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

IIC2343 - Arquitectura de Computadores (I/2026)

Ayudantía 15

Ayudantes: Alberto Maturana (alberto.maturana@uc.cl), Mario Rojas (mario.denzel@estudiante.uc.cl), José Mendoza (jfmendoza@uc.cl)

Pregunta 1: Preguntas conceptuales

(a) Indique qué capacidades/instrucciones gana y pierde el computador básico al agregar soporte para paralelismo a nivel de instrucción.

Solución: Lo más importante que gana el computador básico, lógicamente, es el poder ejecutar múltiples instrucciones en paralelo. Sin embargo:

- Se pierden los llamados a subrutinas. Como el registro SP se elimina y no existe una conexión del PC a la memoria, las subrutinas no se pueden implementar.
- No se pueden hacer instrucciones de salto que impliquen el uso de los bit de estado N, C, O (ya que se eliminan). Notar que además, como se realiza la comparación A - B y el salto en un mismo ciclo, la ALU se conecta directamente a la unidad de salto a través del Reg MEM/EX (separando esa funcionalidad de la unidad de control). Por esto, se elimina también el registro Status, ya que no es necesario.
- Ya no se pueden usar datos de la memoria directamente como parámetros en la ALU, ya que se elimina la conexión entre el Mux B y la salida de memoria. Además, tampoco se puede enviar directamente el resultado de la ALU a la memoria. Lo único permitido es la transferencia de memoria desde y hacia los registros A y B

(b) 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.

(c) 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 realiar solo operaciones de escritura. Indique, justificadamente, si esta modificación introduce algún tipo de error **nuevo** necesario de manejar, no es necesario entregar la 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.

(d) 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 los datos**. A partir de este contexto, conteste de forma justificada las siguientes preguntas:

1. ¿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.

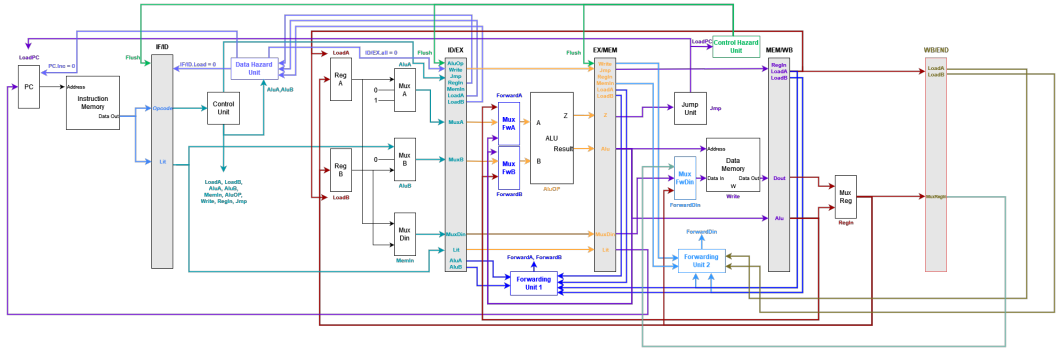
2. ¿Qué combinación de señales permite detectar el *hazard* anterior? Indíquelas según el registro intermedio o etapa donde se encuentren y modifique el diagrama del computador básico con *pipeline* para ilustrar tanto la detección como el *forwarding* del valor actualizado. Puede agregar las conexiones 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.Load.A == 0 AND EX/-MEM.MemIn == A AND EX/MEM.Write == 1 → ForwardDIn =**

WB/END.MuxRegIn

- **if WB/END.LoadB == 1 AND MEM/WB.Load.B == 0 AND EX/-MEM.MemIn == B AND EX/MEM.Write == 1 → ForwardDIn = WB/END.MuxRegIn**

A continuación, el diagrama resultante de esta implementación:



Pregunta 2: Ejecución

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 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: Del *pipeline* a continuación se deduce que el programa termina de correr después de 31 ciclos.

