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

IIC2343 - Arquitectura de Computadores (I/2023)

Examen

Pauta de evaluación

Pregunta 1: Preguntas conceptuales (7 ptos.)

(a) (1 pto.) En el videojuego The Legend of Zelda$^{\text{TM}}$ de la consola *Nintendo Entertainment System*$^{\text{TM}}$, se hace uso de rupias como moneda de cambio. El contador de rupias siempre es mayor o igual a cero, pero cuenta con la particularidad de que deja de incrementarse una vez que llega a 255. Explique, a partir de los contenidos del curso, por qué podría suceder esto.

Solución: La cantidad de rupias se puede interpretar como un tipo de dato `char` sin signo ya que su valor varía de 0 a 255, lo que ocurre al representar el dato con 8 bits. Dado esto, el juego controla que no se genere *overflow* ya que en caso de sumar una unidad más a 255 este valor cambiaría a cero. Se otorgan **0.5 ptos.** por señalar que el contador utiliza 8 bits y **0.5 ptos.** por indicar que deja de incrementarse por esta limitante.

(b) (2 ptos.) Indique si los siguientes números pueden representarse o no como un número `float` bajo el estándar IEEE754 con su valor exacto: (i) $2^{26} + 2$; (ii) 2^{70}. Si se puede representar, indique la representación; en otro caso, justifique por qué no se puede.

Solución: Debemos considerar que el tipo de dato `float` del estándar IEEE754 puede tener una precisión exacta de hasta 23 decimales por su mantisa.

(i) El valor $2^{26} + 2$ requiere un bit igual a 1 en la posición 25 del significante ya que $2^{26} = 10000000000000000000000000b$ y $2 = 10b$. Por este motivo, su representación como `float` pierde los 3 bits menos significativos y terminaría por almacenar el valor 2^{26}. Se otorgan **0.5 ptos.** por señalar que no se puede representar y **0.5 ptos.** por justificar según los bits de significante.

(ii) Como este número no requiere bits de significante, se puede representar solo con su exponente. Se toma el exponente desfasado en 127, que es igual a $197 = 11000101b$. Así, $(2^{70})_{\text{IEEE754}} = 01100010100000000000000000000000$. Se otorgan **0.5 ptos.** por señalar que se puede representar y **0.5 ptos.** por indicar la representación correcta.

(c) (1 pto.) Suponga que le entregan un computador construido por estudiantes del curso "Arquitectura de Computadores". Decide utilizarlo y define una variable de 4 bytes igual a 1024. No obstante, al imprimir su valor, la consola le arroja el valor 262144 (equivalente a 2^{18}). Explique qué podría estar mal con la implementación de los/as estudiantes y cómo podría resolverse.

Solución: La única forma en la que se puede obtener erróneamente el valor 262144 es por una mala interpretación del *endianness*: 1024 = 0x00000400 y 262144 = 0x00040000. Esto implica que se lee el dato usando el *endianness* contrario al que se utilizó para almacenarlo. El problema se resuelve invirtiendo el *endianness* de lectura o escritura para imprimir el valor correcto. Se otorgan **0.5 ptos.** por señalar que es un problema de *endianness* y **0.5 ptos.** por entregar una solución que vaya en línea con el problema.

(d) (2 ptos.) Indique, justificadamente, qué señales y componentes son esenciales en la comunicación entre los dispositivos I/O y la CPU para que esta pueda atender interrupciones ejecutando ISRs.

Solución: El componente esencial para la ejecución de interrupciones es el controlador de interrupciones que, a partir de la solicitud de los dispositivos I/O, le envía la señal IRQ (*Interrupt Request*) a la CPU. Esta, en caso de poder atender interrupciones, le envía de vuelta al controlador la señal INTA (*Interrupt Acknowledge*) para que le envíe el ID del dispositivo de mayor prioridad para acceder a la dirección de su ISR dentro del vector de interrupciones. Se otorgan **0.5 ptos** por mencionar el controlador de interrupciones; **0.5 ptos.** por mencionar la señal IRQ; **0.5 ptos.** por mencionar la señal INTA; y **0.5 ptos.** por mencionar el vector de interrupciones.

(e) (1 pto.) Suponga que se implementa memoria virtual en el computador básico del curso. ¿Las direcciones de los registros `SP` y `PC` deben ser virtuales o físicas en esta implementación? Justifique su respuesta.

Solución: Tanto el registro `SP` como el registro `PC` deben apuntar a direcciones virtuales. Esto se debe a que en el contexto de memoria virtual las direcciones de memoria de los datos e instrucciones dependen de los marcos físicos en los que se ubiquen, los que pueden cambiar a través de *swapping*. Se otorgan **0.5 ptos.** por señalar correctamente que utilizan direcciones virtuales y **0.5 ptos.** por su justificación.

