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

IIC2343 - Arquitectura de Computadores (II/2023)

Interrogación 2

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) (1 pto.) Indique una consecuencia que podría existir si no se respaldan las *flags* computadas por el procesador **antes** de atender la interrupción de un dispositivo I/O en un instante de tiempo determinado.

> **Solución:** Si al retornar de la ejecución de la ISR el procesador ejecuta una instrucción de salto condicional, no se podría asegurar que se realice correctamente ya que se debe hacer uso de las *flags* computadas **antes** de la interrupción. Se otorga **1 pto.** si la respuesta alude a problemas de consistencia respecto a saltos condicionales. No hay puntaje parcial.

(b) (1 pto.) Indique qué tipo de *mapping* debe tener un dispositivo I/O de almacenamiento dentro de un computador para que pueda interactuar con el controlador de DMA.

> **Solución:** Los valores de los registros de origen y destino del controlador de DMA deben ser direcciones del bus compartido para poder realizar la transferencia de datos entre la memoria y dispositivos I/O de almacenamiento. Por esto, el controlador solo puede acceder a direcciones de dispositivos *memory mapped*. Se otorgan **0.5 ptos.** por señalar de forma correcta el tipo de *mapping* y **0.5 ptos.** por entregar argumentos que aludan a las direcciones de memoria del bus compartido.

(c) (2 ptos.) Suponga que posee un computador que hace uso de una caché *directly mapped* y una persona, externa al curso, le dice que su tiempo promedio de acceso a memoria sería **definitivamente** menor si es que su caché fuera de tipo *fully associative*. ¿Es esto cierto? Justifique su respuesta en base a los criterios que inciden sobre el tiempo promedio de acceso a memoria.

Solución: La sentencia anterior **no es cierta**. Si bien las funciones de correspondencia *fully associative* hacen un uso completo de la caché y pueden tener un *hit rate* mayor, su *hit time* es también mayor al tener que realizar una búsqueda en todas sus líneas para verificar la existencia de un bloque de memoria. Entonces, si la función de correspondencia no incidiera en el *hit rate* para una secuencia de accesos determinada, el tiempo promedio para una caché *directly mapped* sería menor. Se otorga **1 pto.** por indicar correctamente que la sentencia no es correcta y **1 pto.** por dar argumentos relativos al *hit time* o búsqueda de la caché para la función de correspondencia *fully associative*.

(d) (2 ptos.) Suponga que sobre un espacio de direccionamiento virtual de 30 bits y un espacio de direccionamiento físico de 20 bits implementa un esquema de paginación con páginas de 8KiB. Indique qué modificación se debe hacer sobre este esquema para que la cantidad de entradas de la tabla de páginas disminuya a **la mitad**.

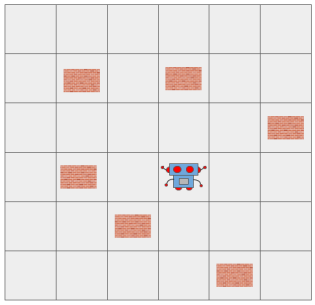
Solución: La cantidad de entradas de una tabla de páginas está dada por el número de páginas. Por lo tanto, basta con duplicar el tamaño de la página para que la cantidad de páginas (y, por ende, de entradas) disminuya a la mitad. Esto correspondería a un tamaño de página de 16KiB. Se otorga **1 pto.** por señalar correctamente que la cantidad de entradas depende del número de páginas y **1 pto.** por indicar que se debe duplicar el tamaño de una página para lograr lo solicitado. No es necesario realizar los cálculos ni entregar el tamaño resultante.

(e) (2 ptos.) Suponga que se modifica la arquitectura del computador básico con *pipeline* para dividir la etapa `MEM` en dos nuevas etapas: `MEM-R` para realizar solo operaciones de lectura; y `MEM-W` para realizar solo operaciones de escritura. Indique, justificadamente, si esta modificación introduce algún tipo de error **nuevo** necesario de manejar, no es necesario que entregue una solución. Puede asumir que la *Jump Unit* y la *Control Hazard Unit* se ubican en la etapa `MEM-R`.

