Pregunta 1. Rendimiento (PEP 1 – 2019-01)

Parte 1

Se tiene un procesador con una tasa de reloj 2 GHz. Un programa *P* contiene un total de 400 mil millones de instrucciones que en este procesador se ejecutan en 13 min y 20 s (tiempo de CPU despreciando tiempos de acceso a memoria, buses y otros).

A) (0,2 pts.) Calcule el CPI para el programa *P*.

CPI = #ciclos / #instrucciones = tiempo CPU * tasa reloj / # instrucciones
= 800 s * 2E9 s^{-1} / 400E9 = 4

B) (0,2 pts.) Calcule el tiempo de CPU para un programa *Q* que tiene 20% más instrucciones que *P*, y para el que se obtuvo igual CPI.

tiempo CPU = CPI * #instrucciones * tiempo ciclo reloj = 4 * 1,2*400E9 * (1/2) s = 960 s = 16 min

C) (0,3 pts.) Si otro procesador obtiene un CPI de 3,5 para *P*, ¿qué procesador es más rápido para este programa y por cuánto?

tiempo_CPU2 / tiempo_CPU1 = CPI_1 / CPI_2 = 4 / 3,5 =1,14. Es decir, el segundo procesador es 1,14 veces más rápido. Notar que se ha asumido que implementan el mismo ISA y corren a igual tasa de reloj.

A) (0,3 pts.) Considere un nuevo procesador que implementa un conjunto de instrucciones distinto y que tiene una frecuencia de reloj de 3,2 GHz. En este nuevo procesador, *P* compila en muchas menos instrucciones de lenguaje ensamblador, y el tiempo de ejecución baja a 10 min. Con estos datos, ¿se puede calcular el número de instrucciones de *P* para este procesador? Calcule el CPI en estas condiciones (si requiere el número de instrucciones, "N", deje expresado el CPI en términos de "N").

CPI = #ciclos / #instrucciones. Con tiempo y tasa de reloj se puede calcular #ciclos, pero faltan datos para determinar el número de instrucciones; no se indica qué características tiene el nuevo ISA. Entonces CPI = 600 s * 3,2E9 s^{-1} / N

Parte 2

A) (1 pt.) Considere el siguiente programa escrito en lenguaje C. Considere que todas las instrucciones toman, en promedio, 1,5 ciclos en ejecutarse, excepto las de acceso a memoria que demoran, en promedio, 5 ciclos. Calcule el CPI para este programa.

```
int i;
for (i=0; i<1000000; i=i+1)
      A[i] = B[i] + 2
```

Notar que el programa por cada iteración requiere al menos: 1 beq, 1 addi y 1 jump para actualización de índice y manejo del bucle, 1 lw para cargar B[i], 1 add para la suma y 1 sw para guardar resultado en A[i]. O sea, por cada iteración hay 2 instrucciones de acceso a memoria y 4 de otro tipo. Entonces
CPI = (1,5*4 + 5*2)/6 = 16/6. (El número total de iteraciones se factoriza y cancela de numerador y denominador).

Pregunta 2. Rendimiento (PEP 1 – 2018-02)

La empresa de procesadores LETNI ha diseñado un nuevo procesador, el LETNI 8080, que promete revolucionar el mercado. Actualmente, si bien la empresa tiene diseñadores prolijos de hardware, estos desconocen las nociones básicas del rendimiento de los procesadores y no saben cómo obtener sus especificaciones técnicas. En busca de alguien capacitado para ayudarles, se dirigen a la Universidad de Santiago de Chile, pues en esta institución los alumnos aprenden todo lo que LETNI necesita acerca del rendimiento de procesadores. Por recomendación del profesor, usted ha sido seleccionado para obtener las especificaciones técnicas de rendimiento del procesador.

El LETNI 8080 tiene un tiempo de ciclo de reloj de 2/3 ns. Un programa *P* contiene un total de 300 mil millones de instrucciones que en el LETNI 8080 se ejecutan en 16 min y 40 s (tiempo de CPU despreciando tiempos de acceso a memoria, buses y otros).

A) (0,3 pts) Calcule el CPI del LETNI 8080 para el programa *P*.

CPI = #ciclos / #instrucciones = tiempo CPU / tiempo ciclo reloj / # instrucciones
= 1000 s / 2/3E-9 s / 3E11 = 5

B) (0,3 pts) Calcule el tiempo de CPU para un programa *Q* que tiene 18% más instrucciones que *P*, y para el que se obtuvo igual CPI.

tiempo CPU = CPI x #instrucciones x tiempo ciclo reloj = 5 x 118/100 x 3E11 x 2/3E-9 s = 1180 s

C) (0,3 pts) La competencia del LETNI 8080 es el DMA Nolhta, que implementa el mismo conjunto de instrucciones (ISA). El DMA Nolhta corre a la misma tasa de reloj que el LETNI 8080, y entrega un CPI de 6 para *P*. ¿Qué procesador es más rápido para este programa y por cuánto?

tiempo_LETNI / tiempo_DMA = CPI_LETNI x #inst x tiempo reloj LETNI / CPI_DMA x #inst x tiempo reloj_DMA = CPI_LETNI / CPI_DMA = 5 / 6 < 1
Es decir, el LETNI es 6/5 veces más rápido que el DMA.

D) (0,3 pts) Considere el procesador LETNI 8080X, idéntico al 8080 (*i.e.*, mismo ISA), pero con una frecuencia de reloj de 1,7 GHz. Se corre un programa *R* que tiene un 15% menos de instrucciones que *P*, y cuyo tiempo de ejecución en el LETNI 8080X es 11 min. Calcule el CPI del 8080X en estas condiciones.

CPI = 660 s x 17E8 Hz / (85/100 x 3E11) = 4,4

Debido a su excelente desempeño en el trabajo anterior, LETNI le solicita ayuda para comparar tres procesadores de su línea Quantium: el Quantium A, el Quantium B y el Quantium C. Estos tienen frecuencias de reloj de 3, 2,2 y 3,6 GHz, respectivamente.

E) (0,3 pts) Para un programa *T*, los Quantium A, B y C arrojaron un CPI de 1,2, 1 y 1,8, respectivamente. En estas condiciones, ¿qué procesador tiene el mejor rendimiento en millones de instrucciones por segundo?

MIPS = #instrucciones / (tiempoCPUx1E6) = tasa reloj / (CPI x 1E6)
MIPS_A = 3E9 / 1,2E6 = 2500
MIPS_B = 2,2E9 / 1E6 = 2200
MIPS_C = 3,6E9 / 1,8E6 = 2000
¿Es el Quantium A mejor?

F) (0,3 pts) *T* ejecuta un total de 18 mil millones de instrucciones en el Quantium A. Sin embargo, para los Quantium B y C se utilizaron distintas extensiones multimedia del ISA de modo que *T*

ejecuta 13 mil doscientas millones de instrucciones en el Quantium B y 14 mil millones de instrucciones en el Quantium C. Calcule los tiempos de CPU para *T* en cada procesador.

tiempoCPU = CPI x #instrucciones / tasa reloj
tiempoCPU_A = 1,2 x 18E9 / 3E9 = 7,2 s
tiempoCPU_B = 1,0 x 13,2E9 / 2,2E9 = 6 s
tiempoCPU_C = 1,8 x 14E9 / 3,6E9 = 7 s

G) (0,2 pts) ¿En qué orden recomendaría estos tres procesadores? ¿Por qué?

Dados los tiempos de CPU, recomendaría primero B, luego C y finalmente A. Notar que la utilización de extensiones multimedia permitió reducir considerablemente el número de instrucciones y con esto el tiempo total de ejecución.