(f) (2 ptos.) Señale qué modificaciones mínimas, a nivel de *hardware*, se deben realizar sobre el computador básico con *pipeline* para que este sea 2-*issue*. Puede asumir que el compilador se hace cargo del envío de *bundles* de 2 instrucciones como VLIW.

Solución: Para que sea 2-*issue* se requiere como mínimo: (i) agregar una unidad de ejecución adicional; (ii) la capacidad de leer dos instrucciones por iteración; (iii) la capacidad de decodificar 2 instrucciones por iteración; (iv) la capacidad de escribir 2 resultados en la memoria de datos por instrucción; y (v) la capacidad de actualizar más de un registro o grupo de registros por iteración. Se otorga **1 pto.** por la mención de la unidad de ejecución adicional y **0.5 ptos.** por la mención de cada uno de los otros puntos, por lo que basta que señalen dos de ellos.

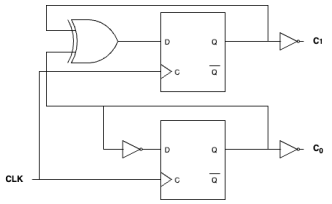
(g) (1 pto.) Explique brevemente por qué las GPU suelen categorizarse como arquitecturas SIMT dentro del paradigma SIMD.

Solución: Se categorizan como SIMT (*Single Instruction, Multiple Threads*) ya que su construcción interna se ajusta a la ejecución de un mismo programa con múltiples *threads* (cada uno con datos distintos). Se otorgan **0.5 ptos.** por la mención de *threads* y **0.5 ptos.** por señalar que su construcción se adapta a la ejecución de estos.

Pregunta 2: Circuitos eléctricos y almacenamiento (5 ptos.)

(a) (5 ptos.) Diseñe, utilizando desde compuertas lógicas **hasta flip-flops**, un contador secuencial de 2 bits **inverso**, que parte con valor 11 y que decrementa con cada flanco de subida de la señal de control *clock* hasta llegar a 00 y luego hacer *overflow* para partir de nuevo, *i.e.* 11-10-01-00-11-10-01-00-etc.

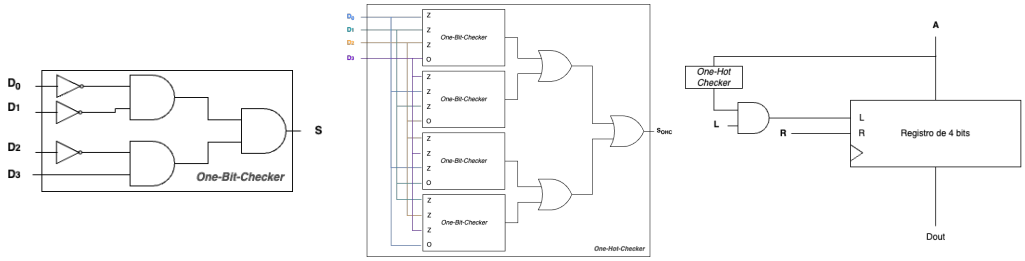
Solución: Se puede diseñar este componente como un contador secuencial de 2 bits incremental, pero con sus señales de salida negadas. Se puede que esto cumple con lo solicitado ya que si se niegan todos los números de la secuencia de este contador, se tiene: 00→11,01→10,10→01,11→00. De esta forma, el circuito resultante es el siguiente:



Se otorgan **2 ptos.** si el bit C_0 es correcto y **3 ptos.** si el bit C_1 es correcto. Se otorgan **3 ptos.** del total si solo existe una falla de diseño y **2 ptos.** del total si existen como máximo dos fallas.

(b) (5 ptos.) Se desea tener un registro de estados de 4 bits de tipo *one-hot*, *i.e.* que siempre tiene un **único** bit igual a 1, no puede existir más de uno ni contener solo ceros. Modifique el diagrama de un registro para que este habilite su señal de carga `L` si, y solo si el valor de entrada cumple con ser *one-hot*.

Solución: Se diseñan dos componentes: *One-Bit-Checker* cuya salida es 1 solo si el bit más significativo de una entrada de 4 bits es el único igual a 1; y *One-Hot-Checker*, que hace uso de cuatro *One-Bit-Checkers* para determinar si una entrada de 4 bits cumple o con ser *one-hot* revisando si es igual a 1000, 0100, 0010 o 0001. Finalmente, se utiliza este último para recibir el valor de entrada del registro de 4 bits y se conecta su salida con una compuerta `AND` y la señal `L` para la carga sea efectiva solo para entradas *one-hot*.