Solución: La modificación introduce un *hazard* estructural. Este ocurre cuando dos o más etapas de un *pipeline* hacen uso simultáneo de un mismo componente. En este caso, las etapas `MEM-R` y `MEM-W` acceden a la memoria de datos, por lo que si tenemos de forma secuencial la ejecución de una instrucción de escritura y luego una de lectura, ambas accederán a la memoria al mismo tiempo. Se otorga **1 pto.** por indicar que sí se introduce un nuevo tipo de error y **1 pto.** por señalar que este es un *hazard* estructural. También se considera correcto si no se utiliza dicho término, pero aluden al acceso de memoria simultáneo.

Pregunta 2: Comunicación con dispositivos I/O (8 ptos.)

Un robot simple, conectado a un computador, es accesible mediante *memory mapping*. Este robot se mueve en un espacio cuadrado infinito, en el cual cada grilla puede estar vacía o contener una muralla. Este robot posee registros internos *mapeados* en memoria desde la dirección base 0x10101000. A continuación, se muestra una ilustración de este ejemplo y las tablas de direcciones y valores de estado y comando del robot:



Offset	Nombre	Descripción
0x00	`robot_stat`	Registro estado
0x04	`robot_comm`	Registro comando
0x08	`robot_turns`	Contador giros

Nombre	Descripción	Valor
`robot_stat`	Recién encendido	0
`robot_stat`	Nada que informar	255
`robot_stat`	Espacio libre adelante	1
`robot_stat`	Muralla adelante	2

Nombre	Descripción	Valor
`robot_comm`	Encender	255
`robot_comm`	Apagar	0
`robot_comm`	Avanzar	1
`robot_comm`	Giro antihor. en 90º	2
`robot_comm`	Examinar	4

En resumen, el robot tiene comandos para ser prendido, apagado, avanzar un espacio hacia adelante, girar en sentido antihorario en 90º o examinar lo que tiene delante de él. Cada vez que el robot se encuentra desocupado, *i.e.* que ha sido recién iniciado o ha terminado una acción, genera una interrupción para informar que es posible darle un nuevo comando. Adicionalmente, cuenta con un registro `robot_turns` que indica la cantidad de veces que ha girado. Este se utiliza para evitar que el robot retroceda, *i.e.* **para evitar que avance si ha girado dos veces**. A continuación, se muestra el fragmento de la ISR en RISC-V que plasma el control del robot:

```
ISR_robot:
  li t0, 0x10101000             # Carga en el registro t0 el valor del literal 0x10101000
  step_one:
    li s0, 0
    li s1, 255
    li s2, 1
    lw t1, 0(t0)
    beq t1, s0, step_two
    beq t1, s1, step_three
    beq t1, s2, step_four
    j step_six                  # Salto incondicional a step_six
  step_two:
    li t2, 0
    sw t2, 8(t0)
  step_three:
    li t2, 4
    sw t2, 4(t0)
    j end_isr
  step_four:
    li t2, 2
    lw t3, 8(t0)
    beq t2, t3, step_six
  step_five:
    li t2, 0
    sw t2, 8(t0)
    li t2, 1
    sw t2, 4(t0)
    j end_isr
  step_six:
    lw t2, 8(t0)
    addi t2, t2, 1
    sw t2, 8(t0)
    li t2, 2
    sw t2, 4(t0)
  end_isr:
    mret                        # Retorno ISR.
```

Describa, basándose en las tablas provistas, las tareas que realiza el robot en cada *label* `step_*` dentro de la ISR. Considere que la instrucción `mret` es equivalente al retorno, pero utilizado en el contexto de interrupciones. Puede obviar el respaldo de los registros.

Solución: A continuación, se describe lo que hace el robot en cada `step`, incluyendo el puntaje asociado por la correctitud de la descripción:

1. `step_one`: El robot revisa su estado y, en base a eso, evalúa qué acción tomar (**0.5 ptos.**): Si está recién encendido, ejecuta `step_two` (**0.5 ptos.**); si no tiene nada que informar, ejecuta `step_three` (**0.5 ptos.**); si tiene un espacio libre adelante, ejecuta `step_four` (**0.5 ptos.**); en otro caso, *i.e.* tiene una muralla adelante, ejecuta `step_six` (**0.5 ptos.**).

