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

IIC2343 - Arquitectura de Computadores (II/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 0,1101101b. 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 con la siguiente distribución: 1 bit de signo de significante; 5 bits de significante; 1 bit de signo de exponente; y 1 bit de exponente. ¿Cuál de estas representaciones presenta una menor pérdida de precisión para almacenar el número? Justifique su respuesta.

Nota: Durante la evaluación, se aclara que en punto flotante se usa la representación con bit explícito, no implícito.

Solución: Para este número, es recomendable hacer uso de la representación de **punto flotante**. Su representación normalizada equivale a $1,101101 \times 10^{-1}$, por lo que **con bit explícito** se almacena la siguiente secuencia: $0_{\text{Signo sig.}}11011_{\text{Significante}}1_{\text{Signo exp.}}1_{\text{Exp.}}$. De este resultado, se deduce que se pierden los dos bits menos significativos, equivalente a una pérdida de precisión de 2^{-7}. En cambio, en el formato de punto fijo, se almacena la siguiente secuencia: $0_{\text{Signo}}000_{\text{Entero}}1101_{\text{Fracción}}$. Se observa que en esta última representación se pierden los 3 bits menos significativos, por lo que su resultado es menos preciso al tener una pérdida de precisión de $2^{-5} + 2^{-7}$. 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 valor o bits perdidos en cada caso.

(b) (1.5 ptos.) Indique el valor que representa el float 0x7FC00010. Justifique su respuesta en base al estándar IEEE754.

Solución: Como el exponente desfasado del número es 255 (*i.e.* todos sus bits iguales a 1), que se reserva para ciertas representaciones, su valor puede ser $\pm\infty$ o `NaN` (*Not a Number*). Luego, su significante posee al menos dos bits distintos de cero, por lo que su valor equivale a `NaN`. Se otorgan **0.75 ptos.** por señalar correctamente el valor que representa el float; y **0.75 ptos.** por justificación en función de su exponente y significante.

(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 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.

- (1.5 ptos.) $A + B$. $A =$ 0x40400000, $B =$ 0x3F800000.

Solución:

- $A =$0100000001000000 00000000000000000b.
  - **Signo:** 0b $\rightarrow$ positivo.
  - **Exponente:** 10000000b $\rightarrow$ 128 - 127 = 1.
  - **Significante:** 10000000000000000000000b $\rightarrow$ 1,1b.
  - **Valor:** $(1,1 \times 10^1)$b = 11b = 3.
- $B =$ 00111111100000000000000000000000.
  - **Signo:** 0b $\rightarrow$ positivo.
  - **Exponente:** 01111111b $\rightarrow$ 127 - 127 = 0.
  - **Significante:** 00000000000000000000000b $\rightarrow$ 1,0b.
  - **Valor:** $(1,0 \times 10^0)$b = 1b = 1.

De lo anterior, se deduce que la operación a realizar es 3 + 1. 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 $0,1 \times 10^1$. Luego, sumando los significantes, se tiene $A + B = (10,0 \times 10^1)$b = $(1,0 \times 10^{10})$. Finalmente, si se almacena como `float`, se tiene el valor 01000000100000000000000000000000b = 0x40800000 = 4. **No existe pérdida de precisión**.

Se otorgan **0.375 ptos.** por la interpretación correcta de cada número; **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.

- (1.5 ptos.) $A \times B$. A = 0x41820000, B = 0x3F200000.

Solución:

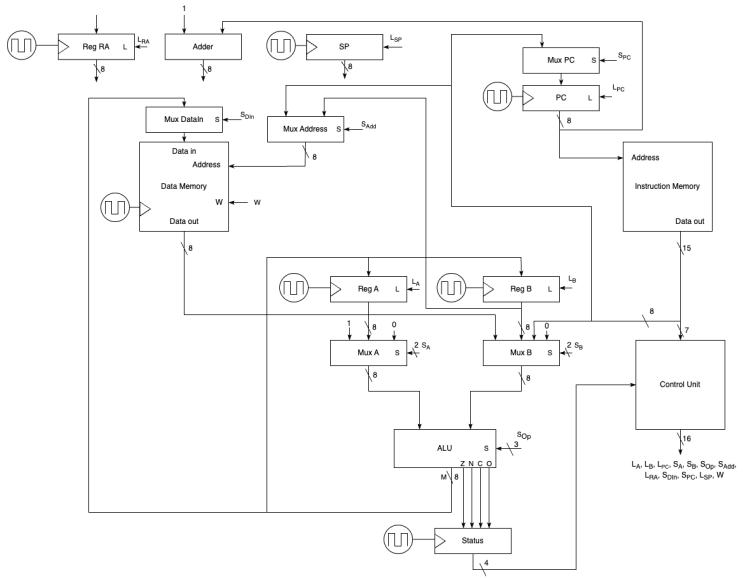
- A = 01000001100000100000000000000000b.
  - **Signo:** 0b → positivo.
  - **Exponente:** 10000011b → 131 - 127 = 4.
  - **Significante:** 00000100000000000000000b → 1,000001b.
  - **Valor:** $(1,000001 \times 10^{100})$b = 10000,01b = 16,25.
- B = 00111111001000000000000000000000b.
  - **Signo:** 0b → positivo.
  - **Exponente:** 01111110b → 126 - 127 = -1.
  - **Significante:** 01000000000000000000000b → 1,01b.
  - **Valor:** $1,01 \times 10^{-1} = 0,101$b = 0,625.

