Tarea 3 Arquitectura de Computadoras

Integrantes:

Balderrama Domínguez Gael

Hernández Gutiérrez Luis Hiram

Ramírez Ramírez Jehu Jonathan

Yánez Bustamante Francisco

Ejercicio 1

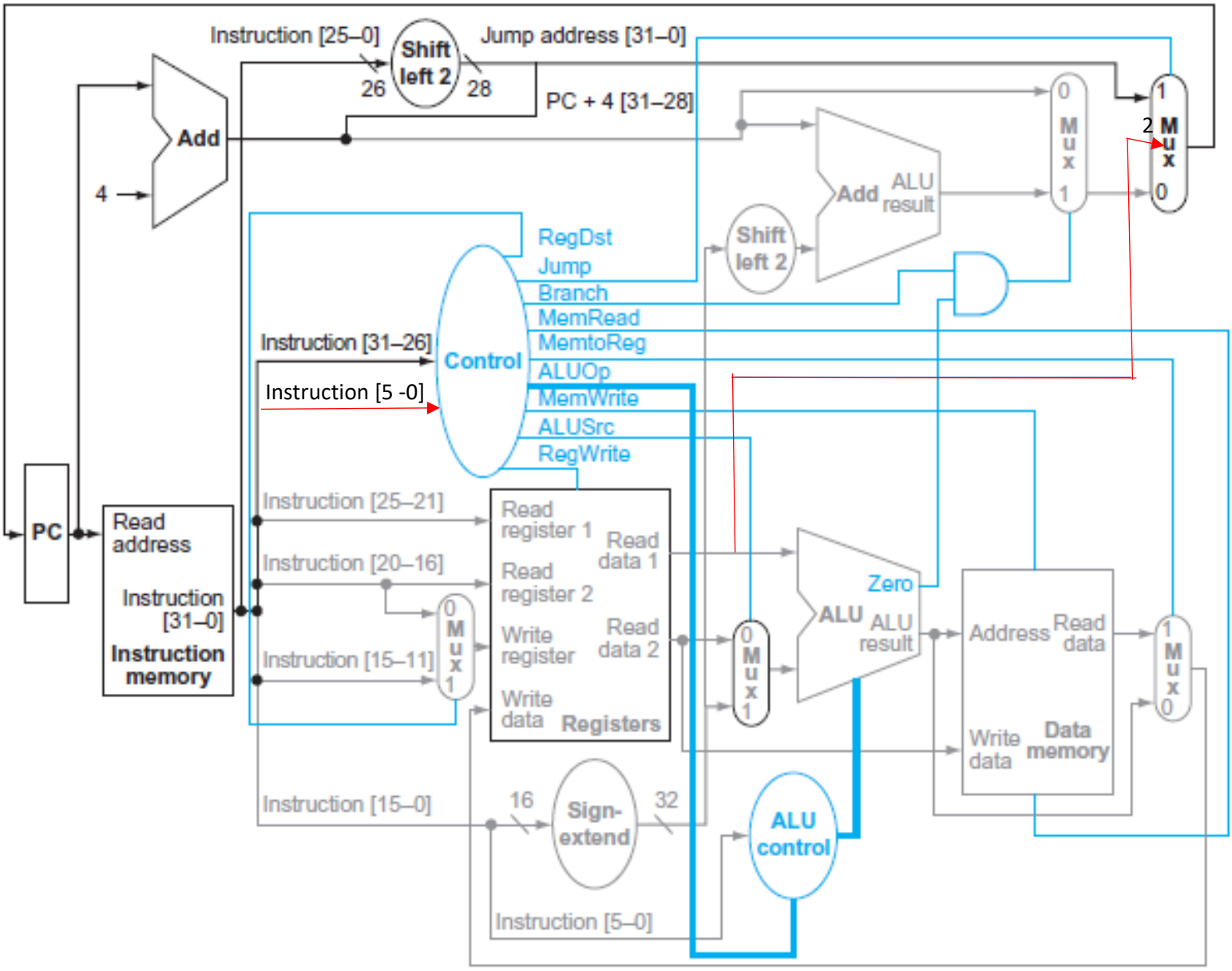


Tabla de señales de control:

RegDest	x
Jump	2
Branch	x
MemRead	x
MemtoReg	x
ALUOp	x
MemWrite	0
ALUSrc	x
RegWrite	0

Ejercicio 2:

a)

Dependencias de datos verdaderas:

1. Dependencia de lectura (RAW) entre la instrucción "sub" y la instrucción "lw":

La instrucción "lw" depende del resultado de la instrucción "sub", ya que utiliza el registro $t3 que es modificado por la instrucción "sub".