2. `step_two`: El robot inicializa en cero el valor del registro `robot_turns` (**0.5 ptos.**) y continúa la ejecución de `step_three` (**0.5 ptos.**).

3. `step_three`: El robot examina lo que tiene al frente (**1 pto.**) y termina la ejecución de la ISR.

4. `step_four`: El robot revisa si ha girado dos veces (**0.5 ptos.**). Si es el caso (*i.e.* está apuntando en sentido opuesto), ejecuta `step_six` (**0.5 ptos.**); en otro caso, continúa la ejecución de `step_five` (**0.5 ptos.**).

5. `step_five`: El robot reinicia su contador de giros (**0.5 ptos.**); avanza hacia adelante (**0.5 ptos.**) y termina la ejecución de la ISR.

6. `step_six`: El robot aumenta en una unidad su contador de giros (**0.5 ptos.**); realiza un giro para evitar una muralla o para evitar retroceder (**0.5 ptos.**) y termina la ejecución de la ISR.

Pregunta 3: Memoria caché y memoria virtual (8 ptos.)

(a) (6 ptos.) Suponga que, en un instante dado, posee el siguiente estado en una caché de 8 líneas y 2 palabras por línea, cada una de 1 byte:

Línea	Validez	*Timestamp*	*Tag*
0	1	2	5
1	1	7	15
2	0	5	9
3	1	0	1
4	1	3	3
5	1	6	9
6	1	4	5
7	1	1	8

La columna "Línea" representa el índice de línea; "Validez" representa el *valid bit*; "*Timestamp*" representa el tiempo **desde el último acceso a la línea**; y "*Tag*" representa el valor, en base decimal, del *tag* que indica el bloque de memoria almacenado en la línea. Con este estado de caché, se realiza el acceso a memoria a las siguientes direcciones: 0x8F, 0x17, 0x11, 0x3D, 0xF5, 0x16. Indique, para cada acceso, si existe un *hit* o *miss*; si corresponde, la línea de la caché que es modificada; y el *hit-rate* para dos funciones de correspondencia distintas:

1. (3 ptos.) *Directly mapped*.
2. (3 ptos.) *4-way associative*.

Asuma que posee una memoria principal de 256 palabras de 1 byte y que la política de reemplazo utilizada es LRU, si corresponde (*i.e.* se reemplaza la línea con mayor *timestamp*). Para responder, complete las tablas adjuntas al enunciado. El *tag* de cada línea lo puede escribir en base binaria o decimal. No necesita indicar la actualización de *timestamp*, pero sí debe considerar su valor en caso de reemplazos.

Solución: Independiente de la función de correspondencia, al tener una memoria de 256 palabras se necesitan $\log_2(256) = 8$ bits para cada dirección. Por otra parte, al ser las líneas de 2 palabras, se requiere $\log_2(2) = 1$ bit de *offset* para ubicar una palabra dentro de una línea. Con esto en consideración, se tiene el siguiente resultado para cada secuencia de accesos según función:

1. *Directly mapped*: Al tener 8 líneas, se necesitan $\log_2(8) = 3$ bits de índice de línea, resultando en $8 - 3 - 1 = 4$ bits de *tag*. Por otra parte, el *timestamp* es irrelevante ya que no aplican las políticas de reemplazo en esta función, si el contenido no está en la línea a la que pertenece, se reemplaza directamente. A continuación, la tabla de accesos resultante:

N° Acceso	Dirección	*Tag*	*Hit/Miss*	ID Línea modificada	Comentarios
0	0x8F	1000b = 8 = 0x8	*Hit*	-	Índice de línea igual a 111b = 7. El bit de validez es 1 y coinciden los *tags*.
1	0x17	0001b = 1 = 0x1	*Hit*	-	Índice de línea igual a 011b = 3. El bit de validez es 1 y coinciden los *tags*.
2	0x11	0001b = 1 = 0x1	*Miss*	000b = 0	Índice de línea igual a 000b = 0. El bit de validez es 1 pero no coinciden los *tags*.
3	0x3D	0011b = 3 = 0x3	*Miss*	110b = 6	Índice de línea igual a 110b = 6. El bit de validez es 1 pero no coinciden los *tags*.
4	0xF5	1111b = 15 = 0xF	*Miss*	010b = 2	Índice de línea igual a 010b = 2. El bit de validez es 0, por lo que el contenido buscado no está.
5	0x16	0001b = 1 = 0x1	*Hit*	-	Índice de línea igual a 011b = 3. El bit de validez es 1 y coinciden los *tags*.

Hit-rate: $\frac{3}{6} = 0{,}5 = 50\,\%$

2. *4-way associative*: Al tener conjuntos de 4 líneas, se tienen $\frac{8}{4} = 2$ conjuntos y, de esta forma, $\log_2(2) = 1$ bit de índice de conjunto, resultando en $8 - 1 - 1 = 6$ bits de *tag*. En este caso, el conjunto 0 corresponde a las líneas 0-3 y el conjunto 1 a las líneas 4-7. A continuación, la tabla de accesos resultante:

N° Acceso	Dirección	*Tag*	*Hit/Miss*	ID Línea modificada	Comentarios
0	0x8F	100011b = 35 = 0x23	*Miss*	101b = 5	Índice de conjunto igual a 1. No existe coincidencia de *tag*. Se reemplaza la línea de mayor *timestamp*, en este caso, la línea 5. La nueva línea de mayor *timestamp* en el conjunto es la 6.
1	0x17	000101b = 5 = 0x5	*Hit*	-	Índice de conjunto igual a 1. El *tag* coincide con el de la línea 6 y su bit de validez es 1, por lo que hay *hit*. La nueva línea de mayor *timestamp* en el conjunto es la 4.
2	0x11	000100b = 4 = 0x4	*Miss*	010b = 2	Índice de conjunto igual a 0. No existe coincidencia de *tag*. El contenido se escribe en la única línea disponible, que corresponde a la 2. La línea de mayor *timestamp* en el conjunto sigue siendo la 1.
3	0x3D	001111b = 15 = 0xF	*Hit*	-	Índice de conjunto igual a 0. El *tag* coincide con el de la línea 1 y su bit de validez es 1, por lo que hay *hit*. La nueva línea de mayor *timestamp* en el conjunto es la 0.
4	0xF5	111101b = 61 = 0x3D	*Miss*	000b = 0	Índice de conjunto igual a 0. No existen coincidencia de *tag*. Se reemplaza la línea de mayor *timestamp*, en este caso, la línea 0. La nueva línea de mayor *timestamp* en el conjunto es la 3.
5	0x16	000101b = 5 = 0x5	*Hit*	-	Índice de conjunto igual a 1. El *tag* coincide con el de la línea 6 y su bit de validez es 1, por lo que hay *hit*. La línea de mayor *timestamp* en el conjunto sigue siendo la 4.

Hit-rate: $\frac{3}{6} = 0{,}5 = 50\,\%$

Para cada función de correspondencia, se asignan **0.5 ptos.** por acceso correctamente descrito (*i.e.* obtención de *tag*; determinación de *hit* o *miss* y selección de línea a reemplazar). Si existen errores de arraste, se otorgan como máximo **1.5 ptos.** por función si en el desarrollo se evidencia una correcta obtención de bits de *offset*, índice y *tag* por dirección.