De lo anterior, se deduce que la operación a realizar es igual a $16,25 \times 0,625$. 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 suman los exponentes sin desfase y se llega al valor $-1 + 4 = 3 = 11$b. Luego, se multiplican los significantes: $1,01 \times 1,000001 = (101 \times 10^{-2}) \times (1000001 \times 10^{-6}) = (101 \times 1000001) \times 10^{-8} = 101000101 \times 10^{-8} = 1,01000101$. Finalmente, se tiene como resultado $1,01000101 \times 10^{11}$b = 1010,00101 = 10,15625, que en formato float se almacena como 01000001001000101000000000000000b = 0x41228000. **No existe pérdida de precisión**.

Se otorgan **0.375 ptos.** por la interpretación correcta de cada número; **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 ptos.)

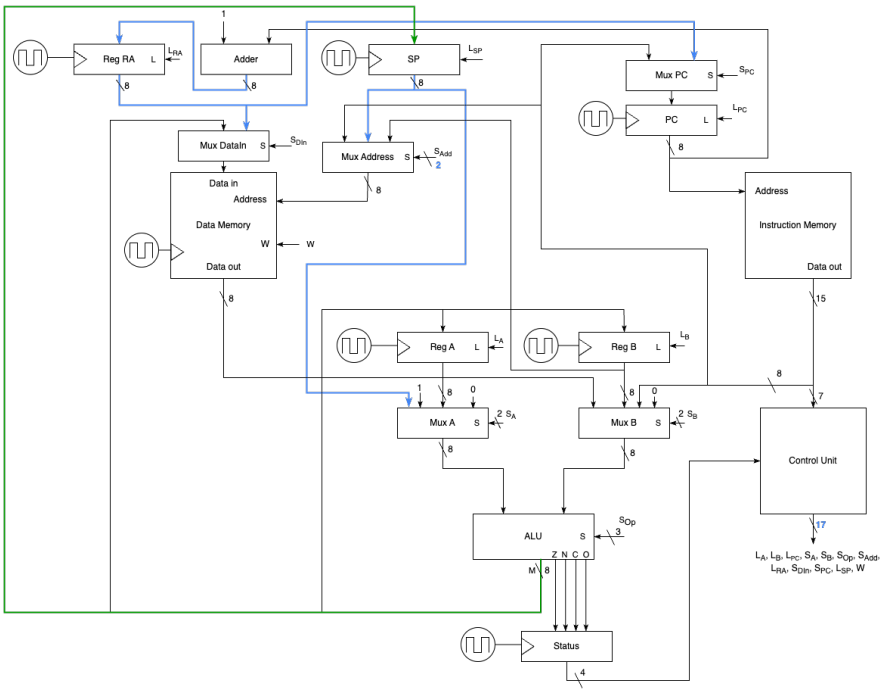
En arquitecturas RISC, las direcciones de retorno de una subrutina se almacenan en un **registro de enlace** (`RA`), a diferencia del computador básico donde se guardan en el *stack*. Asimismo, estas arquitecturas no poseen `PUSH/POP`, sino que acceden al *stack* directamente con el *stack pointer* (`SP`). Deberá contestar las preguntas de esta sección a partir de este contexto y el siguiente diagrama:



(a) **(5 ptos.)** A partir del diagrama de base adjunto, y asumiendo que se eliminan las instrucciones `CALL`, `RET`, `PUSH` y `POP`, realice las modificaciones de *hardware* necesarias e indique la combinación de señales completa para ejecutar las siguientes instrucciones en un ciclo:

- **(0.5 ptos.)** `MOV A,(SP)`. Guarda en `A` el valor `Mem[SP]`.
- **(0.5 ptos.)** `MOV (SP), A`. Guarda en `Mem[SP]` el valor `A`.
- **(1 pto.)** `ADD SP,Lit`. Guarda en `SP` el valor `SP + Lit`. `Lit` puede ser negativo.
- **(1 pto.)** `MOV (SP), RA`. Guarda en `Mem[SP]` el valor `RA`.
- **(1 pto.)** `JAL RA, label`. Guarda `PC+1` en `RA` y salta a la dirección asociada a `label`.
- **(1 pto.)** `JALR RA`. Salta a la dirección almacenada en `RA`.

