerta está cerrada o la luz está encendida, salta a **step_three**. En otro caso, salta directamente a **step_four**.
- **step_three (1 pto.):** A través del emisor, se envía la notificación CLOSED. Luego, se espera a que termine el envío y se apaga el emisor. Finalmente, salta de vuelta al **step_one**.
- **step_four (1 pto.):** A través del emisor, se envía la notificación OPEN. Luego, se espera a que termine el envío y se apaga el emisor. Finalmente, salta de vuelta al **step_one**.

Para cada paso, se descuenta un 25% del puntaje si existe como máximo un error o detalle faltante; un 50% si existen como máximo dos errores; y un 100% en otro caso.