(b) (2 ptos.) Suponga que posee un espacio de direcciones virtuales de 20 bits, direcciones físicas de 19 bits y un tamaño de páginas de 128 KiB. En este esquema, un proceso P_i posee el siguiente estado para su tabla de páginas en un instante dado:

Entrada	Validez	Marco
0	0	Disco
1	0	1
2	1	0
3	0	2
4	0	2
5	1	3
6	0	Disco
7	1	1

La columna "Entrada" representa el número de entrada de la tabla; la columna "Validez" representa si el contenido del marco físico es válido; y "Marco" representa el valor, en base decimal, del marco físico asociado a la página ligada a la entrada. Si en esta celda se encuentra el valor "Disco", esto indica que su contenido se encuentra en el *swap file*. Con este estado de tabla de páginas, se realiza el acceso a memoria a las siguientes direcciones virtuales: 0x8F3A0, 0x54168, 0xE07C4, 0x1701A. Indique, para cada acceso, si existe un *page fault* o no. Solo en caso de no existir *page fault* (*i.e.* que la página de la dirección virtual posea un marco físico asociado válido), indique el número de marco y la dirección física resultante. Para responder, se solicita que complete la tabla adjunta al enunciado. Puede escribir la dirección física en base binaria o decimal.

Solución: Al tener un tamaño de páginas de 128 KiB, se tiene un total de $2^7 \times 2^{10} = 2^{17}$ palabras de memoria por página, por lo que se tienen $\log_2(2^{17}) = 17$ bits de *offset*. De esto se deduce que el número de marco físicos es de $19 - 17 = 2$ bits y el número de página es de $20 - 17 = 3$ bits. Este último coincide con las dimensiones de la tabla de páginas, dado que se tienen $2^3 = 8$ entradas. A continuación, la tabla de traducciones resultante:

N° Acceso	Dirección virtual	Número página	¿*Page fault*?	Número marco	Dirección física	Comentarios
0	0x8F3A0	100b = 4	Sí	-	-	*Page fault* por bit de validez.
1	0x54168	010b = 2	No	00b = 0	00b|1b|0x4168 = 00101000001011010000b = 82280 = 0x14168	
2	0xE07C4	111b = 7	No	01b = 1	01b|0b|0x07C4 = 01000000111110001000b = 133060 = 0x207C4	
3	0x1701A	000b = 0	Sí	-	-	*Page fault* por marco en disco.

Se asignan **0.5 ptos.** por acceso correctamente realizado (*i.e.* determinación de *page fault* o traducción de dirección virtual a física). Se acepta expresar la dirección física resultante como la concatenación de bits y dígitos hexadecimales. Si existen errores de arrastre, se otorga como máximo **1 pto.** si en el desarrollo se evidencia una correcta obtención de bits de *offset*, número de página y número de marco físico para cada dirección virtual.

Pregunta 4: Paralelismo a nivel de instrucción (ILP) (8 ptos.)

(a) (4 ptos.) Si bien la arquitectura del computador básico con *pipeline* es lo suficientemente robusta para resolver un conjunto amplio de *hazards*, existe un caso particular que **no está controlado**:

```
MOV A,3
MOV B,0      // Esta instrucción no incide sobre la dependencia de datos.
MOV (dir),A  // La instrucción previa a la anterior modifica el valor de A.
```

Básicamente, no se controla el caso en el que la instrucción previa a la anterior modifica el valor de un registro que **será utilizado para escribir en la memoria de datos**. A partir de este contexto, conteste de forma justificada las siguientes preguntas:

1. (1 pto.) ¿Qué *forwarding unit* debería encargarse de transmitir el valor actualizado de los registros `A`, `B`?