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.



- La conexión entre `SP` y `Mux Address` permite el uso de `SP` como dirección de memoria con la señal S_{Add}, la que ahora debe ser de 2 bits.
- La conexión entre `SP` y `Mux A` permite su uso como primer operando en la `ALU`.
- La conexión entre la salida de la `ALU` y `SP` permite cargar en este último los resultados de las operaciones con la señal L_{SP}.
- La conexión entre `RA` y `Mux DataIn` permite la escritura de `RA` en memoria.
- La conexión entre `RA` y `PC+1` permite escribir este valor en `RA` con la señal L_{RA}.
- La conexión entre `RA` y `Mux PC` permite realizar saltos hacia direcciones almacenadas en `RA`.

A continuación, la tabla de señales resultante:

Instrucción	L_A	L_B	L_{PC}	L_{SP}	L_{RA}	W	S_A	S_B	S_{OP}	S_{Add}	S_{DIn}	S_{PC}
`MOV A, (SP)`	1	0	0	0	0	0	ZERO	DOUT	ADD	SP	-	-
`MOV (SP), A`	0	0	0	0	0	1	A	ZERO	ADD	SP	ALU	-
`ADD SP, Lit`	0	0	0	1	0	0	SP	LIT	ADD	-	-	-
`MOV (SP), RA`	0	0	0	0	0	1	-	-	-	SP	RA	-
`JAL RA, label`	0	0	1	0	1	0	-	-	-	-	-	LIT
`JALR RA`	0	0	1	0	0	0	-	-	-	-	-	RA

Por cada instrucción, se otorga **la mitad** del puntaje por *hardware* y **la otra mitad** por la combinación de señales. En cada criterio, se descuenta la mitad del puntaje en caso de existir, como máximo, un error de implementación o valor en la combinación de señales. En otro caso, no se otorga puntaje.

(b) **(1 pto.)** Modifique el siguiente fragmento de código para que haga uso exclusivo de las instrucciones anteriores. Debe mantener su funcionamiento original y terminar sin errores.

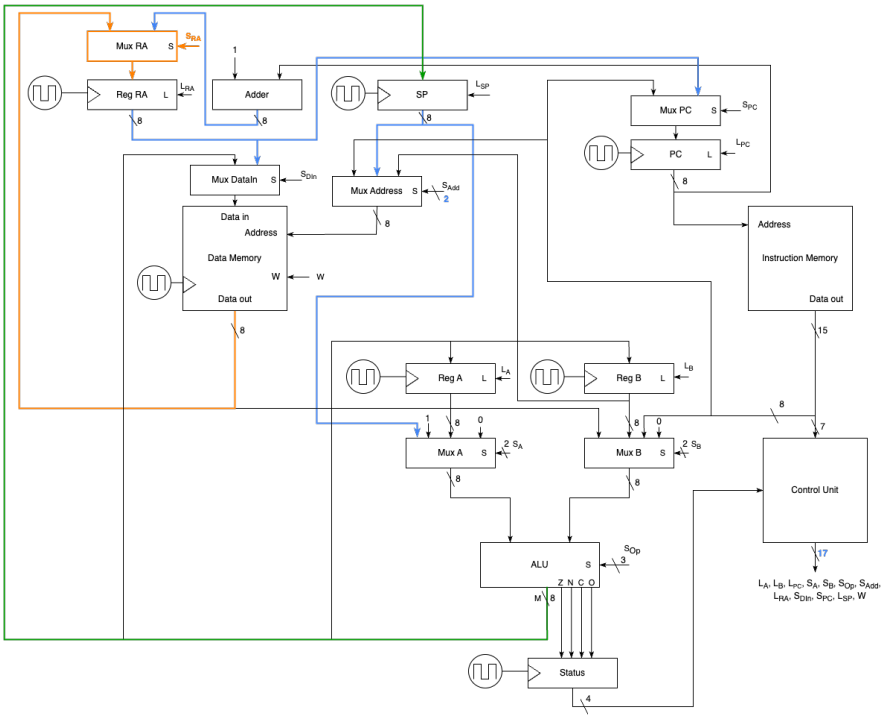
```
MOV A,1
PUSH A
CALL func
POP A
JMP end
func:
  SHL A,A
  CMP A,8
  JEQ end_func
  CALL func
  end_func:
    RET
end:
```