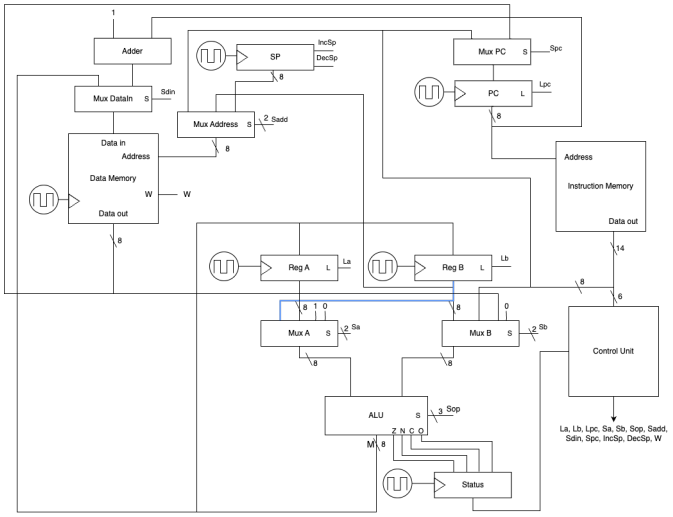


Se otorgan **2 ptos.** por diseñar un circuito que detecte que una señal de 4 bits solo posea un 1 y **1 pto.** si falla en un caso; **2 ptos.** por diseñar con circuito que detecte que una señal de 4 bits sea *one-hot* y **1 pto.** si falla en un caso; y **1 pto.** por conectar el circuito con la señal de carga `L`.

Pregunta 3: Computador básico (7 ptos.)

(a) (3 ptos.) Modifique la arquitectura del computador básico para implementar las instrucciones `NOT B,B`, `SHL B,B` y `SHR B,B`, es decir, que realice las operaciones de un operando sobre el registro `B` y almacene el resultado en él. Modifique el diagrama adjunto o señale todas las conexiones y componentes que se deben añadir. Incluya la combinación de señales que las ejecutan.

Solución: Para esta implementación, se conecta el registro B con el componente `Mux A` para utilizar el valor de B como *input* de las operaciones `NOT`, `SHL` y `SHR` de la ALU:



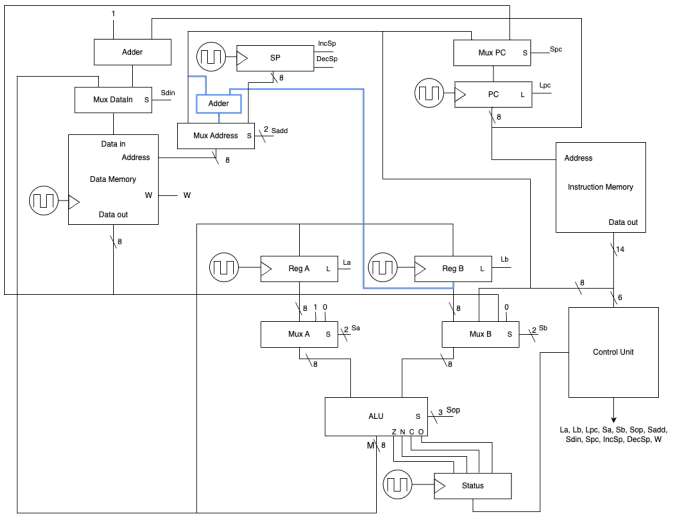
De esta forma, sin modificar ni agregar las señales de la microarquitectura, se tiene la siguiente tabla para la ejecución de las instrucciones implementadas:

Instrucción	L_A	L_B	L_{PC}	W	Inc_{SP}	Dec_{SP}	S_A	S_B	S_{OP}	S_{Add}	S_{DIn}	S_{PC}
`NOT B,B`	0	1	0	0	0	0	B	-	NOT	-	-	-
`SHL B,B`	0	1	0	0	0	0	B	-	SHL	-	-	-
`SHR B,B`	0	1	0	0	0	0	B	-	SHR	-	-	-

Se otorgan **0.5 ptos.** por la modificación correcta de la microarquitectura y **0.5 ptos.** por cada instrucción correctamente implementada según su combinación de señales.

(b) (3 ptos.) Modifique la arquitectura del computador básico para implementar las instrucciones `MOV A,(B+offset)` y `MOV (B+offset),A`, *i.e.* instrucciones de direccionamiento indirecto con registro `B` y *offset*, siendo este último un literal que puede ser positivo o negativo. Modifique el diagrama adjunto o señale todas las conexiones y componentes que se deben añadir. Incluya la combinación de señales que las ejecutan.

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)` y `MOV (B),A`. La tabla de señales es como sigue para las instrucciones implementadas:

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

Se otorgan **1.5 ptos.** por la modificación correcta de la microarquitectura y **0.75 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.