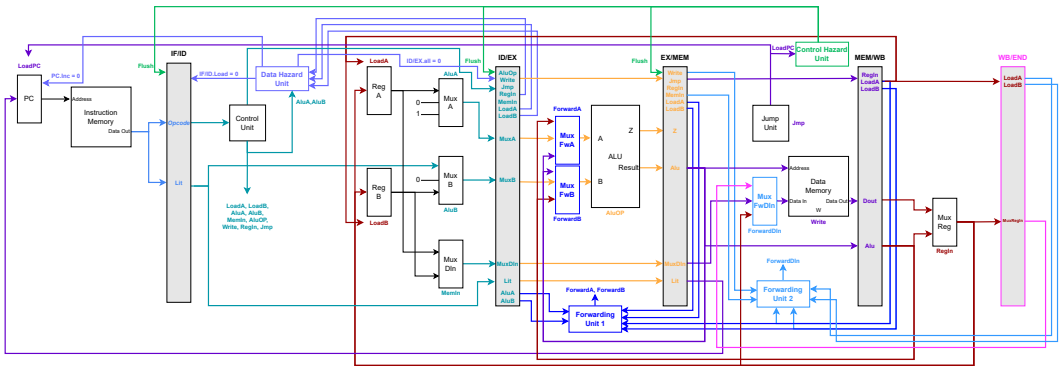
   Solución: Dado que en este caso la escritura se lleva a cabo en la etapa `MEM` y aquí se requieren los registros actualizados, se debe hacer uso de la *forwarding unit* 2 para manejar el *hazard* descrito. Se otorgan **0.5 ptos.** por indicar correctamente la *forwarding unit* y **0.5 ptos.** por justificación. Solo se acepta como respuesta que se señale la *forwarding unit* 1 si es que se utiliza correctamente para resolver el *hazard* según lo solicitado en el siguiente ítem.

2. (3 ptos.) ¿Qué combinación de señales permite detectar el *hazard* anterior? Indíquelas según el registro intermedio o etapa donde se encuentren y modifique el diagrama del computador básico con *pipeline* adjunto para ilustrar tanto la detección como el *forwarding* del valor actualizado. Puede agregar las conexiones y componentes que estime convenientes, pero no debe añadir otra *forwarding unit*.

   Solución: En este caso, contamos con la dificultad de que cuando la instrucción de escritura de memoria se encuentra en la etapa `MEM`, la instrucción previa a la anterior (que modifica el registro) ya concluyó con su ejecución, por lo que sus señales ya no están en ninguno de los registros del *pipeline*. Por lo tanto, si queremos resolver este *hazard* a través de *forwarding*, necesitamos mantener registros de sus señales posterior a su ejecución. Para ello, **agregaremos un nuevo registro intermedio** que llamaremos `WB/END`. Este guardará el valor de las señales `LoadA` y `LoadB` y la salida del componente `Mux RegIn` para que la evaluación de la *forwarding unit* sea la siguiente:

   - `if WB/END.LoadA == 1 AND MEM/WB.LoadA == 0 AND EX/MEM.MemIn == A AND EX/MEM.Write == 1` $\rightarrow$ `ForwardDIn = WB/END.MuxRegIn`
   - `if WB/END.LoadB == 1 AND MEM/WB.LoadB == 0 AND EX/MEM.MemIn == B AND EX/MEM.Write == 1` $\rightarrow$ `ForwardDIn = WB/END.MuxRegIn`

   A continuación, el diagrama resultante de esta implementación:



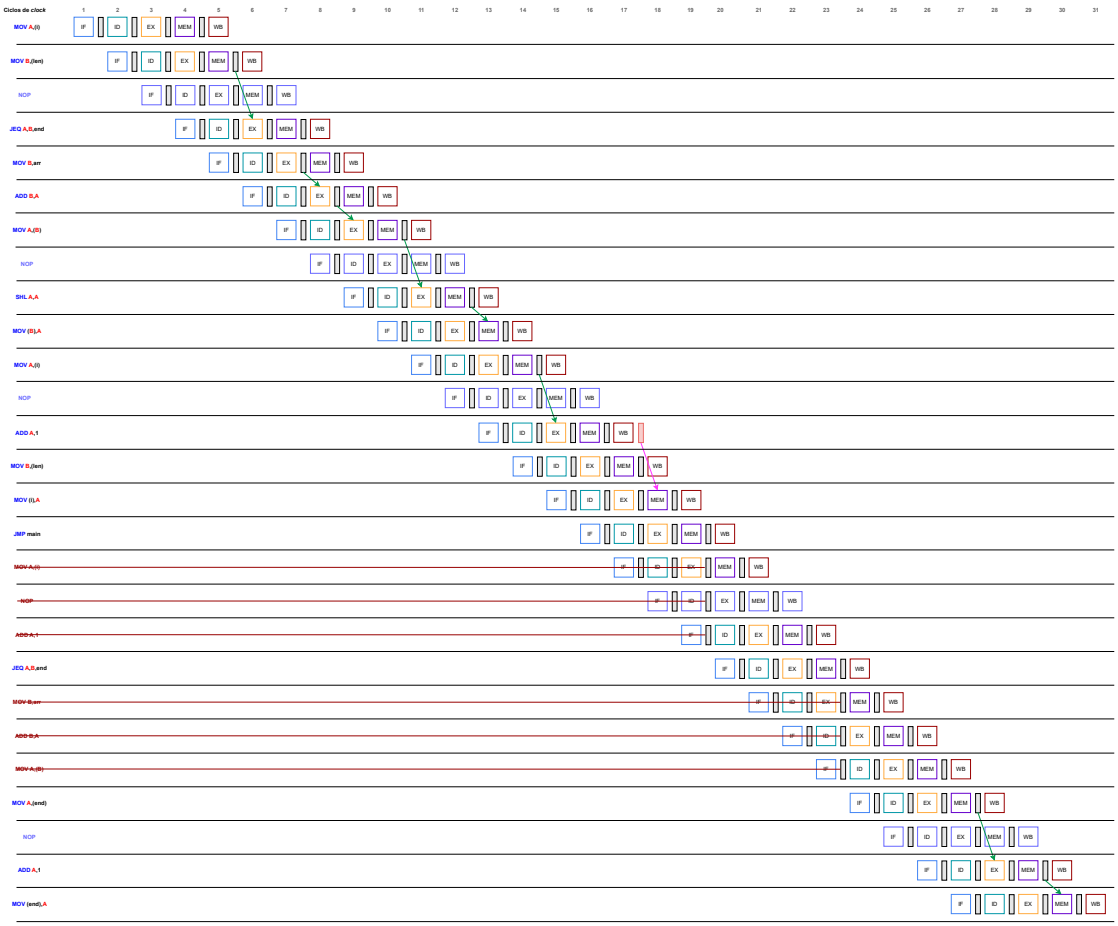
Se otorgan **2 ptos.** por la detección correcta del *hazard* y **1 pto.** por su implementación correcta en el diagrama. Existen muchas formas de implementar la solución, solo se evalúa su correctitud.

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

```
DATA:
  i   0
  arr 11
      13
  len 1
  end 0
CODE:
  MOV A,(i)
  MOV B,(len)
  main:
    JEQ A,B,end // if A == B go to end
    MOV B,arr
    ADD B,A
    MOV A,(B)
    SHL A,A
    MOV (B),A
    MOV A,(i)
    ADD A,1
    MOV B,(len)
    MOV (i),A
    JMP main
  end:
    MOV A,(end)
    ADD A,1
    MOV (end),A
```

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, independiente de que estos sean condicionales o incondicionales. Indique en el diagrama adjunto al enunciado cuando ocurre *forwarding* (con flechas desde el registro hacia la etapa que corresponda), *stalling* y *flushing* (con tachado en las instrucciones *flusheadas*). Asuma que las variables del segmento `DATA` se comienzan a almacenar desde la dirección 0x00.

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 31 ciclos. Se otorga puntaje de la siguiente forma: **0.2 ptos.** por cada *stalling* correctamente detectado, incluyendo el que es *flusheado* (**1 pto.** por todas las detecciones); **0.5 ptos.** por cada *flushing* correctamente detectado (**1 pto.** por todas las detecciones); **0.2 ptos.** por cada *forwarding* correctamente detectado (**1.8 ptos.** por todas las detecciones); y **0.2 ptos.** por la deducción correcta de la cantidad de ciclos en la que corre el programa. Para el *hazard* descrito del ítem anterior, basta con que se señale que existe *forwarding* entre las instrucciones correspondientes, independiente desde qué registro a qué etapa se dibuje su flecha (en esta pauta, se dibuja según la solución planteada). No hay descuento si se incluyen las instrucciones de escritura de memoria correspondientes al segmento `DATA`, pero solo si la ejecución de estas se describe de forma correcta.