Pregunta 3 (PEP 1 – 2018-01)

Considere el siguiente código MIPS. Considere que el bucle se ejecuta muchas veces antes de salir. Para determinar el rendimiento bajo esta condición basta con calcular el rendimiento en estado permanente (o estado estable), lo que significa que la influencia de las primeras y de las últimas iteraciones se puede despreciar. Asuma que se dispone de *forwarding* completo y que la predicción de saltos es perfecta.

Código MIPS:

```
I1:    Label:    addi   r4, r4, 4
I2:              lw     r3, 0(r4)
I3:              lw     r2, 4(r4)
I4:              add    r2, r2, r3
I5:              lw     r3, 8(r4)
I6:              add    r2, r2, r3
I7:              sw     r2, –4(r4)
I8:              slt    r1, r4, zero
I9:              bne    r1, zero, Label
```

A) El código se ejecuta en un *pipeline* de 5 etapas. Determine los CPI para las condiciones dadas. (0,7pts)

En estado permanente, el pipeline completa una instrucción por ciclo de reloj, pero un lw seguido de una instrucción de tipo R genera una espera inevitable. Hay dos de estas dependencias en el código: I4(I3) y I6(I5). Como no hay otras esperas, entonces CPI= (9 + 2)/9 = 1.22

B) ¿Se puede reordenar el código para mejorar el rendimiento? Si su respuesta es "No", explique detalladamente porqué. En caso de responder "Sí", entonces proponga un reordenamiento para este código y determine ahora los CPI. Calcule también la aceleración lograda. (0,6pts)

Sí se puede reordenar. La dependencia I4(I3) se puede resolver si se ubica I8 entre medio. Se ahorra una espera. La dependencia I6(I5) no se puede evitar. Luego CPI=(9+1)/9 = 1.11. Aceleración = 9%.

C) Suponga ahora que no se dispone de *forwarding* de ningún tipo. Para el código obtenido en B), determine los CPI. ¿En cuánto se ve afectado el rendimiento? (0,7pts)

código obtenido en b.

```
I1:    Label: addi  r4,r4,4
I2:           lw    r3,0(r4)
I3:           lw    r2,4(r4)
I8:           slt   r1,r4,zero
I4:           add   r2,r2,r3
I5:           lw    r3,8(r4)
I6:           add   r2,r2,r3
I7:           sw    r2,–4(r4)
I9:           bne   r1,zero,Label
```

Las dependencias sin forwarding son:

I2(I1) lw requiere r4 espera addi 1 ciclo
I3(I1) lw requiere r4 espera addi 1 ciclo
I4(I3) add requiere r2 (pero I8 resuelve la espera) 0 ciclo
I6(I5) add requiere r3 espera lw 1 ciclo
I7(I6) sw requiere r2 espera add 2 ciclos

En total se agregan 5 ciclos, 4 debidos a no disponer de forwarding, 1 por el lw en I5.

CPI = (9 + 5)/9 = 1,55

El rendimiento se afecta en un 44%

Pregunta 4. Procesador Monociclo (PEP 1 – 2019-01)

La implementación básica del procesador MIPS monociclo visto en clases (Figura 1), puede ejecutar sólo ciertas instrucciones. Es posible incluir nuevas instrucciones al conjunto de instrucciones (ISA), pero la decisión de hacerlo depende de, entre otras cosas, el costo y la complejidad que se introduce al camino de datos y el control del procesador. Considere las siguientes instrucciones nuevas:

`lwi rt, rd(rs)` → Interpretación: Reg[rt] = Mem[Reg[rd] + Reg[rs]]
`bne rt, rs, BranchAddr` → Interpretación: if(Reg[rt]!=Reg[rs]) PC=PC+4+BranchAddr

A) (0,6 pts.) ¿Qué bloques existentes se pueden utilizar para incluir `lwi`? ¿Y para incluir `bne`?

Para lwi se puede utilizar memoria de instrucciones, archivo de registros con puertos de lectura y escritura, ALU y memoria de datos. Para bne, además de los bloques anteriores, se puede utilizar el sumador para calcular la dirección de salto.

B) (0,6 pts.) ¿Qué nuevos bloques funcionales, si se requieren, necesitamos para `lwi`? ¿Y para `bne`?

Para lwi no se requiere ningún bloque adicional, se puede implementar con los bloques existentes. Para bne se requiere mayor lógica para determinar si los operandos son distintos, y una opción es incluir un circuito inversor del la señal "zero" de la ALU y un multiplexor que escoja entre "zero" y la señal invertida para alimentar el AND con la señal de control "branch".

C) (0,6 pts.) ¿Qué nuevas señales de control, si se requieren, necesitamos para dar soporte a `lwi`? ¿Y para `bne`?

Para lwi no se requieren nuevas señales de control, basta con actualizar la lógica de control. Para bne sí se requiere una señal nueva que permita distinguir entre beq y bne, y una posibilidad es que controle el multiplexor mencionado en B).

D) (0,2 pts.) Después de incluir estas dos nuevas instrucciones, ¿cambia la tasa de reloj de este procesador monociclo? Explique.

La instrucción lwi tiene igual latencia total que lw, y bne sólo agrega un multiplexor y un inversor que no deberían ser parte de la latencia crítica. Además, la instrucción con mayor latencia es lw. Entonces, la tasa de reloj, dada por la mayor latencia entre todas las instrucciones, no cambia.

Pregunta 5 (PEP 1 – 2018-01)

Considere el procesador MIPS monociclo de la **Error! Reference source not found.**. El conjunto de instrucciones (ISA) de este procesador es modificado para agregar una nueva instrucción tipo-I:

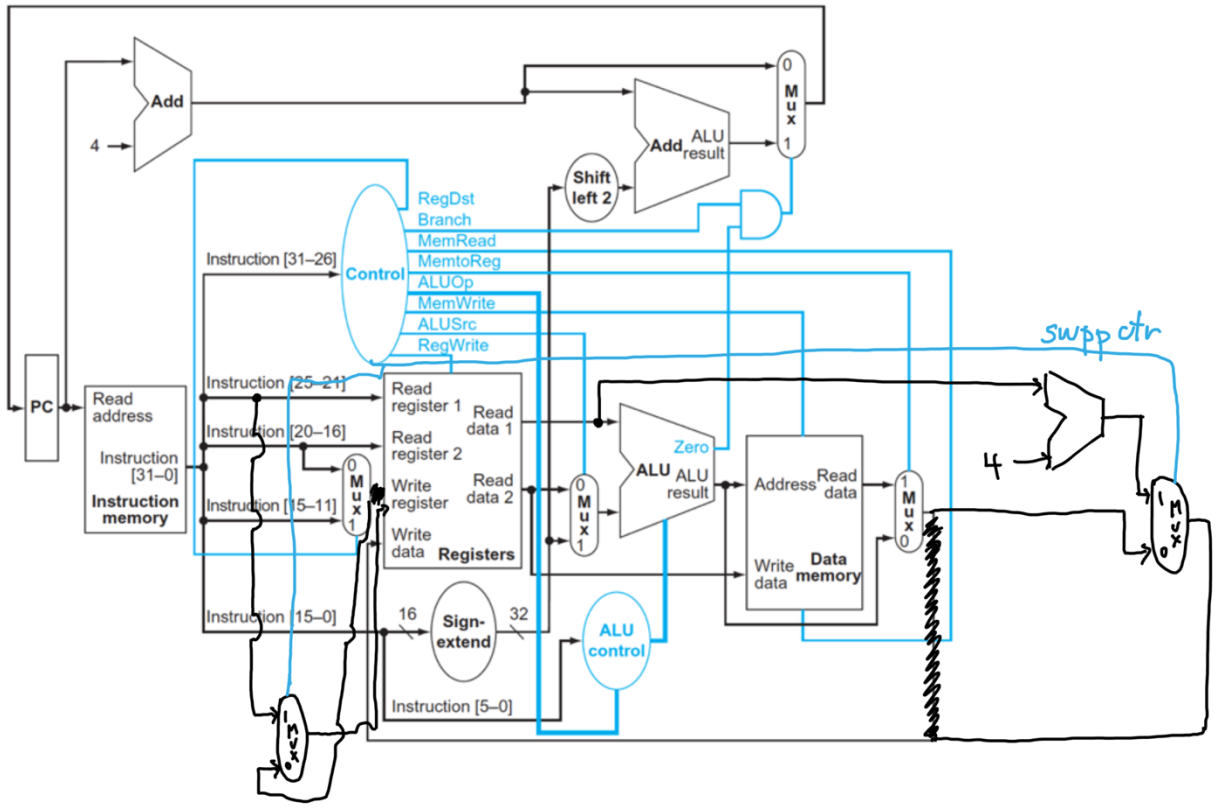
```
swpp $rt, imm($rs)
```

donde `$rt` es el registro que contiene el valor a almacenar, `$rs` el registro de dirección base y `imm` es un valor *offset* de 16 bits. La instrucción almacena en memoria el contenido de `$rt` de igual manera como lo hace `sw`, pero además incrementa el registro de dirección `$rs` de manera de apuntar de manera inmediata a la siguiente palabra en memoria. Esta instrucción es útil si necesitamos almacenar varios elementos en un arreglo de manera rápida. Notar que la instrucción `swpp` es equivalente a dos instrucciones MIPS:

```
sw   $rt, imm($rs)
addi $rs, $rs, 4
```

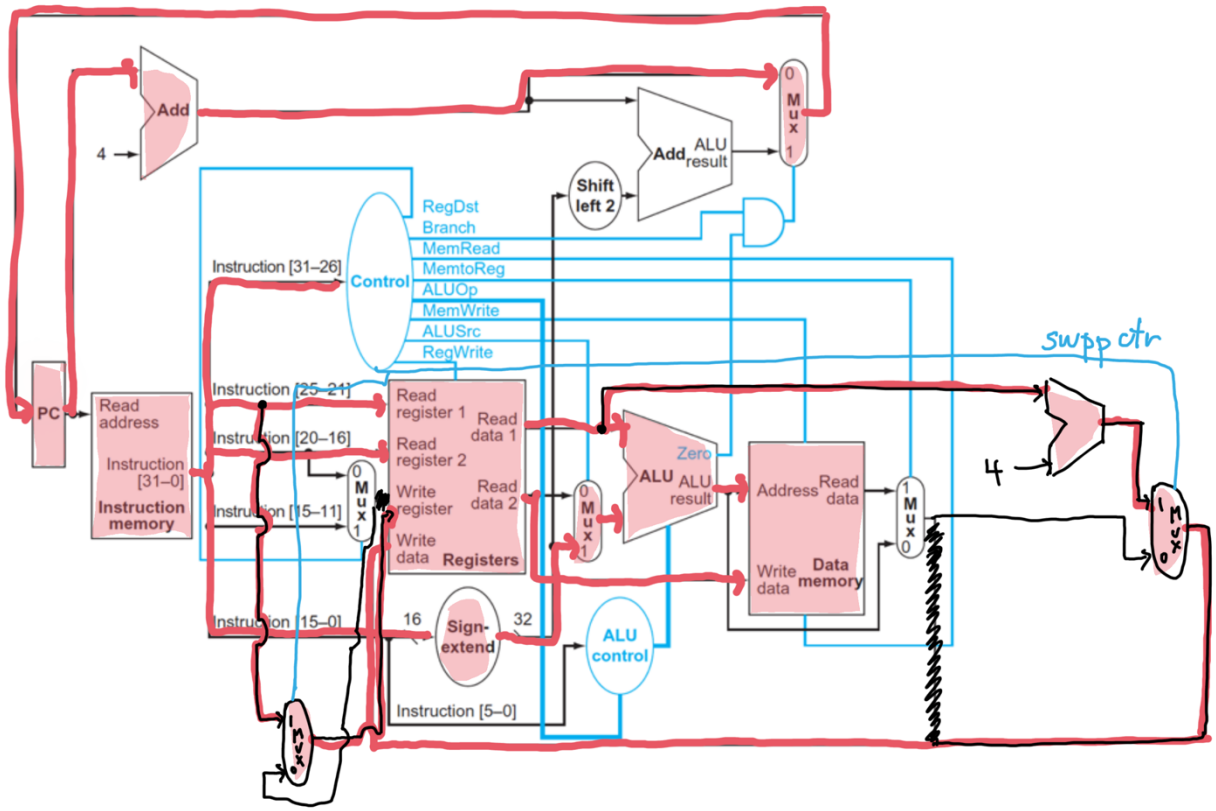
A) Muestre los mínimos cambios, adiciones o eliminaciones de componentes de hardware (*e.g.* multiplexores) e interconexiones que se requieren hacer al camino de datos MIPS monociclo para permitir esta modificación. Incluya las líneas de control que pueda requerir. (0,5pts)

Se requiere una unidad aritmética para sumar 4 a $rs, y luego escribir este resultado en el archivo de registros. Dado que hay otras instrucciones del ISA de MIPS que escriben en el archivo de registros, debemos agregar un multiplexor que determine qué dato se escribirá, i.e., seleccionar entre salida del multiplexor MemtoReg y línea que estamos incorporando para la suma. Además, la dirección del registro a escribir en el archivo de registros, debe poder seleccionarse (multiplexor) entre la línea para otras instrucciones y $rs, que es donde pondremos el resultado de la suma $rs + 4. Finalmente, los dos muliplexores añadidos deben controlarse con una nueva línea de control "swppctr".



B) Marque claramente en su diagrama de la parte A) el camino de datos para `swpp`. (0,5pts)

En el siguiente diagrama, el camino de datos para `swpp` se ha marcado en rojo.



C) Defina los valores de todas las líneas de control para la correcta ejecución de la instrucción `swpp`. Incluya el(los) valor(es) de la(s) línea(s) que haya agregado. Si el valor de una línea es indiferente para esta instrucción, márquelo con una X ("don't care"). (0,5pts)

Similar a `sw`, las líneas de control deben tomar los siguientes valores:

RegDst	X
ALUSrc	1
MemtoReg	X
RegWrite	1
MemRead	0
MemWrite	1
Branch	0
ALUOp	00 (notar que son dos bits)

Además, agregamos una nueva línea de control que debe definirse en 1:

Swppctr	1

D) Suponga que los bloques lógicos del procesador monociclo tienen las siguientes latencias en ps:

I-MEM	Add	Mux	ALU	Regs	D-MEM	Sign-Extend	Shift-Left-2	ALU Ctr
200	70	20	90	90	250	15	10	30

Determine las latencias de las instrucciones `sw`, `addi` y `swpp`. Considerando solo estas 3 instrucciones, ¿cuál sería la tasa de reloj máxima de este procesador? (0,5pts)

`sw`: I-MEM+Regs+Mux+ALU+D-MEM+Mux= 650 ps
`addi`: I-MEM+Mux+Regs+Mux+ALU+Mux = 440 ps

`swpp`: I-MEM+Mux+Regs+Mux+ALU+Add+Mux+D-MEM= 740 ps

Si consideramos solo estas 3 instrucciones, la latencia crítica está dada por los 740 ps de swpp. Entonces la tasa de reloj máxima sería 1000/740 GHz

Pregunta 6. Procesador Monociclo (PEP 1 – 2018-02)

En el diseño de procesadores, para considerar una posible mejora en el camino de datos del procesador, la decisión muchas veces depende del compromiso entre costo y desempeño. Considere el procesador monociclo de la Figura y las siguientes latencias, en ps, y costos, en $, de los bloques que se indican:

	I-Mem	Add	Mux	ALU	Regs	D-Mem	Control
Latencia (ps)	400	100	30	120	200	350	100
Costo ($)	1000	30	10	100	200	2000	500

Considere la adición de un multiplicador a la ALU. Esta potencial mejora agregará 300 ps a la latencia de la ALU, e incrementará su costo en $600. Agregar el multiplicador significará una reducción del 25% del número total de instrucciones, pues ya no será necesario emular la instrucción MUL.