(c) (4 ptos.) Asuma que se agregan las instrucciones anteriores a la ISA del computador básico y se ejecuta el programa adjunto. Señale solo los valores finales de los registros `A`, `B` y `SP`. Asuma que todos los registros son de 8 bits y que `SP` se interpreta como un número positivo.

```
MOV A,2           // Dirección Mem. Instr.: 0x00
MOV B,5           // Dirección Mem. Instr.: 0x01
PUSH A            // Dirección Mem. Instr.: 0x02
PUSH B            // Dirección Mem. Instr.: 0x03
CALL mystery      // Dirección Mem. Instr.: 0x04
POP B             // Dirección Mem. Instr.: 0x05-0x06
POP A             // Dirección Mem. Instr.: 0x07-0x08
JMP end           // Dirección Mem. Instr.: 0x09
mystery:
  MOV B,0         // Dirección Mem. Instr.: 0x0A
  NOT B,B         // Dirección Mem. Instr.: 0x0B
  MOV A,(B + -2)  // Dirección Mem. Instr.: 0x0C
  ADD A,4         // Dirección Mem. Instr.: 0x0D
  MOV (B + -2),A  // Dirección Mem. Instr.: 0x0E
  RET             // Dirección Mem. Instr.: 0x0F-0x10
end:
```

Solución: Se señalan los puntos clave en donde se modifican los registros y por qué:

1. Antes de `CALL mystery`, se tiene `A = 2`, `B = 5` por las instrucciones `MOV` y `SP = 253` por las instrucciones `PUSH`.
2. Posterior a `CALL mystery`, se tiene `A = 2`, `B = 5` y `SP = 252` por el almacenamiento de la dirección de retorno 0x05 (`PC+1`).
3. Antes de `RET`, se tiene `A = 9`, `B = 255` y `SP = 252`. En este punto, lo que se hizo fue modificar la dirección de retorno de la subrutina `mystery`, apuntando ahora a la instrucción `JMP end`.
4. Posterior a `RET`, el programa termina sin ejecutar las instrucciones `POP` y se tiene `A = 9`, `B = 255` y `SP = 253`.

Se otorgan **1.5 ptos.** por obtener el valor final correcto de `A`; **1.5 ptos.** por obtener el valor final correcto de `B`; y **1 pto.** por obtener el valor final correcto de `SP`. Si existe desarrollo del código y se encuentran errores de arrastre, se otorgan **2 ptos.** del total.

Pregunta 4: Memoria caché y memoria virtual (8 ptos.)

(a) (4 ptos.) Suponga que posee una memoria caché 2-*way associative* de 8 líneas y 4 palabras por línea para una memoria principal de 64 bytes. Indique el *hit-rate* y el estado final de la caché (bits de *tag* y bit de validez) para los siguientes accesos de memoria: 0, 49, 19, 34, 2, 51, 16, 35. Asuma que se implementa una política de reemplazo LRU.

Solución: En primer lugar, como la memoria principal es de 64 bytes y se asume que cada dirección almacena una palabra de memoria de 1 byte, entonces se tienen 2^6 direcciones y se requieren $\log_2(2^6) = 6$ bits para cada dirección. Por otra parte, como contamos con 4 palabras por línea, requerimos de $\log_2(4) = 2$ bits de *offset*. Luego, al ser 2-*way associative*, la caché se divide en conjuntos de 2 líneas cada uno, por lo que se tienen $\frac{8}{2} = 4$ conjuntos y se requieren $\log_2(4) = 2$ bits de índice de conjunto. Por lo tanto, los bits restantes de cada dirección se utilizan como *tag*, siendo 2. Ahora, podemos ver la secuencia de accesos a memoria de la siguiente forma: 000000, 110001, 010011, 100010, 000010, 110011, 010000, 100001. De esta secuencia notamos que todos los bloques de memoria se mapean al conjunto 00. Como el conjunto solo puede almacenar 2 líneas, desde el tercer acceso a memoria comenzarán los reemplazos por LRU y por el valor del *tag* de cada dirección nunca existirán *hits*. Se deduce entonces que el *hit-rate* es igual a $\frac{0}{8}$ y que solo las líneas 000 y 001 de la caché quedan con bit de validez 1 y con los *tag* 01 y 10 respectivamente. Se otorgan **0.5 ptos.** por obtener la cantidad correcta de bits por dirección; **0.5 ptos.** por obtener la cantidad correcta de bits de *offset*; **0.5 ptos.** por obtener la cantidad correcta de bits de identificador de conjunto; **0.5 ptos.** por obtener la cantidad correcta de bits de *tag*; **1 pto.** por llegar al *hit-rate* correcto y **1 pto.** por llegar al estado final correcto de la caché. Se otorgarán **3 ptos.** del total si se detecta un buen desarrollo de la pregunta pero con errores de arrastre.

(b) (2 ptos.) Indique si se puede aumentar el *hit-rate* de la pregunta anterior solo cambiando la función de correspondencia. No es necesario que lo calcule, pero sí que justifique su respuesta.

Solución: Si en la pregunta anterior la caché hubiera sido *fully associative*, se habrían utilizado las primeras cuatro líneas de la caché para almacenar los bloques de memoria de las direcciones accedidas y los siguientes cuatro accesos habrían tenido *hit*, por lo que el *hit-rate* aumentaría a $\frac{4}{8} = 0{,}5$. Se otorga **1 pto.** por indicar que sí es factible y **1 pto.** por justificar la respuesta, con o sin cálculos.
Por otra parte, es posible argumentar que no es posible cambiar la función de correspondencia de una caché al ser algo dependiente de la construcción del *hardware*. También se otorga todo el puntaje si se contesta bajo esta consideración.