Solución: Es importante señalar que, con las instrucciones entregadas, **no es posible** entregar un fragmento de código que mantenga el funcionamiento original **sin errores**. Esto se debe al hecho de que no existe una instrucción que permita recuperar una dirección de retorno respaldada en memoria (por ejemplo, `MOV RA, (SP)`. Por este motivo, se consideran correctas respuestas que solo caigan en ese caso. A continuación, una solución posible.

```
MOV A,1
MOV (SP),A        ; Reemplazo por PUSH A
ADD SP,-1         ; Reemplazo por PUSH A
JAL RA,func       ; Reemplazo por CALL func
ADD SP,1          ; Reemplazo por POP A
MOV A,(SP)        ; Reemplazo por POP A
JMP end
func:
  SHL A,A
  CMP A,8
  JEQ end_func
  MOV (SP),RA     ; Respaldo de RA para evitar error en llamado recursivo
  ADD SP,-1       ; Respaldo de RA para evitar error en llamado recursivo
  JAL RA,func     ; Reemplazo por CALL func
  ADD SP,1        ; De todas formas, se restituye el valor de SP
  MOV RA,(SP)     ; Si existiera, podríamos recuperar el valor respaldado de RA
  end_func:
    JALR RA       ; Reemplazo por RET
end:
```

Se otorgan **0.25 ptos.** por reemplazar correctamente la instrucción `PUSH A`; **0.25 ptos.** por reemplazar correctamente la instrucción `CALL label`; **0.25 ptos.** por reemplazar correctamente la instrucción `POP A`; y **0.25 ptos.** por reemplazar correctamente la instrucción `RET`. Adicionalmente, si la respuesta alude a la falta de una instrucción para recuperar la dirección de retorno; o bien se usa de forma hipotética una instrucción que permita resolver el problema, **se otorgarán 0.5 ptos. de bonus al total**.

A modo de completitud, se incluye tanto el diagrama como la tabla de señales de la arquitectura que habilita la ejecución de la instrucción `MOV RA,(SP)`.



Instrucción	L_A	L_B	L_{PC}	L_{SP}	L_{RA}	W	S_A	S_B	S_{OP}	S_{Add}	S_{DIn}	S_{PC}	S_{RA}
`MOV A, (SP)`	1	0	0	0	0	0	ZERO	DOUT	ADD	SP	-	-	-
`MOV (SP), A`	0	0	0	0	0	1	A	ZERO	ADD	SP	ALU	-	-
`ADD SP, Lit`	0	0	0	1	0	0	SP	LIT	ADD	-	-	-	-
`MOV (SP), RA`	0	0	0	0	0	1	-	-	-	SP	RA	-	-
`MOV RA, (SP)`	0	0	0	0	1	0	-	-	-	SP	-	-	DOUT
`JAL RA, label`	0	0	1	0	1	0	-	-	-	-	-	LIT	PC+1
`JALR RA`	0	0	1	0	0	0	-	-	-	-	-	RA	-

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

(a) (1 pto.) Si un computador ya tiene dispositivos *memory mapped* a través de un *address decoder*, ¿se necesita *hardware* adicional para habilitar la comunicación vía *polling*? Justifique.

> **Solución: No**, no es necesario ya que en la comunicación vía *polling* es la CPU la que se encarga de interactuar con los dispositivos a través de la revisión de sus estados, lo que ya puede realizar con el acceso directo a las direcciones de memoria asociadas a ellos. Se otorgan **0.5 ptos.** por responder correctamente; y **0.5 ptos.** por justificación.

(b) (1 pto.) ¿Cuál es la función del vector de interrupciones? ¿Cuál es su contenido?

> **Solución:** La función del vector de interrupciones es permitir la ejecución correcta de la ISR (*Interruption Service Routine*) a partir del identificador del dispositivo a ser atendido, lo que es gestionado por el controlador de interrupciones. Este vector contiene los **punteros** de las ISR, por lo que si se desea atender al dispositivo `i`, se ejecuta el llamado de la subrutina a partir de su dirección igual a `vector_interrupciones[i]`. Se otorgan **0.5 ptos.** por señalar correctamente la función del vector de interrupciones; y **0.5 ptos.** por indicar de forma correcta su contenido.