A) (0,5 pts.) Indique qué función cumple cada uno de los dos bloques sumadores etiquetados como "*Add*" en el diagrama de la **Error! Reference source not found.**. ¿Participan en el camino de datos crítico que determina la tasa de reloj de este procesador?

El sumador de la izquierda incrementa el contador de programa, PC, en 4 para obtener la instrucción siguiente, y el de la derecha, le suma al PC el offset de un salto. Estos sumadores no contribuyen al camino de datos crítico, pues sus latencias son menores que aquellas de Regs y ALU, y no interfieren en los cálculos aritméticos o de direcciones de las instrucciones que toman mas tiempo.

B) (0,5 pts.) ¿Cuál es la tasa de reloj de este procesador monociclo con y sin esta mejora?

El camino crítico está dado por una instrucción de acceso a memoria: I-Mem + Regs (toma mas tiempo que Control) + Mux (seleccionar entrada a ALU) + ALU + D-Mem + Mux (seleccionar valor de memoria para ser escrito en registros). Entonces, sin mejora se tiene una latencia total de 1130 ps (0,88 GHz), y con la mejora, 1430 ps (0,7 GHz) (como la ALU está en el camino crítico, simplemente se suman los 300 ps adicionales de latencia de la ALU en esta condición).

C) (0,5 pts.) ¿Cuál es la aceleración lograda al incorporar esta mejora?

Con la mejora, el reloj es mas lento, pero se requieren 25% menos ciclos en total para el programa. La aceleración es (1/0,75)*(1130/1430) = 1,054. Es decir, el procesador con la mejora sería 5,4% mas rápido.

D) (0,5 pts.) Compare la razón costo/desempeño con y sin esta mejora, y concluya sobre la conveniencia de introducir esta mejora.

Del diagrama del procesador monociclo, y considerando todos los elementos (no solo aquellos del camino crítico), tenemos un costo total de $3890 sin la mejora, y de $4490 con la mejora. El costo relativo es entonces 1,15, y la razón costo/desempeño, 1,15/1,054 = 1,09. Es decir, conviene introducir la mejora pues pagamos mas, pero obtenemos un desempeño comparativamente mejor por el costo adicional.

Pregunta 7. Pipeline (PEP 1 – 2018-01)

Se tienen 3 procesadores *pipeline*, uno de 5 etapas, uno de 6 etapas y uno de 7 etapas. Dado el siguiente código y las latencias de las etapas de cada procesador entregadas en ps, responda lo siguiente.

```
lw   $t1, 0($sp)
add  $t0, $t0, $t2
addi $t2, $zero, 3
sub  $t3, $t2, $t1
sw   $t2, 0($sp)
addi $s0, $t2, 1
addi $t0, $t1, 3
lw $t0, 4($sp)
```

Tabla 1: Latencias del procesador de 5 etapas

IF	ID	EX	MEM	WB
40	60	150	180	90

Tabla 2: Latencias del procesador de 6 etapas

IF	ID	EX	MEM-1	MEM-2	WB
55	60	150	90	100	95

Tabla 3: Latencias del procesador de 7 etapas

IF	ID	EX-1	EX-2	MEM-1	MEM-2	WB
45	40	60	80	100	90	100

A) ¿Cuál procesador ejecutaría más rápido el código? (0,2pts)

B) ¿Cuál procesador se demoraría más en ejecutarlo? (0,2pts)

C) Si el procesador de 5 etapas fuera monociclo, ¿cuánto demoraría la ejecución del código? (0,2pts)

Ciclo	IF	ID	EX	MEM	WB
1	LW $t1,0($sp)	-	-	-	-
2	add $t0,$t0,$t2	LW $t1,0($sp)	-	-	-
3	addi $t2, $zero, 3	add $t0,$t0,$t2	LW $t1,0($sp)	-	-
4	addi $t2, $zero, 3	addi $t2, $zero, 3	add $t0,$t0,$t2	LW $t1,0($sp)	-
5	addi $t2, $zero, 3	addi $t2, $zero, 3	NOP	add $t0,$t0,$t2	LW $t1,0($sp)
6	sub $t3, $t2,$t1	addi $t2, $zero, 3	NOP	NOP	add $t0,$t0,$t2
7	sw $t2,0($sp)	sub $t3, $t2,$t1	addi $t2, $zero, 3	NOP	NOP
8	addi $t0,$t2,1	sw $t2,0($sp)	sub $t3, $t2,$t1	addi $t2, $zero, 3	NOP
9	addi $t0,$t1,3	addi $t0,$t2,1	sw $t2,0($sp)	sub $t3, $t2,$t1	addi $t2, $zero, 3
10	Lw $t0, 4($sp)	addi $t0,$t1,3	addi $t0,$t2,1	sw $t2,0($sp)	sub $t3, $t2,$t1
11	-	Lw $t0, 4($sp)	addi $t0,$t1,3	addi $t0,$t2,1	sw $t2,0($sp)
12	-	-	Lw $t0, 4($sp)	addi $t0,$t1,3	addi $t0,$t2,1
13	-	-	-	Lw $t0, 4($sp)	addi $t0,$t1,3
14	-	-	-	-	Lw $t0, 4($sp)

	Nro ciclos	Tiempo ciclo	Demora
5 fases	14	180	2520
6 fases	16	150	2400
7 fases	18	100	1800

Vemos que:

A) El que menos demora es el procesador de 7 fases

B) El que mas demora es el procesador de 5 fases

C) Si fuera monociclo, demoraría 520 ns en un ciclo, y 4160 ns en ejecutar el código

Pregunta 8. Pipeline (PEP 1 – 2018-02)

Parte 1 de 2

Considere la siguiente secuencia de instrucciones que se ejecutan en un procesador MIPS con *pipeline* de 5 etapas (Figura).

```
add   $t0, $t0, $t1
add   $t0, $t0, $t1
add   $t0, $t0, $t2
```

Inicialmente, los registros `$t0, $t1` y `$t2` almacenan, respectivamente, los valores 0, 10 y 20. El procesador tiene una unidad de adelantamiento (*forwarding unit*) cuya lógica de detección de riesgos y manejo de señales de control es la siguiente:

if (EX/MEM.RegWrite and (EX/MEM.RegisterRd ≠ 0) and (EX/MEM.RegisterRd = ID/EX.RegisterRs)) ForwardA = 10

if (EX/MEM.RegWrite and (EX/MEM.RegisterRd ≠ 0) and (EX/MEM.RegisterRd = ID/EX.RegisterRt)) ForwardB = 10

if (MEM/WB.RegWrite and (MEM/WB.RegisterRd ≠ 0) and (MEM/WB.RegisterRd = ID/EX.RegisterRs)) ForwardA = 01

if (MEM/WB.RegWrite and (MEM/WB.RegisterRd ≠ 0) and (MEM/WB.RegisterRd = ID/EX.RegisterRt)) ForwardB = 01

A) (0,5 pts) Explique qué problema tiene esta implementación de adelantamiento de datos y determine los valores que almacenan `$t0, $t1, $t2` después de completarse la ejecución de las tres instrucciones.

