ARQUITECTURA DE COMPUTADORES
GRADO EN INGENIERÍA INFORMÁTICA

PROBLEMAS DEL BLOQUE 1

ENUNCIADOS

TEMA 1-1: Fundamentos y Principios del Diseño de Computadores

Problema 1-1-1 (HP96, ex.1.1, pp.60; [KW96]pp.7). Suponer que se está considerando mejorar el comportamiento de una máquina añadiendo un modo vectorial. Cuando una operación se realiza en modo vectorial, se consigue una aceleración de factor 20 respecto al modo normal. Se denomina "porcentaje de vectorización" al tanto por ciento del tiempo de ejecución del programa que puede utilizar el modo vectorial.

(a) Dibuja una gráfica en la que se muestre la relación entre el speedup y el porcentaje de cómputo realizado en modo vectorial. Etiqueta el eje Y como "speedup" y el eje X como "porcentaje de vectorización".
(b) ¿Qué porcentaje de vectorización es necesario para conseguir un speedup de 2?
(c) ¿Qué porcentaje de vectorización se necesita para conseguir la mitad del speedup máximo posible?
(d) Suponer que hasta ahora el porcentaje de vectorización medido para los programas es del 70%. El grupo de hardware ha indicado que pueden doblar la velocidad de ejecución de la parte vectorial. En este momento queremos saber si el grupo de compiladores podría conseguir el mismo resultado aumentando exclusivamente el uso del modo vectorial. ¿Cuál es el porcentaje de vectorización necesario para conseguir con el software la misma ganancia en prestaciones que se consiguiría con la mejora del hardware?

Problema 1-1-2 (HP96, ex.1.4, pp.61; [KW96]pp.12). Suponer que se está considerando el cambio de un repertorio de instrucciones. La máquina original dispone sólo de instrucciones de carga y almacenamiento para acceder a memoria, y todas las operaciones trabajan sobre registros. La frecuencia promedio de aparición de las distintas instrucciones así como el CPI es el siguiente:

Tipo de Instrucción	Frecuencia	CPI
Operaciones ALU	43%	1
Cargas	21%	2
Almacenamientos	12%	2
Saltos	24%	2

Supongamos que el 25% de las operaciones con ALU utilizan un operando que se ha cargado desde memoria y que no vuelve a ser utilizado de nuevo. Se propone aumentar el repertorio de instrucciones con instrucciones que utilizan la ALU en la que uno de sus operandos es obtenido directamente de memoria. Esta nueva instrucción registro-memoria tarda 2 ciclos de reloj. Suponer que el repertorio de instrucciones extendido incrementa en 1 el número de ciclos de las instrucciones de salto, pero no afecta al tiempo de ciclo. ¿Mejora este cambio el rendimiento del procesador?.

Problema 1-1-3 (HP96, ex.1.5, pp.61; [KW96]pp.14). Suponer que se tiene una máquina con una memoria cache perfecta que se comporta como en la tabla del problema anterior.

Con una cache real, se ha medido que las instrucciones tienen una frecuencia de fallo del 5%, una frecuencia de fallo de datos del 10%, y una penalización por fallo de cache de 40 ciclos.

Encontrar el CPI para cada instrucción con estas tasas de fallos de caché, y determinar cuánto de rápida es la máquina sin fallos de cache con respecto a la que tiene fallos.

Problema 1-1-4 (HP03, Fig.1.17). Suponer que se dispone de 3 máquinas (A, B, y C) en las que se ejecutan 2 programas: P1 y P2. Se miden los tiempos de ejecución que se pueden observar en la siguiente figura, los cuales están normalizados al valor obtenido en una máquina determinada.

	Normalizado a A			Normalizado a B			Normalizado a C		
P1	1.0	10.0	20.0	0.1	1.0	2.0	0.05	0.5	1.0
P2	1.0	0.1	0.02	10.0	1.0	0.2	50.0	5.0	1.0
Tiempo total	1.0	0.11	0.04	9.1	1.0	0.36	25.03	2.75	1.0
Media aritmética									
Media geométrica									

Calcula la media aritmética y geométrica para cada máquina suponiendo los tres tipos de normalización mostrados. A partir de estos resultados, razona ¿cuál es la mejor forma de describir las prestaciones de un computador?

PROBLEMA 1-1-5. Este problema aborda el análisis de prestaciones de un procesador.

a) Nombrar los 3 componentes principales de la ecuación del tiempo de ejecución. ¿Cómo se combinan para cuantificar dicho tiempo de ejecución?
b) ¿Cuál es el CPI y la medida MIPS de la máquina? Suponer una frecuencia de reloj de 300 MHz y las siguientes medidas estadísticas:

Tipo de instrucción	Frecuencia (%)	Ciclos
ALU	40	1
Carga/Almacenamiento	30	2
Saltos	20	3
Coma flotante	10	5

TEMA 1-2: Aritmética Avanzada

Problema 1-2-1 (HP96, ex.A.1, pp.A-72). Genera una expresión para los números mayores y menores que pueden ser representados por *n* bits en la representación en "complemento á 2".

Problema 1-2-2 (HP96, ex.A.3, pp.A-72). Utilizando números de 4 bits, multiplica –8 x –8 utilizando el algoritmo de Booth.

Problema 1-2-3 (HP96, ex.A.5, pp.A-72). En una máquina que no puede detectar overflow en hardware, muestra cómo se podría detectar overflow en una operación de suma en la que los números tuvieran signo.

Problema 1-2-4 (HP96, ex.A.6, pp.A-72). Representa los siguientes números en simple y doble precisión utilizando la representación IEEE754: (a) 10, (b) 10.5, (c) 0.1.

Problema 1-2-5 (HP96, ex.A.7, pp.A-72). En la siguiente lista de números en punto flotante y simple precisión, genera para cada uno de ellos la representación en binario, decimal e IEEE754.

Problema 1-2-6 (HP96, ex.A.10, pp.A-73). Utilizando una representación de números en punto flotante de 4 bits de precisión, muestra cómo el algoritmo de multiplicación binaria en punto flotante realiza el producto de 1.875 x 1.875.

Problema 1-2-7 (HP96, ex.A.11, pp.A-73). Con relación a la suma de exponentes en la multiplicación en punto flotante:

(a) Cómo sería el hardware para la suma de exponentes?

(b) Si el desplazamiento en simple precisión fuera 129 en vez de 127, la suma sería más complicada de implementar?

Problema 1-2-8 (Examen Parcial 2005). Realiza la operación de multiplicación de los números "-8" y "-8" utilizando el algoritmo de Booth.