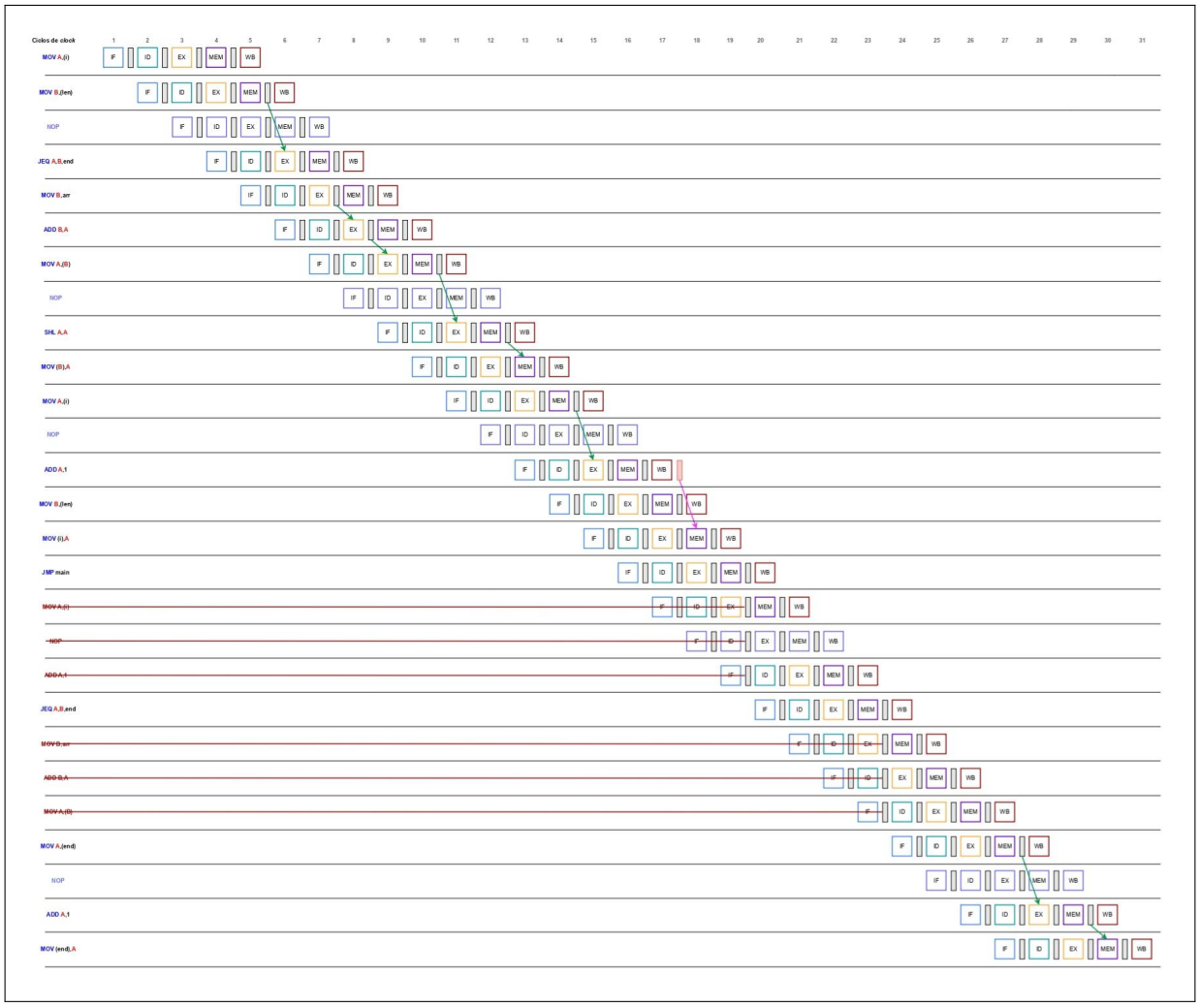
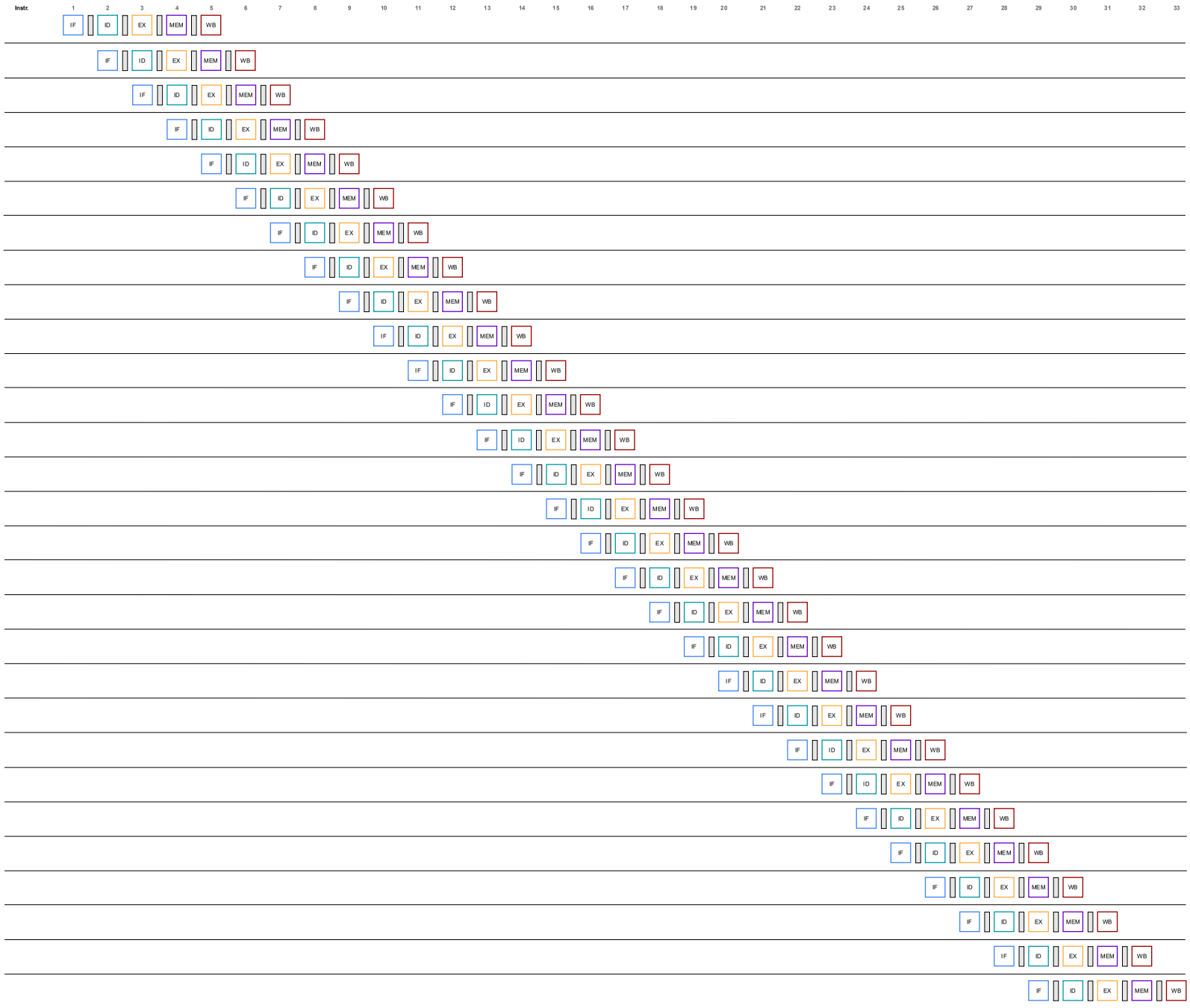

Ciclos de clock
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
MOV A,(i)
MOV B,(ten)
NOP
JEQ A,B,end
MOV B,arr
ADD B,A
MOV A,(B)
NOP
SHL A,A
MOV (B),A
MOV A,(i)
NOP
ADD A,1
MOV B,(ten)
MOV (i),A
JMP main
MOV A,(i)
NOP
ADD A,1
JEQ A,B,end
MOV B,arr
ADD B,A
MOV A,(B)
MOV A,(end)
NOP
ADD A,1
MOV (end),A
IF ID EX MEM WB

Diagrama vacío



Feedback ayudantía

Escanee el QR para entregar feedback sobre la ayudantía.