(c) (4 ptos.) Suponga que decide, con el propósito de ahorrar tiempo, automatizar la preparación de café filtrado a partir de un molinillo y hervidor eléctricos conectados a un computador con ISA RISC-V. Para la comunicación con los electrodomésticos, sus registros de 32 bits son mapeados en memoria desde la dirección 0x600DCAFE:

Offset	**Nombre**	**Descripción**
0x00	`coffee_grinder_status`	Registro estado molinillo
0x04	`coffee_grinder_command`	Registro comando molinillo
0x08	`coffee_grinder_grind_size`	Config. grosor molienda
0x0C	`coffee_grinder_weight`	Config. gramos a moler
0x10	`kettle_status`	Registro estado hervidor
0x14	`kettle_command`	Registro comando hervidor
0x18	`kettle_temperature`	Config. temperatura hervidor (ºC)

Nombre	**Descripción**	**Valor**
`coffee_grinder_status`	Apagado	0
`coffee_grinder_status`	Encendido	1
`coffee_grinder_status`	Moliendo	2
`kettle_status`	Apagado	0
`kettle_status`	Encendido	1
`kettle_status`	Hirviendo	2
`coffee_grinder_command`	Encender	1
`coffee_grinder_command`	Apagar	255
`coffee_grinder_command`	Moler	128
`kettle_command`	Encender	1
`kettle_command`	Apagar	255
`kettle_command`	Hervir	128
`coffee_grinder_grind_size`	Fino	1
`coffee_grinder_grind_size`	Medio	2
`coffee_grinder_grind_size`	Grueso	3

Pensando en su preparación favorita, desarrolla el siguiente programa:

```
coffee_system:
# Carga 0 + 0x600DCAFE en t0
addi t0, zero, 0x600DCAFE
addi t2, zero, 0
addi t3, zero, 0
addi s1, zero, 1
addi s2, zero, 2
addi s3, zero, 128
addi s4, zero, 255
addi s5, zero, 30
addi s6, zero, 92

step_one:
    lw t1, 0(t0) # Carga Mem[t0 + 0] en t1
    # Salta a continue_step_one si t1 != s2
    bne t1, s2, continue_step_one
    jal ra, step_five # Llamado a step_five
    beq zero, zero, step_three
    continue_step_one:
        beq t1, s1, step_two
        sw s1, 4(t0) # Carga s1 en Mem[t0 + 4]
step_two:
    sw s2, 8(t0)
    sw s5, 12(t0)
    sw s3, 4(t0)
    jal ra, step_five
# Continúa en step_three -->
```

```
step_three:
    lw t1, 16(t0) # Carga Mem[t0 + 16] en t1
    # Salta a continue_step_three si t1 != s2
    bne t1, s2, continue_step_three
    jal ra, step_six
    beq zero, zero, end # Salto incondicional
    continue_step_three:
        beq t1, s1, step_four
        sw s1, 20(t0)
step_four:
    sw s6, 24(t0)
    sw s3, 20(t0)
    jal ra, step_six
    beq zero, zero, end
step_five:
    while_step_five:
        lw t1, 0(t0)
        beq t1, s2, while_step_five
    sw s4, 4(t0)
    jalr zero, 0(ra) # Retorno
step_six:
    while_step_six:
        lw t1, 16(t0)
        beq t1, s2, while_step_six
    sw s4, 20(t0)
    jalr zero, 0(ra)
end: # Término del programa.
```

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` **(0.7 ptos.)**: Se revisa el estado del molinillo. Si está moliendo, se ejecuta `step_five` y se salta a `step_three`. Si está encendida, se salta a `step_two`. En otro caso, se enciende antes de continuar con `step_two`.
- `step_two` **(0.7 ptos.)**: Se configura el molinillo con una molienda de grosor medio y un total de 30 gramos. Luego, empieza a moler y se ejecuta `step_five` antes de seguir a `step_three`.
- `step_three` **(0.7 ptos.)**: Se revisa el estado del hervidor. Si está hirviendo, se ejecuta `step_six` y se termina el programa. Si está encendida, se salta a `step_four`. En otro caso, se enciende antes de continuar con `step_four`.
- `step_four` **(0.7 ptos.)**: Se configura el hervidor con una temperatura de 92ºC. Luego, empieza a hervir y se ejecuta `step_six` antes de terminar el programa.
- `step_five` **(0.6 ptos.)**: Se revisa el estado del molinillo hasta que deje de moler.
- `step_six` **(0.6 ptos.)**: Se revisa el estado del hervidor hasta que deje de hervir.

Para cada `step`, se otorga la mitad del puntaje si existe como máximo un error o detalle faltante. Si existen dos o más errores o detalles faltantes, no se otorga puntaje.