TEMA 1-3: Técnicas Avanzadas de Segmentación

Problema 1-3-1 (HP96, ex.3.1, pp.214). Utilizar el siguiente fragmento de código:

```
loop:     LW   R1,0(R2)
          ADDI R1,R1,#1
          SW   R1,0(R2)
          ADDI R2,R2,#4
          SUB  R4,R3,R2
          BNZ  R4, loop
```

Suponer que el valor inicial de R3 es R2+396. Suponer también que se utiliza la implementación segmentada del procesador DLX de 5 etapas para enteros, y que todos los accesos a memoria cache son aciertos. Además, suponer que los saltos se resuelven y actualizan el PC al final de la etapa MEM, y que no existen saltos retardados.

a. Mostrar la temporización de esta secuencia de instrucciones para la segmentación del DLX suponiendo que no se utiliza anticipación y que en un mismo ciclo de reloj se puede realizar una lectura y escritura en el banco de registros. Suponer también que los saltos condicionales se predice que no se va a realizar el salto, y se manejan eliminando las instrucciones que se han introducido posteriormente al salto. Si todas las referencias a memoria aciertan en la cache, ¿cuántos ciclos tarda el bucle en ejecutarse?.
b. Mostrar la temporización de esta secuencia de instrucciones para el DLX segmentado suponiendo la aplicación de la técnica de anticipación. Suponer que en los saltos condicionales se predice que no se va a realizar el salto. Si todas las referencias a memoria aciertan en la cache, ¿cuántos ciclos tarda el bucle en ejecutarse?.
c. Suponiendo que se tiene un procesador DLX segmentado cuyos saltos condicionales se resuelven ahora en la etapa ID y están retardados en 1 ciclo, y que además se ha implementado la técnica de anticipación, planifica las instrucciones del bucle incluyendo los saltos retardados. Para ello se pueden reordenar las instrucciones y modificar los operandos de las instrucciones, pero no se deben modificar los códigos de operación ni el número de instrucciones. Mostrar el diagrama temporal de los segmentos y el número de ciclos necesarios para ejecutar todo el bucle.

Problema 1-3-2 (HP96, ex.3.3, pp.216). Exploraremos aquí las técnicas de segmentación para una arquitectura memoria-registro. La arquitectura tiene dos formatos de instrucción: un formato registro-registro y un formato registro-memoria. Existe un modo de direccionamiento a memoria para las operaciones registro-memoria que es del tipo: inmediato + registro base.

Se dispone de un conjunto de operaciones ALU con los siguientes formatos:

Formato 1: `ALUop Rdest, Rsrc1, Rsrc2`
Formato 2: `ALUop Rdest, Rsrc1, MEM`

Donde `ALUop` representa a uno de los siguientes operaciones: Add, Subtract, And, Or, Load (ignorando Rsrc1 en el formato 2), Store. `Rsrc` o `Rdest` son registros. MEM representa al par (registro base, inmediato).

Los saltos condicionales utilizan una comparación entre el contenido de dos registros en la etapa ALU2, y el salto se realiza relativo al Contador de Programa (PC). Suponer que la máquina es segmentada y monoescalar. Por lo tanto, una nueva instrucción comienza a ejecutarse cada ciclo de reloj. Su microarquitectura segmentada evoluciona ciclo a ciclo de la siguiente manera (parecida al VAX 8700):

CLK1	CLK2	CLK3	CLK4	CLK5	CLK6	CLK7	CLK8	CLK9	CLK10	CLK11
IF	RF	ALU1	MEM	ALU2	WB					
	IF	RF	ALU1	MEM	ALU2	WB				
		IF	RF	ALU1	MEM	ALU2	WB			
			IF	RF	ALU1	MEM	ALU2	WB		
				IF	RF	ALU1	MEM	ALU2	WB	
					IF	RF	ALU1	MEM	ALU2	WB

La primera etapa de la ALU (ALU1) se utiliza para el cálculo de la dirección de las referencias a memoria y de las direcciones de salto. El segundo ciclo ALU (ALU2) se utiliza para realizar las operaciones ALU, así como las comparaciones asociadas a los saltos. RF es la etapa de decodificación y de búsqueda de los operandos. Suponer que se pueden realizar lecturas y escrituras al mismo registro en el mismo ciclo de reloj.

a. Encontrar el número necesario de sumadores y mostrar una combinación de instrucciones y etapas de segmentación que justifique la respuesta. Sólo se requiere mostrar una sola combinación que requiera todos los sumadores.
b. Encontrar el número de puertos de lectura y de escritura que se necesitan del banco de registros así como el número de puertos de lectura y escritura que se requieren de la memoria.
c. Determinar las anticipaciones de datos que se necesitan entre cualquiera de las ALUs. Suponer que existen ALUs separadas para las etapas ALU1 y ALU2. Especificar las conexiones necesarias entre las ALUs para implementar las anticipaciones y así reducir las paradas del flujo de segmentación.
d. Mostrar todos los requerimientos de anticipación de datos que son necesarios cuando las unidades fuentes o destinos no son ALUs.
e. Mostrar el resto de dependencias de datos que no puedan resolverse por anticipación y que impliquen al menos que una de las unidades fuente o destino no sea una ALU. Determinar el número de ciclos de paradas necesarios.
f. Mostrar todos los tipos de dependencias de control a través de ejemplos y especifica la longitud de la parada.

Problema 1-3-3 (HP96, example, pp.137). Suponer que una máquina multiciclo no segmentada tiene un ciclo de reloj de 10 ns. y que utiliza 4 ciclos para operaciones ALU y saltos condicionales y 5 ciclos para operaciones de memoria. Suponer que las frecuencias relativas de estas operaciones son 40%, 20% y 40% respectivamente. Suponer que debido al "clock skew and setup", la máquina segmentada aumenta 1 ns de overhead al ciclo de reloj. ¿Cuánto speed-up en el tiempo de ejecución de las instrucciones se obtendrá con la segmentación respecto a la implementación multiciclo?.

Problema 1-3-4 (HP96, ex.3.4, pp.217). Suponer que se tiene una situación como la del problema anterior en la que el overhead en cada uno de los segmentos del procesador es de 1 ns. Suponer también que cada una de las 5 etapas en que está dividido el procesador tarda 10 ns. sin incluir el overhead. Dibujar el speed-up de la máquina segmentada en relación a la no segmentada a medida que el número de segmentos se incrementa hasta un total de 20, considerando sólo el impacto del overhead y suponiendo que el trabajo puede ser homogéneamente dividido a medida que el número de etapas aumenta (lo cual no es completamente cierto). Dibujar además el speed-up perfecto que se obtendría si no existiera overhead.

Problema 1-3-5 (HP96, ex.3.5, pp.217). Una máquina se denomina "subsegmentada" si se pueden combinar varias etapas de segmentación sin cambiar apreciablemente el comportamiento respecto a las paradas del flujo de ejecución de las instrucciones. Suponer que en la máquina DLX segmentada de 5 etapas se han unificado las etapas EX y MEM para lo que el ciclo de reloj se ha alargado en un 50%. ¿Cuánto de más rápida sería la máquina DLX convencional respecto a la máquina subsegmentada para código sobre enteros?. Utilizar los datos para el programa gcc (el 4% de las instrucciones producen una parada del flujo debido a saltos condicionales, y un 5% producen una parada del flujo de instrucciones debido a LW). Suponer que los saltos condicionales se resuelven en la etapa ID.