El problema es que para el *tercer* "add" se adelantará el resultado de la operación del *primer* "add" (desde el registro MEM/WB) pues en la lógica de la unidad de adelantamiento se detectó el riesgo de datos entre estas dos instrucciones. Así los valores almacenados al terminar la ejecución de los tres "add" serán: $t0=30, $t1=10 y $t2=20.

B) (0,3 pts) Explique brevemente cómo resolvería el problema descrito en A).

Para resolver el problema, habría que modificar la lógica de la unidad de adelantamiento para adelantar el resultado almacenado en el registro EX/MEM, que es el más reciente, en vez de aquel del registro MEM/WB. Concretamente, habría que agregar la siguiente condición al primer "if" (y condiciones similares a los demás):
and not(EX/MEM.RegWrite and (EX/MEM.RegisterRd ≠ 0) and (EX/MEM.RegisterRd ≠ ID/EX.RegisterRs))
*No es necesario explicitar esta condición en la respuesta, basta con una explicación descriptiva.

Parte 2 de 2

Considere la siguiente secuencia de instrucciones que se ejecuta en un procesador MIPS con *pipeline* de 5 etapas (IF, ID, EX, MEM y WB). `A` y `B` son constantes, y el valor inicial del registro `$t0` es 1. Asuma un esquema de predicción de bifurcaciones "*branch not taken*" o bifurcación no tomada, y que la decisión de bifurcación se toma en la etapa EX.

```
FOR:  beq   $t0, $zero, NEXT
      lw    $t1, A($s0)
      lw    $t2, B($s0)
      add   $t3, $t1, $t2
      sw    $t3, A($s0)
      addi  $t0, $t0, -1
      j     FOR
NEXT: addi  $t0, $t0, 1
      sw    $t0, A($s0)
```

A) (0,3 pts) Muestre los riesgos (*hazards*) de datos y de control que existen en este programa.

Hay un riesgo de control en la línea 1 (rc1).
Riesgos de datos: entre líneas 2 y 4 (rd1), entre líneas 3 y 4 (rd2), entre líneas 4 y 5 (rd3), entre líneas 6 y 1 (rd4; después del salto) y entre líneas 8 y 9 (rd5).

B) (0,3 pts) Asumiendo un procesador sin adelantamiento (*forwarding*), muestre donde y cuantas esperas (burbujas) son necesarias agregar para eliminar los riesgos detectados.

Sin adelantamiento, rd1 y rd4 requieren una burbuja; rd2, rd3 y rd5, dos.
En caso de una predicción incorrecta, rc1 se resuelve con dos burbujas (y "flush" de las dos "lw" que venían detrás) pues la decisión de bifurcación se toma en etapa EX.

C) (0,4 pts) Considere ahora un procesador con adelantamiento. ¿Se requiere agregar esperas para resolver los riesgos de datos y/o control? Dibuje un diagrama de ciclos de reloj del *pipeline* (ciclos avanzan de izquierda a derecha) para esta secuencia de instrucciones.

Para una predicción incorrecta, el rc1 de todas maneras requiere burbujas, independiente de la existencia de adelantamiento.
Para los riesgos de datos, rd1, rd3, rd4 y rd5 pueden resolverse con adelantamiento sin necesidad de burbujas. Sin embargo, dado que "lw" y "add" son consecutivas y el dato de "lw" no está disponible sino hasta la etapa MEM, rd2 no puede resolverse sólo con adelantamiento y se requiere una burbuja.

Inst. \ CR	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
beq	IF	ID	EX	ME	WB												
lw		IF	ID	EX	ME	WB											
lw			IF	ID	EX	ME	WB										
add→nop				IF	ID	***	***	***									
add						ID	EX	ME	WB								
sw					IF	IF	ID	EX	ME	WB							
addi							IF	ID	EX	ME	WB						
j								IF	ID	EX	ME	WB					
beq									IF	ID	EX	ME	WB				
lw→nop										IF	ID	***	***	***			
lw→nop											IF	***	***	***	***		
addi												IF	ID	EX	ME	WB	

sw													IF	ID	EX	ME	WB

En este esquema, *** representa una burbuja. Notar que la burbuja se mueve por el pipeline, y por eso se repite horizontalmente a medida que avanzan los ciclos de reloj. Sin embargo, esto no significa que aparezca una nueva burbuja. En cambio, más burbujas aparecen en esta secuencia al cancelar la predicción del beq, y éstas también se propagan por el pipeline. En total, se producen tres burbujas para este programa cuando se utiliza adelantamiento.

D) (0,2 pts) ¿Cuánto mejora el rendimiento al incluir adelantamiento?

Con adelantamiento, se requieren 17 ciclos de reloj para completar esta secuencia de instrucciones, mientras que, sin adelantamiento, como habría siete burbujas más, se requieren 24 ciclos para la misma secuencia. El rendimiento mejora entonces en 1,41 veces.

Pregunta 9. Pipeline (PA – 2018-01)

Considere el siguiente bucle en MIPS para un procesador con un *pipeline* de 5 etapas:

```
loop:   lw    r1, 0(r1)
        and   r3, r1, r2
        lw    r1, 0(r1)
        lw    r1, 0(r1)
        beq   r1, r0, loop
```

Asuma que se tiene predicción de saltos perfecta, es decir, no hay esperas por *hazards* de control, y que el *pipeline* dispone de *forwarding* completo. Además, suponga que se ejecutan muchas iteraciones de este bucle antes de que el bucle termine.

A) (0,5 pts.) Indique qué tipo de instrucción (R, I o J) es cada una de las instrucciones del bucle.
lw→I, and→R, beq→I

B) (0,5 pts.) Muestre un diagrama de ejecución del *pipeline* para la tercera iteración de este bucle, desde el ciclo en el cual se puede hacer *fetch* de la primera instrucción de esa iteración, hasta (pero no incluyendo) el ciclo en el cual se puede hacer *fetch* de la primera instrucción de la iteración siguiente. Muestre todas las instrucciones que están en el *pipeline* durante estos ciclos (no solo aquellos desde la tercera iteración).

```
LW  R1,0(R1)      WB
LW  R1,0(R1)      EX  MEM WB
BEQ R1,R0,Loop    ID  *** EX  MEM WB
LW  R1,0(R1)      IF  *** ID  EX  MEM WB
AND R1,R1,R2            IF  ID  *** EX  MEM WB
LW  R1,0(R1)                IF  *** ID  EX  MEM
LW  R1,0(R1)                        IF  ID  ***
BEQ R1,R0,Loop                          IF  ***
```

C) (0,5 pts.) ¿En qué porcentaje de los ciclos se tiene que las cinco etapas del *pipeline* están haciendo trabajo útil? (por ejemplo, la instrucción "`add r3, r1, r2`" no realiza trabajo útil durante la etapa MEM).
En el diagrama anterior, las etapas marcadas en celeste no realizan trabajo útil (notar que beq está haciendo trabajo útil en MEM pues está determinando el valor correcto de la siguiente instrucción en esa etapa). En todos los ciclos hay o bien etapas en espera o sin realizar trabajo útil y, por lo tanto, en el 0% de los ciclos se tienen todas las etapas del *pipeline* haciendo trabajo útil.

D) (0,5 pts.) Determine el rendimiento medido en CPI para estas condiciones.
Las 5 instrucciones del bucle son ejecutadas en 8 ciclos de reloj. Entonces, CPI = 8/5.