(c) (4 ptos.) Suponga que se tiene un espacio de direcciones virtuales de 30 bits y direcciones físicas de 20 bits. Indique, en bytes, el tamaño de página **mínimo** que permite que la tabla de páginas de un proceso "quepa" por completo en una página, *i.e.* que su tamaño no supere el tamaño de la tabla. Asuma que en cada entrada de la tabla de páginas se utilizan 3 bits de *metadata*.

Solución: El tamaño de página mínimo necesario para cumplir lo solicitado es de 32KB. Para este tamaño, se tiene un total de $\log_2(2^{15}) = 15$ bits de *offset*, lo que deriva en 15 bits de número de página y 5 bits de número de marco. Entonces, sabemos que se tiene un total de 2^{15} entradas de tabla de página y un ancho de entrada de $5 + 3 = 8$ bits (1 byte). Esto da un total de 2^{15}B = 32KB, haciendo que la tabla de páginas quepa de forma exacta en una página. A partir de la elección de un tamaño de página (independiente de ser el esperado), se otorgan: **0.5 ptos.** por obtener la cantidad correcta de bits de *offset*; **0.5 ptos.** por obtener la cantidad correcta de bits de número de página; **0.5 ptos.** por obtener la cantidad correcta de bits de número de marco; **0.5 ptos.** por obtener la cantidad correcta de entradas de la tabla de página; **0.5 ptos.** por obtener el tamaño correcto de entrada de la tabla de páginas; y **0.5 ptos.** por obtener el tamaño correcto de la tabla de páginas. Se otorga **1 pto.** por llegar al resultado correcto. Se consideran también correctas respuestas cuyos cálculos hayan sido realizados con la transformación 1000B = 1KB en vez de 2^{10}B = 1KB (y de la misma forma para MB, GB y TB).

Pregunta 5: RISC-V e I/O (6 ptos.)

(a) (2 ptos.) Indique justificadamente si el siguiente fragmento de código respeta la convención de llamada de RISC-V.

```
.data
  N:       .word 23
  divisor: .word 12
.text
  main:
    addi sp, sp, -4
    sw ra, 0(sp)
    call remainder
    lw ra, 0(sp)
    addi sp, sp, 4
    j end
  remainder:
    lw a0, N
    lw a1, divisor
    rem a0, a0, a1
    ret
  end:
```

Solución: El fragmento de código **no respeta** la convención de RISC-V por dos motivos: (1) Los registros `a0, a1` son *caller-saved*, por lo que debieron ser respaldados en el *stack* antes del llamado de la subrutina; y (2) los registros `a0, a1` se usan como argumentos de la subrutinas, por lo que deberían cargar sus valores **antes** del llamado y no dentro de ella. Se otorga **1 pto.** por señalar correctamente que no respeta la convención de RISC-V y **1 pto.** por justificar su respuesta con alguno de los argumentos señalados. Se otorga **1 pto.** en total si se señala que el fragmento de código respeta la convención con justificación por respaldo de `ra`, uso de `a0-a1` como argumentos o `a0` como retorno.

(b) (2 ptos.) Indique si el siguiente fragmento de código en RISC-V termina su ejecución. Si lo hace, indique el valor del registro `a0`. En otro caso, justifique por qué no termina.

```
.data
    N: .word 4
.text
    main:
        addi sp, sp, -4
        sw   ra, 0(sp)
        lw   t0, N
        mv  a0, t0
        call factorial
        lw   ra, 0(sp)
        addi sp, sp, 4
        j end
    factorial:
        addi sp, sp, -4
        sw   a0, 0(sp)
        blez a0, factorial_zero
        addi a0, a0, -1
        call factorial
        lw   t0, 0(sp)
        mul  a0, a0, t0
        j factorial_end
        factorial_zero:
            li   a0, 1
        factorial_end:
            addi sp, sp, 4
        ret
    end:
```

Solución: El programa anterior **no termina** su ejecución ya que posee llamadas recursivas que **no respaldan la dirección de retorno del registro `ra`**. Por dicho motivo, el programa podría no terminar o arrojar errores. Se otorga **1 pto.** por señalar que el programa no termina correctamente y **1 pto.** por justificarlo.

Para las siguientes preguntas, suponga que tiene una impresora cuyos registros de estado y comando de 32 bits están mapeados en memoria desde la dirección 0x10150000. A continuación, se indican las tablas de direcciones, comandos y estado:

Offset	**Nombre**	**Descripción**
0x00	`printer_stat`	Registro de estado.
0x04	`printer_comm`	Registro de comando.
0x08	`printer_clr`	Registro de color.
0x0C	`printer_sz`	Registro de tamaño hoja.
0x10	`printer_or`	Registro de orientación hoja.