Problema 1-3-6 (HP96, ex.3.9, pp.217). Suponer que la frecuencia de saltos (en % de todas las instrucciones) son las siguientes:

Saltos condicionales: 20% (en el 60% se produce el salto)
Saltos incondicionales y llamadas a rutinas: 5%

Estamos examinando aquí una segmentación de 4 etapas donde los saltos incondicionales se resuelven al final del segundo ciclo y los saltos condicionales se resuelven al final del tercer ciclo. Suponiendo que sólo la primera etapa de segmentación (IF) se puede realizar independientemente de si el salto se realiza o no e ignorando las otras etapas de la segmentación, ¿cuánto de más rápida sería la máquina sin dependencia de control por saltos?.

PROBLEMA 1-3-7 (HP96, ex.3.10, pp.218; Examen Final 27-1-2006). Suponer que un procesador dispone de las siguientes etapas de segmentación:

Etapa	Función
1 (IF)	Búsqueda de instrucción
2 (ID)	Decodificación y Búsqueda de Operandos
3 (EX)	Ejecución, Acceso memoria, Resolución de saltos, Escritura de resultados

Todas las dependencias de datos se restringen a la interrelación entre el registro escrito en la etapa 3 (EX) de la instrucción *i* y la lectura de registro de la instrucción *i+1* en la etapa 2 (ID) antes de que la instrucción *i* se haya completado. La probabilidad de que aparezca esta dependencia es de *1/p* , es decir p^{-1}. La probabilidad representa a la frecuencia de aparición de la dependencia.

Se está considerando un cambio en la organización del procesador que resolvería los saltos y escribiría el resultado de una instrucción en una cuarta etapa de la segmentación (WB). Esto reduciría el tamaño del periodo de reloj en una cantidad *d*. Es decir, si el periodo original es *T*, el periodo que resultaría a consecuencia del cambio sería *T-d*. Esta modificación origina la aparición de un nuevo riesgo de penalización por dependencia de datos entre las instrucciones *i* e *i+2*. La probabilidad que aparezca una dependencia de datos entre la instrucción *i* y la instrucción *i+2* es p^{-2}. Suponer que:

- El valor de p^{-1} incluye los casos de las dependencias tanto entre *i* e *i+1* como *i* e *i+2*
- Las instrucciones de salto también se resolverían en esta cuarta etapa (WB)

- El banco de registros no permite la escritura y lectura de un mismo registro en el mismo ciclo de reloj

a. Suponer que no se añade hardware adicional para aplicar anticipación a la cuarta etapa. Considerar exclusivamente las dependencias de datos, y encontrar el límite inferior de *d* que ocasiona que el cambio a 4 etapas proporcione un menor tiempo de ejecución.
b. Suponer ahora que hemos utilizado anticipación para eliminar las penalizaciones extras que aparecen a raíz del cambio a 4 etapas de segmentación y que son originadas por las dependencias de datos "i → i+2". Es decir, que para todas las dependencias de datos, la longitud efectiva de la ruta segmentada es de 3 etapas. Este nuevo diseño no tiene por qué ser mejor debido al impacto de las dependencias de control que se originan en una ruta de 4 segmentos frente a una de 3 segmentos. Suponer que en ambos procesadores, la siguiente instrucción a un salto puede entrar y permanecer "sólo" en la etapa 1 (IF) antes que se decida si el salto se produce o no. Queremos analizar el impacto de las dependencias de control por saltos antes de que segmentemos más la ruta de datos para saber si se va a conseguir mayor nivel de prestaciones. Encontrar un límite superior para el porcentaje de saltos en el programa en función de la relación entre *d* y el ciclo de reloj original (T), de tal forma que la modificación de la ruta de datos proporciona mejor nivel de prestaciones. En el caso de que *d* represente un 10% de reducción del periodo de reloj, ¿cuál es el valor máximo del porcentaje de saltos condicionales antes de que un mayor número de segmentos produzca una disminución del nivel de prestaciones? Suponer que el porcentaje de saltos condicionales tomados es del 60%.

PROBLEMA 1-3-8 (fa98-preq_soln.pdf, Question 2). La siguiente secuencia de código corresponde al ensamblador DLX. Suponer que se va a ejecutar en un procesador con 5 etapas de segmentación y saltos retardados de 1 ciclo.

```
0:          add  r6,r0,r0
4:          lw   r1,16(r19)
8:          lw   r2,32(r19)
12:   loop:slli r4,r2,#3
16:         addu r5,r4,r1
20:         lw   r4,0(r5)
24:         add  r6,r6,r4
28:         bnez r2,loop
32:         addi r2,r2,-1
36:         j    exit
40:         add  r10,r0,r0
44:   exit:addi r11,r10,#12
```

Identifica todas las dependencias de datos que aparecen en este programa, y clasifícalas en uno de los siguientes tipos: RAW, WAR, WAW. Observar que se está pidiendo que identifiques todas las dependencias del programa y no sólo las dependencias de la segmentación.

Para cada dependencia, determinar si es necesario aplicar la técnica de anticipación (la del procesador DLX segmentado en 5 etapas), o si se requiere parar el flujo de ejecución de la segmentación aunque se utilice anticipación. Suponer que un registro puede ser actualizado y leído en el mismo ciclo de reloj sin necesidad de anticipación, y por lo tanto no produce parada del flujo de instrucciones.

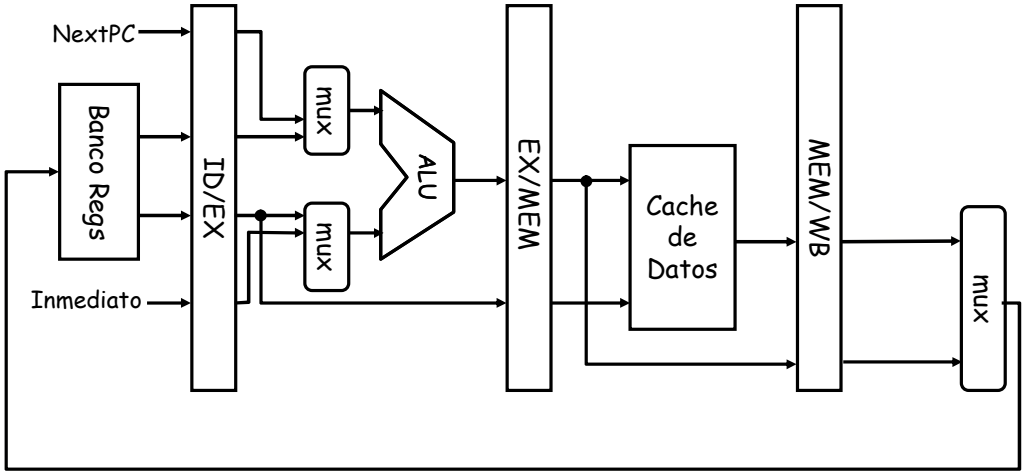
Para resolver este problema, rellena la tabla que aparece a continuación. "Instrucción Fuente" se refiere a la que escribe en el registro, "Instrucción Destino" a la que lee el registro. Para cada una de ellas,

escribir el número de posición de memoria que aparece a la izquierda. "Registro" representa al registro involucrado en la dependencia. Se muestran dos ejemplos:

Instr. Fuente	Inst. Destino	Tipo Depend	Registro	Anticipación	Parada
32	32	WAR	R2	no	No
40	44	RAW	R10	si	No

PROBLEMA 1-3-9 (preq-soln.pdf, 2). Este problema involucra a las dependencias de una ruta de datos segmentada.

a) Existen 3 tipos diferentes de dependencias de datos: RAW, WAR, y WAW. Definirlas, proporcionando una corta secuencia de código que ilustre cada una de ellas. Indicar brevemente para cada tipo de dependencia, cuál es la técnica que permite eliminarla.



b) ¿Qué son las dependencias de control?. Mencionar 2 de las técnicas que se utilizan para resolverlas.

c) Dibuja y describe modificaciones simples a la ruta de datos de la figura anterior que se necesitan para eliminar las dependencias de datos. Incluir las señales de control necesarias.

d) Las modificaciones anteriores de la ruta de datos dispondrán de un bloque que las controle. Dibuja en un diagrama las señales que requeriría tal módulo de control así como sin son de entrada o salida.
e) Explica cómo la unidad de resolución de dependencias reconoce y resuelve las dependencias de datos para cada una de las siguientes secuencias. Se deben utilizar las señales de entrada y salida del apartado
f) La expresión que debe aparecer en la parte de reconocimiento de la dependencia debe hacerse con expresiones booleanas. La expresión que debe aparecen en la parte de resolución debe proporcionar valores para las señales de salida del apartado (d).

`lw r1,0(r2)`
`add r3,r1,r2`
`add r5,r4,r6`
	`add r1,r2,r3`
	`add r5,r6,r7`
	`sub r4,r1,r2`
		`add r1,r2,r3`
		`sub r4,r4,r1`
		`add r5,r5,r3`

PROBLEMA 1-3-10 (HP96, ex.3.11, pp.218). Construir una tabla con las condiciones de una ruta de datos FP segmentada para un procesador DLX32mf con múltiples unidades funcionales que originen

las paradas del flujo de instrucciones por dependencias RAW. No considerar las divisiones, sólo las instrucciones Enteros/FP y FP/FP.

PROBLEMA 1-3-11 (HP96, ex.3.15, pp.219). Construir una tabla con las condiciones de la ruta de datos FP del procesador R4000 que produce paradas por dependencias estructurales.

PROBLEMA 1-3-12 (examen)

a) Dar una definición sencilla de "Excepciones/Interrupciones Precisas".
b) Explicar cómo la presencia de saltos retardados complica la descripción del punto donde se produce una excepción precisa (pensar en la información que el sistema operativo requiere para continuar la ejecución después de la excepción).
c) El procesador Alpha 21064 soportaba excepciones precisas para la memoria virtual pero no para operaciones en punto flotante. ¿Qué tipo de argumento fundamentaría la justificación de esta complicada combinación de comportamientos?.
d) En manejo de eventos externos de Entrada/Salida, ¿cuándo es mejor utilizar interrupciones que encuestas?, ¿y al revés?.
e) Menciona 3 eventos que se intente predecir en computadores modernos.
f) ¿Por qué la predicción de saltos es deseable?.

PROBLEMA 1-3-13 (examen). Suponer que se dispone de un procesador cuyo propósito es que el flujo de instrucciones nunca se pare a no ser que los operandos no estén disponibles. Sus características son las siguientes:

- Monoescalar (sólo se envía a la etapa EX una instrucción a la vez)
- Segmentación en orden con las siguientes etapas: 1 búsqueda (IF), 1 decodificación (ID), múltiples etapas de ejecución (EX1, EX2, ...), 1 postescritura (WB).

Suponer las siguientes latencias de la única etapa EX que se encuentra segmentada de forma tal que cada tipo de instrucción requiere los siguientes números de ciclos de reloj:

- multf: 5 ciclos
- addf: 3 ciclos
- divf: 2 ciclos
- operaciones sobre enteros y saltos: 1 ciclo
- cargas/almacenamientos: 3 ciclos (incluido el cálculo de la dirección)

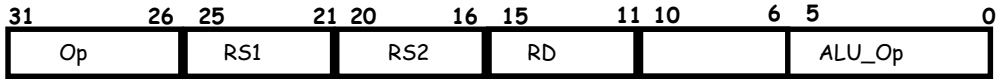
Suponer que este procesador segmentado consiste en una secuencia lineal de etapas en las que las últimas etapas de EX equivalen a no-operaciones para las instrucciones que requieren un número inferior de ciclos en la etapa EX.

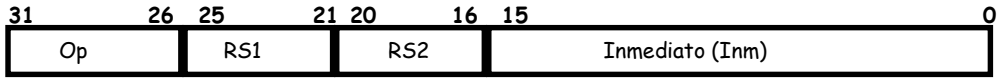
a) Dibujar los distintos segmentos y asignarles siglas distintas. Describir la función que se le asigna a cada segmento y mostrar a través de flechas los caminos de anticipación entre etapas. Etiqueta cada flecha con el tipo de instrucción que anticipará sus resultados a través de estas trayectorias. Por ejemplo, utilizar “Ld” para las cargas/almacenamientos, "M" para multf, "D" para divf, "A" para addf, e "I" para operaciones con enteros.
b) ¿Cuántas instrucciones extra se requieren entre cada una de las siguientes combinaciones de instrucciones para evitar paradas (por ejemplo, suponer que una segunda instrucción utiliza un valor que proporciona la primera): divf/almacenamiento, carga/multf, 2 operaciones consecutivas sobre números enteros, multf/addf, addf/divf, enteros/almacenamiento?
c) ¿Cuántas instrucciones se podrán ejecutar dentro de un salto retardado si la condición se establece comparando dos registros? Razonar la respuesta.
d) ¿Existen dependencias WAR y WAW?

Pregunta 1-3-14 (examen). Describe las modificaciones de la ruta de datos segmentada del procesador DLX32p con unidad funcional de 1 ciclo para resolver la dependencia de datos siguiente utilizando anticipación:

```
ADD R1,R3,R4 ;
SW  R1,0(R2) ;
```

Formatos para las instrucciones REGISTRO-REGISTRO (R-R) y REGISTRO-INMEDIATO (R-I)





R-I

PROBLEMA 1-3-15 (examen). Suponer que se está analizando la introducción de un nuevo modo de direccionamiento a la arquitectura de repertorio de instrucciones que ejecuta el procesador DLX32p, el cual permite que las instrucciones aritmético-lógicas puedan disponer de un operando fuente en memoria. Para aliviar la complejidad, suponemos que para todo tipo de operaciones el direccionamiento es indirecto, incluidas las instrucciones de carga y almacenamiento (LW, SW). Es decir, la dirección de memoria se encuentra contenida en todos los casos en un registro del banco de registros y no se requiere sumarle un inmediato de 16 bits extendido en signo.