Pregunta 10. Pipeline (PEP 1 – 2019-01)

Parte 1

Considere la siguiente secuencia de instrucciones que se ejecuta en un procesador MIPS con *pipeline* de 6 etapas, similar a MIPS con *pipeline* de 5 etapas, pero en el que IF se ha separado en dos etapas. Es decir, las etapas del *pipeline* de este procesador son: IF1, IF2, ID, EX, MEM y WB. Los valores iniciales de los registros `$t1` y `$t2` son 1 y 0, respectivamente. Asuma un esquema de predicción de bifurcaciones "*branch not taken*" o bifurcación no tomada, que la decisión de bifurcación se toma en la etapa ID, y que la dirección de destino de un salto incondicional está inmediatamente disponible en IF1.

```
1     FOR:  beq   $t1, $t2, NEXT
2           lw    $t3, 0($s0)
3           lw    $t4, 0($s0)
4           sub   $t5, $t4, $t3
5           addi  $t2, $t1, 1
6           j     FOR
7     NEXT: add   $t0, $t2, $t5
```

E) (0,3 pts.) Muestre los riesgos (*hazards*) de datos y de control que existen en este programa.

Hay un riesgo de control en la línea 1 (rc1).
Riesgos de datos: entre líneas 2 y 4 (rd1), entre líneas 3 y 4 (rd2) y entre líneas 5 y 1 (rd3).

F) (0,2 pts.) Asumiendo un procesador **sin** adelantamiento (*forwarding*), indique cuantas esperas (burbujas) son necesarias agregar para eliminar cada uno de los riesgos identificados. **No** es necesario mostrar un diagrama de ciclos de reloj en esta parte.

Sin adelantamiento, rd1 y rd3 requieren una burbuja, y rd2, dos.
En caso de una predicción incorrecta, rc1 se resuelve con dos burbujas (y "flush" de las dos "lw" que venían detrás) pues la decisión de bifurcación se toma en etapa ID.

G) (0,2 pts.) Considere ahora un procesador **con** adelantamiento. ¿Cambian sus respuestas en B)? Y en caso afirmativo, ¿en cuánto?

Sí, ahora sólo se requiere una burbuja para rd2, y ninguna para rd1 y rd3. Rc1 sigue igual.

H) (0,7 pts.) Dibuje un diagrama de ciclos de reloj del *pipeline* (ciclos avanzan de izquierda a derecha) para esta secuencia de instrucciones considerando adelantamiento.

Inst. \ CR	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
beq	IF1	IF2	ID	EX	ME	WB										
lw		IF1	IF2	ID	EX	ME	WB									
lw			IF1	IF2	ID	EX	ME	WB								
sub→nop				IF1	IF2	ID	***	***	***							
sub							ID	EX	ME	WB						
addi					IF1	IF2	IF2	ID	EX	ME	WB					
j						IF1	IF1	IF2	ID	EX	ME	WB				
beq								IF1	IF2	ID	EX	ME	WB			
lw→nop									IF1	IF2	***	***	***	***		
lw→nop										IF1	***	***	***	***	***	
add											IF1	IF2	ID	EX	ME	WB

Parte 2

A) (0,6 pts.) Encuentre una expresión para calcular el rendimiento, medido en CPI, de un procesador con *pipeline* de "*N*" etapas (*N>6*) que ejecuta un programa de "*I*" instrucciones en el que se detectan "B_k" riesgos de datos que se resuelven con "*k*" burbujas. Considere $B_k = I/(2k)$ para $i = \{1, 2, 3, 4, 5, 6\}$ y $B_k = 0$ en cualquier otro caso.

CPI = #ciclos/#instr = (I + N-1 + sum(Bk*k)) / I = (I + N-1 + 6*I/2) / I = 4 + (N-1)/I

ORGANIZACIÓN DE COMPUTADORES
SEMESTRE DE VERANO 2018
PEP1 – Solución Propuesta

Profesor: Néstor González Valenzuela
Fecha: 5 de enero de 2018
Tiempo Disponible: 100 minutos

Datos a usar:

Instrucciones tipo A, de cálculo:	1 ciclo
Instrucciones tipo B, de load/store:	2 ciclos
Instrucciones tipo C, de salto:	3 ciclos
Velocidad de procesador:	3 GHz

Código MIPS:

```
        add   $r2, $r0, $r0
otra:   beq   $r2, $r8, fin
        add   $r3, $r2, $r9
        lw    $r4, 0($r3)
        sw    $r4, 1($r3)
        addi  $r2, $r2, 2
        beq   $r0, $r0, otra
fin:
```

Pregunta 1.- Materia: Performance. (2,0 puntos en total)
Considere los siguientes datos iniciales para el código MIPS: r0 = 0 ; r8 = 6.
Si necesita otro valor para sus respuestas, defínalo usted mismo(a).
Con la información dada:

a. Determine el tiempo de ciclo para este procesador. (0,2 ptos.)

R: La velocidad del procesador es de 3Ghz = 3 x 10^9 Hz. Como 1 Hz es 1 ciclo/seg, luego el ciclo de reloj o tiempo de ciclo es de 1/3 x 10^{-9} seg, ó 0,33 ns [ns = nano segundos], ó 330 ps.
(Nota: el libro P&H expresa los tiempos en ps [pico segundos]

b. Determine la cantidad de instrucciones de cada tipo A, B, C para la ejecución completa del código. (0,2 ptos.)

R: Con r0 = 0 ; r8 = 6 se ejecutan

Inicio y primer ciclo: r0=0				(r2=2)					(r2=4)				
add	$r2, $r0, $r0 (r2=0)		A	add	$r2, $r0, $r0		C		add	$r2, $r0, $r0			
otra:	beq	$r2, $r8, fin	C	otra:	beq	$r2, $r8, fin	A		otra:	beq	$r2, $r8, fin	C	
	add	$r3, $r2, $r9	A		add	$r3, $r2, $r9	B			add	$r3, $r2, $r9	A	
	lw	$r4, 0($r3)	B		lw	$r4, 0($r3)	B			lw	$r4, 0($r3)	B	
	sw	$r4, 1($r3)	B		sw	$r4, 1($r3)	A			sw	$r4, 1($r3)	B	
	addi	$r2, $r2, 2	A		addi	$r2, $r2, 2	C			addi	$r2, $r2, 2	A	
	beq	$r0, $r0, otra	C		beq	$r0, $r0, otra				beq	$r0, $r0, otra:	C	
fin:		(r2=2)		fin:		(r2=4)			fin:		(r2=6)	C	

Se ejecutan 7 instrucciones de tipo A, 6 instrucciones de tipo B y 7 instrucciones de tipo C. En total 20 instrucciones.

c. Determine la frecuencia correspondiente a cada tipo de instrucción. (0,1 pto.)

R: Tipo A: 7/20 = 35%; Tipo B: 6/20 = 30%; Tipo C = 7/20 = 35%

d. Calcule los CPI para cada tipo de instrucción. (0,2 ptos.)