Nombre	**Descripción**	**Valor**
`printer_stat`	Apagada.	0
`printer_stat`	Prendida.	1
`printer_stat`	Prendiendo.	127
`printer_stat`	Imprimiendo.	255
`printer_comm`	Prender.	0
`printer_comm`	Apagar.	1
`printer_comm`	Imprimir.	2
`printer_clr`	Blanco y negro.	0
`printer_clr`	Color.	1
`printer_sz`	Tamaño carta.	0
`printer_sz`	Tamaño oficio.	1
`printer_or`	Horizontal.	0
`printer_or`	Vertical.	1

Suponga que bajo este contexto se ejecuta el siguiente programa en RISC-V:

```
.text
  main:
    li t0, 0x10150000
    lw t1, 0(t0)
    li s0, 1
    li s1, 127
    li s2, 0
    beq t1, s0, actionThree
    beq t1, s1, actionTwo
    beq t1, s2, actionOne
    j end
  actionOne:
    li t2, 0
    sw t2, 4(t0)
  actionTwo:
    lw t1, 0(t0)
    beq t1, s0, actionThree
    j actionTwo
  actionThree:
    li s4, 1
    sw s4, 8(t0)
    li s5, 0
    sw s5, 12(t0)
    li s6, 1
    sw s6, 16(t0)
    li t2, 2
    sw t2, 4(t0)
  end:
```

Respecto a este programa:

(c) (2 ptos.) Indique justificadamente qué tipo de comunicación se presenta entre la CPU y la impresora en el código anterior.

Solución: La CPU se comunica con la impresora a través de *polling*, ya que esta consulta su estado consistentemente hasta que cambia, particularmente desde que está en estado "prendiendo" hasta que pasa a "prendida". Se otorga **1 pto.** por señalar correctamente el tipo de comunicación y **1 pto.** por justificarlo correctamente. Se otorga **1 pto.** del total si no señalan *polling*, pero sí que los dispositivos son de tipo *memory mapped*.

(d) (4 ptos.) Explique en términos prácticos, según el programa y las tablas adjuntas, lo que realizan las acciones `actionOne`, `actionTwo` y `actionThree`.

Solución: En el segmento `actionOne` la CPU le da el comando a la impresora para que se prenda; en el segmento `actionTwo` la CPU espera a que la impresora termine de prender (paso de estado "prendiendo" a "prendida"); y en el segmento `actionThree` la CPU se encarga de configura la impresora para que imprima una hoja a color de tamaño carta con orientación vertical. Se otorga **1 pto.** por describir correctamente `actionOne`; **1 pto.** por describir correctamente `actionTwo` y **2 ptos.** por describir correctamente `actionThree`. Se hace un descuento de: **0.25 ptos.** por errores menores en la descripción de `actionOne`; **0.25 ptos.** por errores menores en la descripción de `actionTwo`; **0.5 ptos.** si no se especifica la configuración completa de la hoja en `actionThree`; y **0.25 ptos.** si hay solo un error en la descripción de la configuración de la hoja en `actionThree`.

Pregunta 6: Paralelismo y Coherencia de caché (8 ptos.)

(a) (2 ptos.) Describa cómo se puede modificar el computador básico con *pipeline* para que las predicciones de salto erróneas pierdan dos ciclos en vez de tres.

> **Solución:** Como la *flag* `Z` se computa en la etapa `EX`, una posible modificación consiste en posicionar la *Jump Unit* en dicha etapa para poder determinar en una iteración menos si corresponde realizar el salto o no. De esta forma, tanto para saltos incondicionales como condicionales se requerirá un *flush* solo para las etapas `IF` e `ID` si corresponde, perdiendo dos ciclos y no tres. Se otorga **1 pto.** por describir una modificación y **1 pto.** por su explicación o justificación de su funcionamiento.

(b) (4 ptos.) A partir del siguiente programa del computador básico con *pipeline*:

```
DATA:
  i   0
  arr 13
      13
  len 1
CODE:
  main:
    MOV A,(i)
    MOV B,(len)
    JEQ A,B,end // if A == B go to end
    MOV B,arr
    ADD B,A
    ADD A,1
    MOV (i),A
    MOV A,(B)
    SUB A,2
    MOV (B),A
    JMP main
  end:
    MOV A,0
    MOV B,0
    ADD (i)
```

Determine el número de ciclos que demora su ejecución detallando en un diagrama los estados del *pipeline* por instrucción. Asuma que el manejo de *stalling* es por *software* a través de la instrucción `NOP` y que la unidad predictora de saltos asume que estos nunca se realizan. Indique en el diagrama cuando ocurre *forwarding* (con flechas del registro a la etapa que correspondan), *stalling* y *flushing* (con tachado en las instrucciones *flusheadas*).