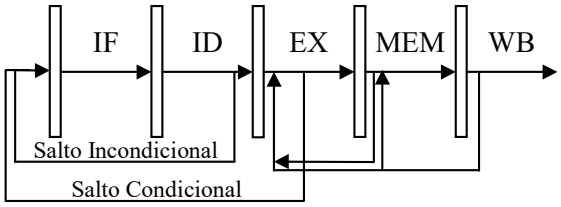
(a) Proponer el/los cambios necesarios a la segmentación del procesador DLX32p original para implementar el modo de direccionamiento indirecto. Describirlos a través del lenguaje de transferencias entre registros. Las operaciones asociadas a la ejecución de los distintos tipos de instrucciones se muestran en la siguiente tabla (expresadas en lenguaje de transferencia entre registros).

Etapa	**Instrucciones ALU**	**Instrucciones LW/SW**	**Instrucciones SALTOS Condicional**
IF	IR←Mem[PC] PC←PC+4	IR←Mem[PC] PC←PC+4	IR←Mem[PC] if cond PC←ALUout else PC←PC+4 PC'←PC
ID	A←RS1 B←RS2 Inm←$((IR_{15})^{16}$ | | $IR_{15..0})$	A←RS1 (dirección) B←RS2 (dato para sw) Inm←$((IR_{15})^{16}$ | | $IR_{15..0})$	A←RS1 Inm←$((IR_{15})^{16}$ | | $IR_{15..0})$ PC''←PC'
EX	ALUout←A op (B || Inm)	ALUout ← A + Inm B'←B	ALUout←PC'' + Inm cond←A op 0
MEM	ALUout '← ALUout	MDR←Mem[ALUout] (LW) Mem[ALUout] ←B' (SW)	
WB	RD←ALUout'	RD←MDR (LW)	

IR: equivalente al registro de instrucción
A, B: registros que almacenan la salida de datos del banco de registros
ALUout: equivale al registro que almacena el resultado de la operación de la ALU
MDR: equivale al registro de datos de salida de la memoria de datos (forma parte de un registro de segmentación)

(b) Cuáles son las nuevas dependencias de datos que aparecen al insertar este nuevo modo de direccionamiento y que introducen ciclos de penalización. Proporcionar un ejemplo de cada una de ellas.
(c) Enumerar todos los cambios o añadidos que deben ser realizados en el hardware para implementar este modo de direccionamiento.
(d) Enumerar todos los aspectos del funcionamiento del nuevo procesador que se deben tomar en consideración para estimar el impacto sobre sus prestaciones

Problema 1-3-16 (Examen Parcial 30/1/04). Procesador con Organización Segmentada Simple



Instrucción	Resol.	Semántica
BR label	ID	Salto incondicional a "label"
B*cc* Rx,label	EX	Salto condicional, si *cc*(Rx)
OP Rx,Ry,Rz	EX	(Rx OP Ry) → Rz
LDQ Rx,off(Ry)	MEM	Carga desde Memoria a Rx
STQ (Ry),Rx	MEM	Almacena Rx en Memoria

La figura superior muestra de forma simplificada la organización segmentada de 5 etapas de un procesador. Sólo se muestra la organización en etapas y los principales caminos de control y de datos entre las distintas etapas. Las instrucciones siempre comienzan a ejecutarse en orden y modifican el estado del procesador (registros y memoria) también en orden. Suponemos que no se producen fallos ni en la cache de datos ni en la de instrucciones. No existe predicción de saltos (es decir, se continúan obteniendo instrucciones de la cache de instrucciones suponiendo que los saltos no se toman).

El repertorio de instrucciones y la etapa en que cada tipo de instrucción se resuelve se muestran en la tabla que aparece a la derecha de la figura. Los saltos incondicionales se resuelven al final de la segunda etapa y los saltos condicionales al final de la tercera. Las operaciones aritméticas se resuelven al final de la tercera etapa. Las operaciones de acceso a memoria acaban al final de la cuarta etapa. En la quinta etapa se escribe el resultado de la operación en el registro destino (si es necesario). Las instrucciones no tienen que esperar a que sus operandos estén almacenados correctamente en el banco de registros, sino que pueden obtener el valor por caminos de anticipación justo al comenzar el ciclo siguiente a que se resuelva la instrucción anterior.

Sobre el procesador anterior se ejecuta el programa que se muestra a continuación.

```
1  LDQ     R5,0(R7)       ; R5 ← MEM(R7)
2  LDQ     R6,8(R7)       ; R6 ← MEM(R7+8)
3  LDQ     R1,-8(R6)      ; R1 ← MEM(R6-8)
4  BR      11             ; salta a instrucción 11
5  LDQ     R2,0(R6)       ; R2 ← MEM(R6)
6  SUBQ    R2,R1,R3       ; R3 ← R2-R1
7  BGE     R3, 9          ; if R3>=0 salta a 9
8  MOVQ    R1,R2          ; R1 ← R2
9  ADDQI   R6,8,R6        ; R6 ← R6+8
10 SUBQI   R5,1,R5        ; R5 ← R5-1
11 BNE     R5, 5          ; if R5!=0 salta a 5
12 LDA     R7,100(R7)     ; R7 ← 100+R7 (NO ACCEDE A MEMORIA)
13 STQ     R1,(R7)        ; MEM(R7) ← R1
```

(b) Para este fragmento de programa y para este procesador, ¿cuál es el promedio de instrucciones ejecutadas por ciclo (IPC = instrucciones por ciclo = 1/CPI)? Identificar cada uno de los casos que provocan que el IPC sea menor que uno. Para cada caso diferente, indicar el número de ciclos de penalización que provoca y dar una explicación de por qué se produce la penalización.

ciclo	*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*	*32*	*33*
IF	1	2	3	..																													
ID		1	2	..																													
EXE			1	..																													
MEM				1																													
WB					1																												

ciclos	*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*
1.R5←M(R7)	**IF**	**ID**	**EX**	**M**	**WB**																						
2.R6←M(R7+8)		**IF**	**ID**																								
3.R1←M(R6-8)			**IF**																								
4.goto 11																											
11.R5≠0⇒ 5																											
5.R2←M(R6)																											
6.R3←R2-R1																											
7.R3≥0⇒ 9																											
9.R6←R6+8																											
10.R5←R5-1																											
11.R5≠0⇒ 5																											
5.R2←M(R6)																											
6.R3←R2-R1																											
7.R3≥0⇒ 9																											
8.R1←R2																											
9.R6←R6+8																											
10.R5←R5-1																											
11.R5≠0⇒ 5																											
12.R7←R7+100																											
13.M(R7)←R1																											

(b) Para este fragmento de programa y para este procesador, ¿cuál es el promedio de instrucciones ejecutadas por ciclo (IPC = instrucciones por ciclo)? Identificar cada uno de los casos que provocan que el IPC sea menor que uno. Para cada caso diferente, indicar el número de ciclos de penalización que provoca y dar una explicación de por qué se produce la penalización.