R: Los CPI de cada tipo de instrucción son los datos dados.
CPI Instrucciones Tipo A:1; CPI Instrucciones CPI Tipo B: 2; CPI Instrucciones Tipo 3: 3

e. Calcule los CPI para el programa completo. (0,2 ptos.)

$$CPI = \frac{7x1 + 6x2 + 7x3}{20} = 2$$

o bien:

$$CPI = 1x0{,}35 + 2x0{,}3 + 3x0{,}35 = 2$$

f. Asuma que se pueden hacer dos mejoras. Mejora1: disminuir un ciclo de las instrucciones de tipo B; y Mejora2, disminuir un ciclo de las instrucciones de tipo C. Para las condiciones del código dado y usando ley de Amdahl ¿Cuál mejora recomienda? (0,5 ptos)

$$Ley\ de\ Amdahl => A = \frac{1}{(1 - Fm) + {Fm}/{Am}}$$

De los datos dados:
Mejora 1: Fracción de mejora=12/40= 0,3; mejora = 1 ciclo de 2 => mejora el doble = 2

$$A_1 = \frac{1}{(1 - 0{,}3) + {0{,}3}/{2}} = \frac{1}{0{,}7 + 0{,}15} = \frac{1}{0{,}85} = 1{,}176$$

Luego, mejora1 = 17.6%

Mejora 2: Fracción de mejora=21/40 = 0,525; mejora = 1 ciclo de 3 => mejora = 3/2

$$A_1 = \frac{1}{(1-0{,}525) + {}^{0{,}525}/_{1{,}5}} = \frac{1}{0{,}475 + 0{,}35} = \frac{1}{0{,}825} = 1{,}212$$

Luego, mejora 2 = 21,2%

Confirmación calculando los CPI

$$CPI\ mejora\ 1 = 1x0{,}35 + 1x0{,}3 + 3x0{,}35 = 1{,}70$$

Speedup = 2/1,7 = 1,176 => 17,6%

$$CPI\ mejora\ 2 = 1x0{,}35 + 2x0{,}3 + 2x0{,}35 = 1{,}65$$

Speedup = 2/1,65 = 1,212 => 21,2%

Luego: se recomienda la mejora 2.-

g. Suponga que cada tipo de mejora tiene un costo de implementación ¿Puede calcular cuál sería la relación de costo entre las mejoras que haría cambiar la decisión tomada en el punto anterior? Si puede, entonces haga el cálculo. Si no puede entonces explique porqué. (0,6 ptos).

R: Depende cómo se enfrente el problema por el alumno. Pero en general, se debe plantear una relación Coto/Beneficio. En principio, si la mejora 2, que es un 3,6% superior a la mejora 1 tiene un costo mayor, la decisión se podría cambiar. Si, para simplificar, se otorga el mismo peso al Costo que al Speedup logrado, se puede plantear una regla de 3. Si hubiera otras consideraciones, siempre sería posible plantear una ecuación para determinar la mejor decisión incluyendo los pesos definidos. Si el Costo de implementación y el Speedup tuvieran el mismo peso, entonces bastaría con que la mejora 2 tenga un Costo (C2) mayor que 1,031 veces el Costo (C1) de la mejora 1 para revertir la decisión.

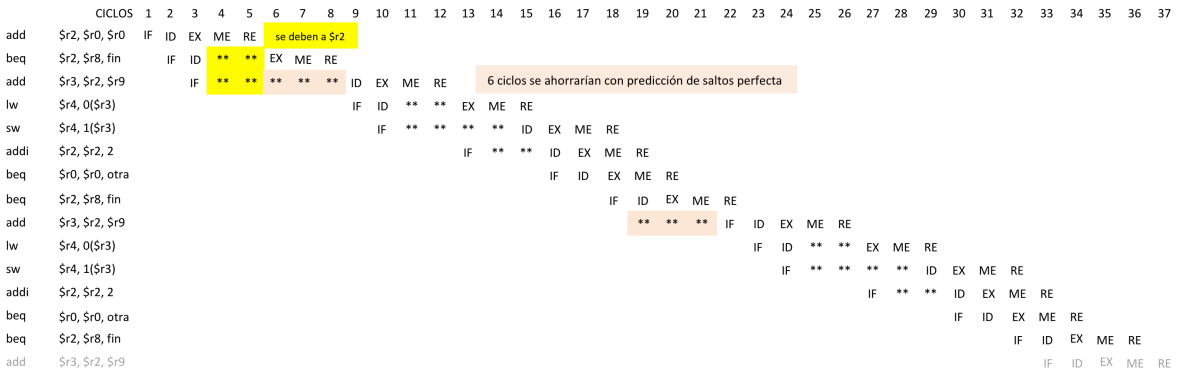
Pregunta 2.- Pipeline y hazards. (2,0 puntos en total)

Considere el mismo código MIPS. Pero con datos iniciales r0 = 0 ; r8 = 4.
Considere que el procesador funciona con pipeline de 5 etapas: IF, ID, EX, MEM y REG
Considere que es la primera versión del procesador y por lo tanto no tiene implementada ninguna mejoría para resolver los estancamientos del pipeline.
(En principio, considere que el branch es taken)

a. Determine todas las dependencias de datos. Indique aquellas que pueden producir estancamiento del pipeline. (0,5 ptos.)

```
        add   $r2, $r0, $r0
otra:   beq   $r2, $r8, fin     $r2 se escribe en la instrucción previa: RAW, estancaría el pipeline
        add   $r3, $r2, $r9     $r2 se lee y fue escrito en add previo: RAW, estancaría el pipeline
        lw    $r4, 0($r3)       $r3 se lee y fue escrito en add previo: RAW, estancaría el pipeline
        sw    $r4, 1($r3)       $r3 se lee y fue escrito en add previo: RAW, estancaría el pipeline
                                $r4 se lee y fue escrito en lw previo: RAW, estancaría el pipeline
        addi  $r2, $r2, 2       Se escribe $r2 que será usado en otra: RAW, estancaría el pipeline.
        beq   $r0, $r0, otra
fin:
```

b. Desarrolle el comportamiento del pipeline del programa completo bajo las condiciones establecidas. (0,5 ptos.)

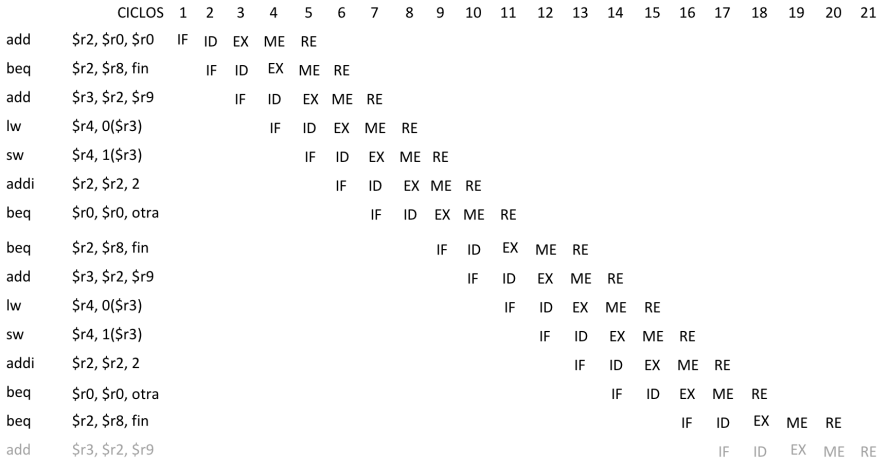


c. Asuma que se ha mejorado el procesador para que tenga una unidad de predicción de saltos perfecta, es decir, predice con 100% de certeza ¿Disminuye el tiempo de ejecución del programa? Si su respuesta es positiva, muestre cómo fue que lo determinó haciendo uso de información extraída desde el pipeline y diga cuanto fue en términos de tiempo (no de ciclos). Si su respuesta es negativa, explique porqué. (0,5 ptos.)

R: Del pipeline de la respuesta b) se deduce que hay 6 ciclos perdidos debidos a los saltos. Entonces, el tiempo perdido sería = 330x6 = 1.980ps = 1,98ns

d. Si la mejora propuesta en el punto previo no ha resuelto todos los estancamientos del pipeline. Agregue usted las mejoras necesarias para que se reduzcan al mínimo, o que se eliminen, todos los estancamientos. Con las mejoras propuestas, muestre en qué se modificó el comportamiento del pipeline. (0,5 ptos.)