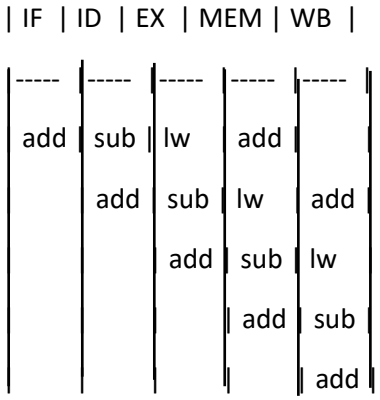
2. Dependencia de escritura (WAW) entre la instrucción "add" (primer) y la instrucción "add" (segundo):

La instrucción "add" (segundo) depende del resultado de la instrucción "add" (primer), ya que ambas intentan escribir en el mismo registro $t6.

b)

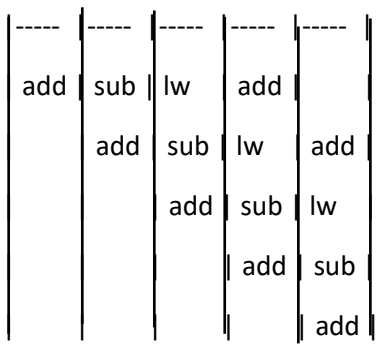
** Algo a notar: dadas las características del código que estamos utilizando viendo que no hay instrucciones que permitan aprovechar al máximo el bypass en P2 los diagramas de pipeline para P1 y P2 serían similares. Ambos tendrían una estructura de 5 etapas y se verían aproximadamente igual . Lo que si cambiará son los tiempos de ciclo, debido a las diferencias en la arquitectura de las dos computadoras.

Diagrama de P1 sin bypass



c)

Diagrama de P2 con bypass



D)

Para P1:

Ciclo de reloj: 100 ps

Cantidad de etapas del pipeline: 5

Cantidad total de instrucciones: 4

Tiempo total de ejecución = Ciclo de reloj * Etapas * Instrucciones = 100 ps * 5 * 4 = 2000 ps

Para P2:

Ciclo de reloj: 120 ps

Cantidad de etapas del pipeline: 5

Cantidad total de instrucciones: 4

Tiempo total de ejecución = Ciclo de reloj * Etapas * Instrucciones = 120 ps * 5 * 4 = 2400 ps

Comparando los tiempos totales de ejecución, vemos que P1 es más rápida que P2.

Speedup se calcula como el tiempo de ejecución de la máquina más lenta dividido por el tiempo de ejecución de la máquina más rápida:

Speedup = Tiempo de ejecución de P2 / Tiempo de ejecución de P1

= 2400 ps / 2000 ps

≈ 1.2

Por lo tanto, la computadora P1 es más rápida y el speedup es aproximadamente 1.2.

(25 puntos) Suponer una CPU superescalar y el siguiente código:

I1: Loop: lw r1, 0(r2)

I2: addi r1, r1, 1

I3: sw r1, 0(r2)

I4: addi r2, r2, 4

I5: sub r4, r3, r2

I6: bnz r4, Loop

a) (10 puntos) Encontrar y escribir todas las dependencias.

Dependencia de datos entre I2 y I3 (RAW) - I2 produce un resultado que I3 necesita.

Dependencia de datos entre I1 y I2 (WAR) - I1 modifica r1, que I2 intenta leer.

Dependencia de datos entre I4 e I5 (RAW) - I4 modifica r2, que I5 intenta leer.

Dependencia de datos entre I3 e I5 (WAW) - I3 escribe en r1, que I5 también intenta escribir.

Dependencia de datos entre I5 e I6 (RAW) - I5 modifica r4, que I6 intenta leer.

b) (15 puntos) Renombrar los registros mediante el algoritmo visto en clase.

Load r1, 0(r2) (I1) - Utiliza r2 como base y produce un valor para r1. No hay conflictos.

Addi r1, r1, 1 (I2) - Aunque modifica r1, no hay conflicto con la instrucción anterior ya que se ha renombrado a un registro diferente, digamos, r1'.

Store r1, 0(r2) (I3) - Aquí, se necesita una unidad de reserva para asegurarse de que la instrucción no se ejecute hasta que la dependencia de datos con la instrucción I1 se haya resuelto. Supongamos que se renombra a un registro r3.

Addi r2, r2, 4 (I4) - No hay conflictos con instrucciones anteriores. Renombramos a r2'.

Sub r4, r3, r2 (I5) - Aquí, r3 está renombrado como r4 y r2 como r2'. No hay conflictos.

Bnz r4, Loop (I6) - Se necesita la unidad de reserva para asegurarse de que se ejecute la instrucción cuando todas las dependencias se hayan resuelto. No hay conflictos directos con instrucciones anteriores

Dado el siguiente código:

lw r2,0(r1)

add r2, r0, r2

label1: lw r3,0(r2)

mul r3, r0, r1

add r1, r3, r1 2

sub r3, r2, r0

beq r2, r1, label1

Suponer que el brinco se toma la primera vez y la segunda vez no. Suponer una CPU escalar con las siguientes características: número ilimitado de estaciones de reserva y memoria para el ROB, grado 3, compromiso sencillo, 2 unidades de ejecución para sumas y restas con latencia de 2 ciclos, una unidad de ejecución para multiplicaciones con latencia de 10 ciclos, en caso de contención por una unidad de ejecución, la instrucción más antigua en orden del programa tiene la preferencia, predicción de brincos no perfecta. Usando el algoritmo de Tomasulo visto en clase, llenar una tabla como la siguiente, indicando en que ciclo pasa cada instrucción por cada etapa.

Instrucción	Issue	EX	WB	Commit
lw r2,0(r1)	1	2	3	4
add r2, r0, r2	2	3	4	5

lw r3,0(r2)	3	4	5	6
mul r3, r0, r1	4	14	15	16
add r1, r3, r1	5	6	7	8
sub r3, r2, r0	6	7	8	9
beq r2, r1, label1	7	8	9	10