(c) Suponer que no existieran caminos de anticipación de datos y que los resultados de una instrucción no pudieran ser utilizados por una instrucción posterior hasta haberse escrito en el banco de registros. Supondremos que mientras una instrucción se encuentra en la etapa WB, escribiendo el resultado en el banco de registros, otra instrucción en la etapa ID puede leer este resultado del banco de registros. ¿Cuántos ciclos de penalización se añadirían en el programa del apartado a? Suponemos que las instrucciones de salto siguen tratándose como en el apartado a. ¿Qué porcentaje de pérdida se produciría?

ARQUITECTURA DE COMPUTADORES
GRADO EN INGENIERÍA INFORMÁTICA

PROBLEMAS DEL BLOQUE 1

SOLUCIONES

SOLUCIONES TEMA 1-1

Problema 1-1-1 (HP96, ex.1.1, pp.60; [KW96]pp.7). Suponer que se está considerando mejorar el comportamiento de un procesador añadiendo un modo vectorial. Cuando una operación se realiza en modo vectorial, se consigue una aceleración de factor 20 respecto al modo normal. Se denomina *"porcentaje de vectorización"* al tanto por ciento del tiempo de ejecución del programa que puede utilizar el modo vectorial.

(a) Dibuja una gráfica en la que se muestre la relación entre el Speed-Up y el porcentaje de cómputo realizado en modo vectorial. Etiqueta el eje Y como "Speed-Up" y el eje X como "porcentaje de vectorización".

Solución:

Para poder realizar la gráfica, primero debemos calcular una serie de valores para el speedup dependiendo del porcentaje de cómputo realizado en modo vectorial. La ley de Amdahl (ley de rendimientos decrecientes) dice que:

$$\text{Speed-Up} = 1 / (F/S + (1 - F))$$

Donde, F corresponde al tanto por ciento de ejecución del programa que utiliza el modo vectorial, y S al factor de mejora.

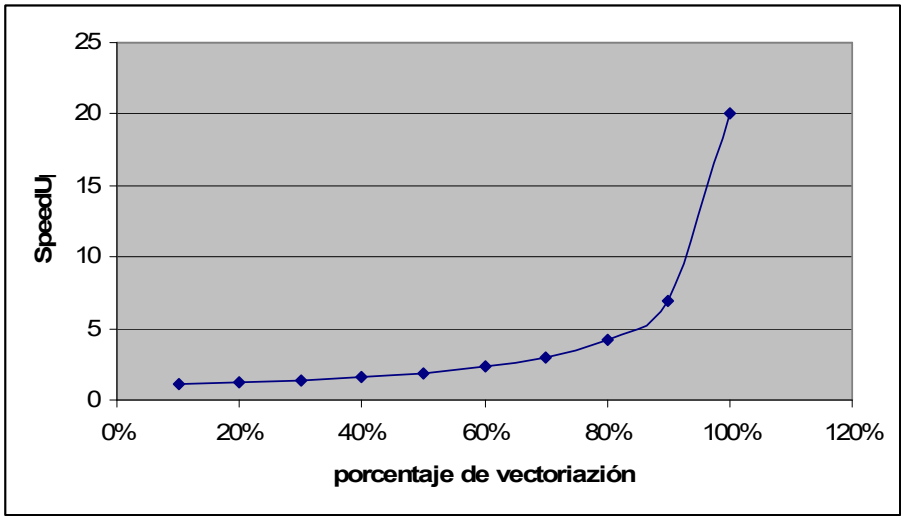
A continuación, vamos a ir calculando el Speed-Up dependiendo del "porcentaje de vectorización" que se irá incrementado en un 10% hasta llegar a 100%:

$$\text{Speed-Up}_{10\%} = 1 / (0.1/20 + 0.9) = 1.10$$

El resto se calcula de forma similar y vemos los resultados en la siguiente tabla:

% de vectoriación (F)	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
SpeedUp	1.10	1.23	1.40	1.61	1.9	2.33	3	4.2	6.9	20

La representación gráfica de esta tabla es:



Podemos ver que se pueden diferenciar en esta curva tres pendientes:

1. cuando el porcentaje de vectorización se encuentra entre 10% y 50%, la tasa de incremento de Speed-Up es pequeña;
2. cuando el porcentaje de vectorización se encuentra entre 50% y 80%, el Speed-Up crece a mayor ritmo, llegándose a alcanzar un factor de 5;
3. y en el último tramo de vectorización entre 85% y 100%, el Speed-Up crece exponencialmente hasta llegar a su valor máximo posible en 100% = 20.

(b) ¿Qué porcentaje de vectorización es necesario para conseguir un speedup de 2?

Solución:
Aplicando la fórmula de la ley de Amdahl tenemos: $2 = 1 / (F/20 + (1 – F))$; y despejando F obtenemos: $F = 20/38 = 0.526$, lo cual equivale a un 52.6% de vectorización.

(c) ¿Qué porcentaje de vectorización se necesita para conseguir la mitad del Speed-Up máximo posible?

Solución:
El Speed-Up máximo posible es 20, su mitad es 10. Nuevamente, aplicando la fórmula de la ley de Amdahl tenemos: $10 = 1 / (F/20 + (1 – F))$; y despejando F obtenemos: $F = 180/190 = 0.947$, lo cual equivales a un 94.7% de vectorización.

(d) Suponer que hasta ahora el porcentaje de vectorización medido para los programas es del 70%. El grupo de hardware ha indicado que puede doblar el ritmo de ejecución de instrucciones de la parte vectorial. En este momento queremos saber si el grupo de compiladores podría conseguir una mejora de prestaciones equivalente aumentando exclusivamente el uso del modo vectorial. ¿Cuál es el porcentaje adicional de vectorización necesario para conseguir con el software la misma ganancia en prestaciones que se conseguiría con la mejora del hardware?

Solución:
Tenemos que F = 70%, y un factor 40 de aceleración respecto al modo normal (no vectorial). Calculamos el correspondiente Speed-Up en estas condiciones:

$$\text{Speed-Up}_{70\%\text{doble}} = 1 / (0.7/40 + 0.3) = 3.15X$$

Para conseguir con el software la misma ganancia en prestaciones, el porcentaje de vectorización debe ser: $3.15 = 1 / (F/20 + (1 – F))$; y despejando F obtenemos: $F = 0.716$, lo cual equivale a un 71.6% de vectorización, es decir, un 1.6% de vectorización adicional para conseguir por software el mismo efecto sobre el Speed-Up que conseguiría la optimización del hardware.

Problema 1-1-2 (HP96, ex.1.4, pp.61; [KW96]pp.12). Suponer que se está considerando el cambio de un repertorio de instrucciones. La máquina original dispone sólo de instrucciones de carga y almacenamiento para acceder a memoria, y todas las operaciones trabajan sobre registros. La frecuencia promedio de aparición de las distintas instrucciones, así como el CPI es el siguiente:

Tipo de Instrucción	Frecuencia	CPI
Ops ALU	43%	1
Cargas	21%	2
Almacenamientos	12%	2
Saltos	24%	2

Supongamos que el 25% de las operaciones con ALU utilizan un operando que se ha cargado desde memoria y que no vuelve a ser utilizado de nuevo.