R: Queda por eliminar los estancamientos debidos a las dependencia de datos de tipo RAW. Con mecanismos de Forwarding se pueden eliminar dichas esperas.



Pregunta 3.- Trayectoria de Datos (2,0 puntos en total)

Considere la figura de la página 3.

Los bloques lógicos usados en la implementación tienen las siguientes latencias:

I-Mem	Add	Mux	ALU	Regs	D-Mem	Sign-Extend	Shift-Left-2	ALU Ctrl
200ps	70ps	20ps	90ps	90ps	250ps	15ps	10ps	30ps

a. Determine las latencias de las distintas posibles trayectorias de este procesador asumiendo que la Unidad de Control tiene latencia cero o es despreciable. (0,5 ptos.)

R: Consideremos tres tipos de instrucciones representativas:

Instrucción Load: I-Mem + Mux + Regs + Mux + ALU + D-Mem + Mux
200 + 20 + 90 + 20 + 90 + 250 + 20 = 690ps

Instrucción jump: I-Mem + Add + ShLf2 + Mux
200 + + 10 + 20 = 230ps

Instrucción tipo R (add): I-Mem + Mux + Regs + Mux + ALU + Mux
200 + 20 + 90 + 20 + 90 + 20 = 440ps

b. Determine la Trayectoria Crítica y cuál es el tipo de instrucción para la cual ocurre, indicando las unidades funcionales (bloques) que utiliza (0,5 ptos)

R: La Trayectoria Crítica es el caso de Load, con 690ps. Las unidades funcionales están indicadas en la respuesta anterior.

c. Para evitar hacer aun más lenta la Trayectoria Crítica determinada en el punto anterior, se busca generar la señal MemWrite lo más rápida y tempranamente posible. ¿Cuánto tiempo debe tomar la Unidad de Control para generar MemWrite para cumplir esta premisa? Explique. (1,0 pto).

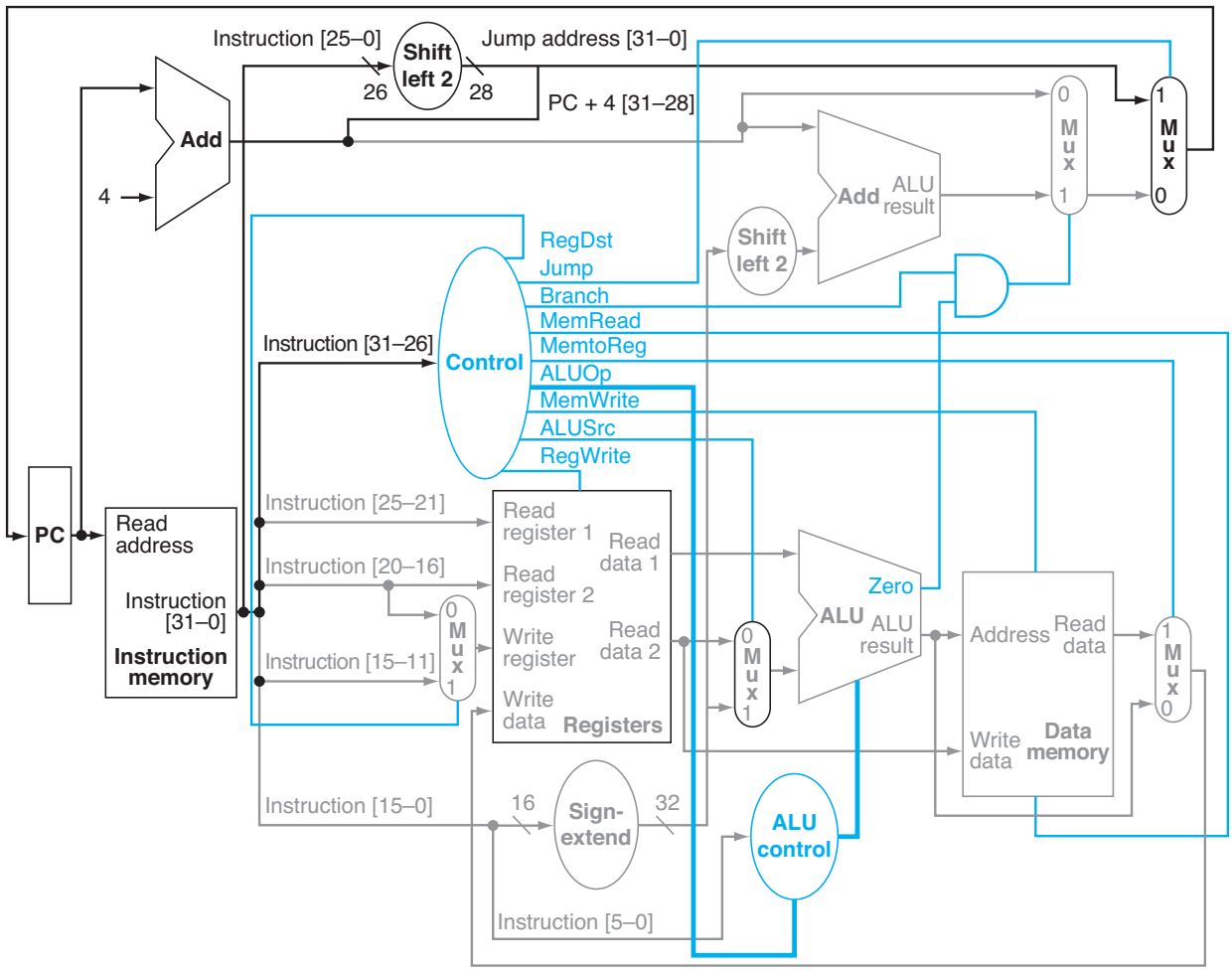
La Unidad de Control solo puede comenzar a generar MemWrite después de leer I-Mem. Debería terminar de generar esta señal antes del término de ciclo de reloj. MemWrite es una señal que habilita D-Mem (en rigor habilita los flip-flops de D-Mem). Si la escritura es habilitada por la transición del reloj, entonces la señal MemWrite debe llegar antes que ello ocurra, de lo contario debe esperar un ciclo de reloj. Entonces, MemWrite debe ser generado en un ciclo de reloj menos el tiempo de acceso a I-Mem. Considerando la Trayectoria Critica de 690ps como la que define el ciclo de reloj, entonces MemWrite debería generarse en 690-200=490ps.

Bonus Track: Ganará 1 punto adicional si escribe el código C que corresponde al código MIPS usado en la pregunta 1.

R: Código C

```
for (i=0; i!=j; i+=2)
        a[i+1] = a[i];
```

Nota: En el código MIPS se tendría: i en $r2; j en $r8; a en $r9


Instruction [25–0]
Shift left 2
Jump address [31–0]
26
28
PC + 4 [31–28]
Add
4
0
M
u
x
1
1
M
u
x
0
Add
ALU result
Shift left 2
RegDst
Jump
Branch
MemRead
MemtoReg
ALUOp
MemWrite
ALUSrc
RegWrite
Instruction [31–26]
Control
PC
Read address
Instruction [31–0]
Instruction memory
Instruction [25–21]
Instruction [20–16]
Instruction [15–11]
Instruction [15–0]
Instruction [5–0]
Read register 1
Read register 2
Write register
Write data
Read data 1
Read data 2
Registers
16
Sign-extend
32
ALU control
Zero
ALU
ALU result
Address
Read data
Write data
Data memory