Solución: A continuación, se muestra el *pipeline* resultante de la ejecución del programa.

Del *pipeline* anterior se deduce que el programa termina de correr después de 30 ciclos. Se otorga puntaje de la siguiente forma: **0.4 ptos.** por cada *stalling* correctamente detectado; **0.5 ptos.** por cada *flushing* correctamente detectado; **0.2 ptos.** por cada *forwarding* correctamente detectado; y **0.2 ptos.** por la deducción correcta de la cantidad de ciclos en la que corre el programa.

(c) (4 ptos.) Suponga que posee una arquitectura MIMD de memoria compartida con 2 CPU que poseen su propia caché y que siguen el protocolo MESI para asegurar consistencia. La memoria contiene los siguientes datos de variables y las CPU0 y CPU1 ejecutan los siguientes programas:

Dirección	*Label*	Valor
0x00	i	1
0x01	arr	1
0x02		10

```
// CPU0        // CPU1
MOV B,(i)      MOV A,(arr)
MOV A,(arr)    MOV B,arr
ADD B,A        ADD B,A
SHR A,A        MOV (B),A
MOV (B),A      NOP
```

Asumiendo que cada dirección se almacena en una línea distinta, indique el estado de cada línea de las caché (M/E/S/I) para cada iteración completando la tabla adjunta, asumiendo que todas las líneas parten en estado I:

	CPU0			**CPU1**		
Instrucción	**i**	**arr[0]**	**arr[1]**	**i**	**arr[0]**	**arr[1]**
1						
2						
3						
4						
5						

Solución: A continuación se muestra la tabla final esperada, que incluye columnas para los valores de los registros A y B y el valor en memoria de las variables en cada iteración:

	CPU0					**CPU1**					**Memoria**		
Instrucción	**i**	**arr[0]**	**arr[1]**	**A**	**B**	**i**	**arr[0]**	**arr[1]**	**A**	**B**	**i**	**arr[0]**	**arr[1]**
1	E	I	I	-	1	I	E	I	1	-	1	1	10
2	E	S	I	1	1	I	S	I	1	1	1	1	10
3	E	S	I	1	2	I	S	I	1	2	1	1	10
4	E	S	I	0	2	I	S	M	1	2	1	1	1
5	E	S	M	0	2	I	S	I	1	2	1	1	0

Se otorgan **2 ptos.** por cada CPU con estado final correcto. Se otorga la mitad del puntaje por CPU si se tiene como máximo una celda con estado erróneo.

Pregunta 7: Déjà vu (7 ptos.)

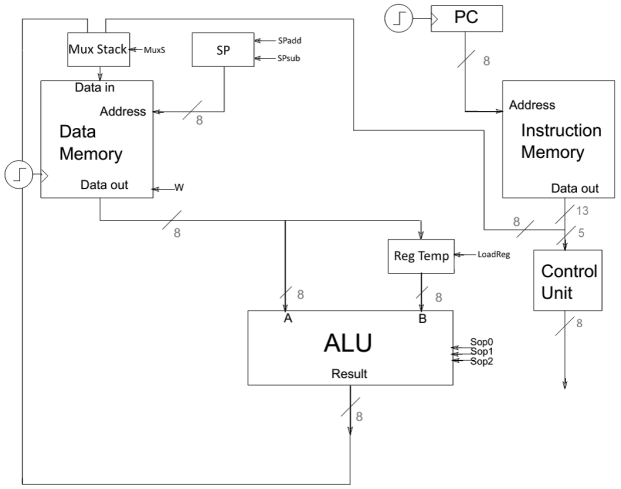
(a) (3 ptos.) Dados números naturales K, N, T, donde $K \geq 2$ y $N > T > 0$, describa un algoritmo para transformar de manera **eficiente** números naturales codificados en base K^N a K^T.

Solución: Un algoritmo eficiente puede realizarse en dos pasos:

1. Transformar cada cifra del número original en N cifras que representen el mismo número en base K, manteniendo el orden correlativo entre cifras.
2. Agrupar las cifras del número expandido en grupos de T cifras y transformar cada grupo en un número en base K^T, nuevamente manteniendo el orden correlativo.

Este algoritmo funciona ya que K^N y K^T son de la misma base, por lo que se puede **mostrar** a través de transformación a base decimal que efectivamente se respeta el valor del número transformado. Se otorgan **1 pto.** por detallar un algoritmo, **0.5 ptos.** si no presenta ningún error y **1.5 ptos.** si no requiere transformaciones de bases intermedias.