Se propone aumentar el repertorio de instrucciones con instrucciones que utilizan la ALU en la que uno de sus operandos es obtenido directamente de memoria. Esta nueva instrucción registro-memoria tarda 2 ciclos de reloj (CPI=2). Suponer que el repertorio de instrucciones extendido incrementa en 1 el número de ciclos de las instrucciones de salto (CPI=3), pero no afecta al tiempo de ciclo. ¿Mejora este cambio el rendimiento del procesador?

Solución:

Para conocer si existe mejora en el rendimiento del procesador con este cambio debemos calcular la relación de tiempos de ejecución de las dos máquinas, con y sin mejora.

En la máquina original el tiempo de CPU es el siguiente: $t_{cpu}=N \times CPI \times T$; donde N es el número de instrucciones que desconocemos. El CPI (ciclo promedio por instrucción) lo calculamos multiplicando el porcentaje de cada tipo de instrucciones por su CPI correspondiente; es decir, el 43% × 1 ciclo de las instrucciones aritmético-lógicas, más el 21% × 2 ciclos de las instrucciones de carga, más el 12% × 2 ciclos de las instrucciones de almacenamiento, más el 24% × 2 ciclos de las instrucciones de salto. El T es el periodo de reloj que tampoco está indicado en el enunciado del problema. Por lo tanto, el tiempo de CPU de la máquina original es:

$$t_{cpu} = N \times (0.43 \times 1 + 0.21 \times 2 + 0.12 \times 2 + 0.24 \times 2) \times T = 1.57 \times N \times T$$

En la nueva máquina nos dice que el 25% de las operaciones aritmético-lógicas van a tener un operando cargado previamente que luego no se va a usar de nuevo. Por lo tanto, debemos calcular ese número de cargas para después restárselo al total de instrucciones de carga, ya que, con la nueva instrucción, esas cargas no se ejecutan: $0.25 \times 0.43 = 0.1075$. Es decir, el 10.75% × N son instrucciones de carga que vamos a eliminar porque se incluyen en la nueva instrucción.

Número de cargas en la nueva máquina: 0.21 - 0.1075 = 0.1025. Es decir, el 10.25% × N de las instrucciones siguen siendo instrucciones de carga en la nueva máquina.

En la máquina mejorada, el tiempo de CPU es el siguiente: $t_{cpu}' = N' \times CPI' \times T$

Calculamos el número de instrucciones de la nueva máquina en función de N:

$$N' = 43\% \times N + 10.25\% \times N \text{ (nuevo número de cargas)} + 12\% \times N + 24\% \times N =$$
$$= 0.43 \times N + 0.1025 \times N + 0.12 \times N + 0.24 \times N = 0.89 \times N$$

Calculamos el CPI' calculando primero por separado el CPI de cada tipo de instrucción:

- Instrucciones aritmético – lógicas (originales y nuevas):

$$CPI\,'(A-L)=\frac{(0.43-0.1075\,)N}{0.89\,N}\times 1+\frac{0.1075\;N}{0.89\,N}\times 2=0.6$$

El primer sumando es el porcentaje del total de las instrucciones aritmético-lógicas que no se ven sometidas al nuevo repertorio y por lo tanto siguen durando lo mismo. El segundo operando corresponde a las nuevas instrucciones aritmético-lógicas que tardan 2 ciclos.

- Instrucciones de carga:

$$CPI'(CARGA)=\frac{0.1025N}{0.89N}\times 2=0.23$$

- Instrucciones de almacenamiento:

$$CPI'(ALMACENAMIENTO)=\frac{0.12N}{0.89N}\times 2=0.27$$

- Instrucciones de salto: Como nos dice que en la nueva máquina las instrucciones de salto sufren un aumento de su CPI en uno, por lo que ahora pasan de tardar 2 ciclos a tardar 3 ciclos.

$$CPI'(SALTO)=\frac{0.24N}{0.89N}\times 3=0.81$$

Con lo cual el CPI' es la suma de todos los CPIs que será: CPI'(total) = 0.6+0.23+0.27+0.81 = 1.91

Por lo tanto, el tiempo de CPU para la máquina con la mejora será: t_{cpu}'= 0.89 × N × 1.91 × T, y la mejora existente entre las dos máquinas es:

$$Speed-Up=\frac{t_{CPU}}{t'_{CPU}}=0.92X$$

Esto indica que la mejora introducida por la nueva máquina **no proporciona** mejores prestaciones a la máquina original, sino al contrario, **empeora** las prestaciones del procesador.

Problema 1-1-3 (HP96, ex.1.5, pp.61; [KW96]pp.14). Suponer que se tiene una máquina con una memoria cache perfecta que se comporta como en la tabla del problema anterior.

Con una cache real, se ha medido que las instrucciones tienen una frecuencia de fallo del 5%, una frecuencia de fallo de datos del 10%, y una penalización por fallo de cache de 40 ciclos.

Encontrar el CPI para cada instrucción con estas tasas de fallos de cache, y determinar cuánto de rápida es la máquina sin fallos de cache con respecto a la que tiene fallos.

Solución:

El CPI de la máquina original, es decir, sin fallos de cache será el de la máquina del problema anterior 1-1-2: CPI(sin fallos) = 0.43 × 1 + 0.21 × 2 + 0.12 × 2 + 0.24 × 2 = 1.57

El CPI de la máquina con fallos de cache será igual al valor de una máquina sin fallos con el añadido de las penalizaciones por instrucciones introducidas por estos fallos. En primer lugar, tenemos fallos de la cache de instrucciones que ocurren en un 5% de los casos y cuya penalización es de 40 ciclos, por lo que habrá que sumarle al CPI el valor 0.05 × 40. Por otro lado, tenemos los fallos de cache de datos que ocurren en un 10% de las veces que se accede a ella y cuya penalización es de 40 ciclos. En el caso de la cache de datos, solo se accederá a ella cuando el tipo de instrucciones sean de cargas o almacenamiento, por lo que habrá que sumar al CPI una penalización de: [0.21 (cargas) + 0.12 (almacenamientos)] × 0.1 × 40.

Esto hace que el CPI con fallos de cache sea igual a:

CPI(con fallos) = 1.57 + penalizaciones por instrucción = 1.57 + 0.05 × 40 + (0.21 + 0.12) × 0.1 × 40 = 4.89

Para determinar cuánto más rápida es una máquina de otra utilizaremos el Speed-Up.

$$Speed-Up = \frac{t_{CPU}(máquina\ con\ fallos)}{t_{CPU}(máquina\ \text{sin}\ fallos)} = \frac{N \times 4.89 \times T}{N \times 1.57 \times T} = 3.1X$$

Esto indica un 211% de empeoramiento (4.89 – 1.57 / 1.57 = 2.11) en el tiempo de ejecución de los programas debido a fallos de cache. Por lo que se concluye que el hecho de que existan fallos de cache repercutirá mucho negativamente en las prestaciones de una máquina.

Problema 1-1-4 (HP03, Fig.1.17). Suponer que se dispone de 3 máquinas (A, B, y C) en las que se ejecutan 2 programas: P1 y P2. Se miden los tiempos de ejecución que se pueden observar en la siguiente figura, los cuales están normalizados al valor obtenido en una máquina determinada.