(b) (4 ptos.) Una máquina de *stack* es un computador que utiliza una memoria de *stack* en vez de registros para almacenar los resultados de las operaciones. Suponga que le entregan, **sin que el equipo docente se entere**, el siguiente diagrama e ISA:



Instrucción	*Opcode*	Señales de control
`PUSH Lit`	00000	`SPsub = 1`
	00001	`W = 1, MuxS = 1`
`POP`	00010	`SPadd = 1`
`ADD`	00011	`SPadd = 1, LoadReg = 1`
	00100	`Sop = ADD, W = 1, MuxS = 0`
`SUB`	00101	`SPadd = 1, LoadReg = 1`
	00110	`Sop = SUB, W = 1, MuxS = 0`
`AND`	00111	`SPadd = 1, LoadReg = 1`
	01000	`Sop = AND, W = 1, MuxS = 0`
`OR`	01001	`SPadd = 1, LoadReg = 1`
	01010	`Sop = OR, W = 1, MuxS = 0`
`XOR`	01011	`SPadd = 1, LoadReg = 1`
	01100	`Sop = XOR, W = 1, MuxS = 0`
`SHL`	01101	`Sop = SHL, W = 1, MuxS = 0`
`SHR`	01110	`Sop = SHR, W = 1, MuxS = 0`
`NOT`	01111	`Sop = NOT, W = 1, MuxS = 0`
`NOP`	10000	`W = 0`

De lo anterior se destaca que las instrucciones de tipo `PUSH` toman dos ciclos, mientras que las del computador básico toman un ciclo. Explique qué diferencias prácticas existen en el manejo de `SP` del computador básico y el de la máquina presentada, profundizando en cómo cambia la forma en la que se maneja la memoria de *stack*.

Solución: La diferencia en el manejo de `SP` está en la dirección a la que apunta después de un `PUSH`: En la máquina de *stack* presentada siempre apunta a la dirección del último elemento ingresado; mientras que en el computador básico se encuentra en **una posición por sobre el tope**. Esto implica que la instrucción `POP` de la máquina de *stack* toma un ciclo en comparación con el computador básico donde se requieren dos ciclos por el incremento de `SP` para que apunte a la dirección correcta. Si la máquina de *stack* manejara `SP` de la misma forma que el computador básico, entonces sus operaciones aritméticas y lógicas tomarían un ciclo adicional. Otra consecuencia es que, para 8 bits, siempre se parte escribiendo en la dirección 254, por lo que la dirección 255 queda sin uso a menos que se establezca un valor inicial de `SP` igual a cero. Se otorgan **2 ptos.** por señalar la diferencia de la dirección a la que apunta `SP` y **2 ptos.** por señalar la consecuencia respecto a los ciclos de ejecución de las instrucciones.

(c) (3 ptos.) Considere un sistema con páginas de 16KB, direcciones virtuales de 48 bits, direcciones físicas de 30 bits y 4 bits de *metadata* para las traducciones. Proponga un esquema de paginación **multinivel** que asegure que la tabla de páginas de cada nivel "quepa" en una página, *i.e.* que su tamaño no supere los 16KB.

Solución: Inicialmente se tiene un tamaño de páginas de 16KB. Si se asume que cada dirección almacena una palabra de memoria de un byte, se tienen $\log_2(2^{14}) = 14$ bits de *offset*, 34 bits de número de página y 16 bits de número de marco. Entonces, para un nivel, la tabla de páginas tiene 2^{34} entradas y un ancho de entrada de $16 + 4 = 20$ bits $= 2.5$B. Esto da un tamaño total de $2^{34} * 2{,}5\text{B} = 40$GB, lo que no cumple con lo requerido.

Una primera opción es usar esquema de paginación de 2 niveles, usando 17 bits para el índice de cada nivel. Así, la tabla de primer nivel tendrá un tamaño igual a $2^{17} * 30\text{b}$ $= 480$KB, donde los 30 bits provienen de la dirección física que posee cada entrada para ubicar la tabla de segundo nivel, mientras que la tabla de segundo nivel tendrá un tamaño igual a $2^{17} * 2{,}5\text{B} = 320$KB, siendo esta la que tiene en su entrada los bits de marco físico y *metadata*. Esta propuesta sigue sin cumplir lo requerido.

Una segunda opción es utilizar un esquema de paginación de tres niveles, utilizando 11 bits para los índices de los primeros dos niveles y 12 bits para el índice de tercer nivel. En este caso, las tablas de primer y segundo nivel tendrán un tamaño igual a $2^{11} * 30\text{b}$ $= 7.5$KB, mientras que la tabla de tercer nivel tendrá un tamaño igual a $2^{12} * 2{,}5\text{B} =$ 10KB. Por lo tanto, con esquemas de tres o más niveles se cumple lo requerido.

Se otorga el puntaje completo si la propuesta cumple con el requisito de tamaño. En otro caso, por desarrollo se otorgan:

- **0.5 ptos.** por obtener la cantidad correcta de bits de *offset*.
- **0.5 ptos.** por obtener la cantidad correcta de bits de índice por nivel.
- **0.5 ptos.** por obtener el tamaño correcto de entrada de la tabla de páginas del último nivel.
- **0.5 ptos.** por calcular de forma correcta el tamaño de la tabla de páginas en todos los niveles.