	Normalizado a A			**Normalizado a B**			**Normalizado a C**		
P1	**1.0**	**10.0**	**20.0**	**0.1**	**1.0**	**2.0**	**0.05**	**0.5**	**1.0**
P2	**1.0**	**0.1**	**0.02**	**10.0**	**1.0**	**0.2**	**50.0**	**5.0**	**1.0**
Tiempo total	**1.0**	**0.11**	**0.04**	**9.1**	**1.0**	**0.36**	**25.03**	**2.75**	**1.0**
Media aritmética									
Media geométrica									

Calcula la media aritmética y geométrica para cada máquina suponiendo los tres tipos de normalización mostrados. A partir de estos resultados, razona ¿cuál es la mejor forma de describir las prestaciones de un computador?.

Solución:

Utilizamos las siguientes expresiones para calcular las medias aritmética y geométrica.

- Media aritmética

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} a_i = (a_1 + \cdot$$

- Media geométrica

$$\bar{x} = \sqrt[n]{\prod_{i=1}^{n} x_i} = \sqrt[n]{x_1}$$

Los resultados se muestran en la siguiente tabla:

	Normalizado a A			Normalizado a B			Normalizado a C		
	A	B	C	A	B	C	A	B	C
P1	1.0	10.0	20.0	0.1	1.0	2.0	0.05	0.5	1.0
P2	1.0	0.1	0.02	10.0	1.0	0.2	50.0	5.0	1.0
Tiempo total	1.0	0.11	0.04	9.1	1.0	0.36	25.03	2.75	1.0
Media aritmética	1.0	5.05	10.01	5.05	1.0	1.1	25.03	2.75	1.0
Media geométrica	1.0	1.0	0.63	1.0	1.0	0.63	1.58	1.58	1.0

Tal como se puede apreciar con la media aritmética el resultado depende de la máquina a la que se normalice. Siempre se beneficia el resultado en la máquina que se utiliza para normalizar los resultados. Cuando se normaliza a los resultados medidos en A, la máquina A es la que menor promedio aritmético exhibe; cuando se normaliza con los datos de la máquina B, ésta es la mejor; al igual que ocurre con la

máquina C. Por lo que esta medida es dependiente de la máquina que se emplea para normalizar los resultados.

Por otro lado, observando los resultados obtenidos con la media geométrica, se concluye que la máquina C es la mejor independientemente de la máquina de referencia. El problema de esta medida es que no podemos tenerla en cuenta de forma cuantitativa.

Por lo tanto y como conclusión, a la hora de medir prestaciones, es mejor utilizar la medida geométrica que la media aritmética, aunque no podemos considerar los resultados de la media geométrica de forma cuantitativa.

PROBLEMA 1-1-5. Este problema aborda el análisis de prestaciones de un procesador.

a) Nombrar los 3 componentes principales de la ecuación del tiempo de ejecución. ¿Cómo se combinan para cuantificar dicho tiempo de ejecución?

Solución:

Para calcular el tiempo de ejecución debemos tener en cuanta los siguientes componentes:

- Número de Instrucciones (N): El número de instrucciones depende del Programa que se este ejecutando, del compilador utilizado y del repertorio de instrucciones.
- Ciclos promedio por instrucción (CPI): El CPI a su vez dependerá del Compilador utilizado, del repertorio de instrucciones y de la organización del hardware.
- Frecuencia de reloj (T): Dependerá de la organización del hardware y de la tecnología utilizada.

Para calcular el tiempo de ejecución del programa tenemos que utilizar la siguiente fórmula:

$$\text{Tiempo CPU} = N \times CPI \times T$$

Para calcular el CPI utilizaremos a su vez la siguiente fórmula:

$$CPI = \text{CPI ideal} + \text{Nº Promedio ciclos de reloj de parada por instrucción}$$

b) ¿Cuál es el CPI y la medida MIPS de la máquina? Suponer una frecuencia de reloj de 300 MHz y las siguientes medidas estadísticas:

Tipo de instrucción	Frecuencia (%)	Ciclos
ALU	**40**	**1**
Carga/Almacenamiento	**30**	**2**
Saltos	**20**	**3**
Coma flotante	**10**	**5**

Solución:

El CPI de la máquina es el siguiente:

$$CPI = (0{,}4 \times 1 + 0{,}3 \times 2 + 0{,}2 \times 3 + 0{,}1 \times 5) = 2{,}1$$

Para calcular los MIPS debemos utilizar la siguiente fórmula:

$$\begin{aligned} MIPS &= \text{Número de Millones de Instrucciones Ejecutadas / Tiempo de Ejecución} = \\ &= \text{Frecuencia de Reloj} / (CPI \times 10^6) = \\ &= 142.9 \text{ MIPS} \end{aligned}$$

PROBLEMA 1-1-6. Calcula los valores de SPECrate2017_fp_base y SPECrate2017_fp_peak del computador ASUS RS720-E9(Z11PP-D24) Server System a partir de las medidas de prestaciones del benchmark SPEC CPU 2017 que se proporcionan a continuación (referencia: https://www.spec.org/cpu2017/results/res2018q3/cpu2017-20180709-07697.txt)**.**

SPECrate2017_fp_base = media geométrica (Base Rate)
SPECrate2017_fp_peak = media geométrica (Base Peak)

Rate: copies * (Tiempo en el ordenador de referencia / Run Time)

	Benchmark	Base # Copies	Base Run Time	Base Rate	Peak # Copies	Peak Run Time	Peak Rate
503	bwaves_r	112	2259	497	112	2258	497
507	cactuBSSN_r	112	543	261	112	545	259
508	namd_r	112	404	263	112	404	262
510	parest_r	112	2341	125	112	2338	125
511	povray_r	112	616	424	112	535	488
519	lbm_r	112	969	121	112	965	122
521	wrf_r	112	1124	223	112	1121	223
526	blender_r	112	473	360	112	473	360
527	cam4_r	112	611	320	112	609	321
538	imagick_r	112	361	771	112	361	770
544	nab_r	112	322	584	112	322	584
549	fotonik3d_r	112	2698	161	112	2701	161
554	roms_r	112	1849	96	112	1810	98
		SPECrate2017_fp_base		**269,3**	SPECrate2017_fp_peak		**272,6**

SOLUCIONES TEMA 1-2

Problema 1-2-8 (Examen Parcial 2005). Realiza la operación de multiplicación de los números "-8" y "-8" utilizando el algoritmo de Booth.

Solución:

-8 = 11000

Paso	Resultado	Multiplicando	Q0	Multiplicador	Observaciones
1	00000	1100 **0**	**0**	11000	Res&Muldo&Q0 >> 1
2	00000	0110 **0**	**0**	11000	Res&Muldo&Q0 >> 1
3	00000	0011 **0**	**0**	11000	Res&Muldo&Q0 >> 1
4	00000	0001 **1**	**0**	11000	Res&Muldo&Q0 >> 1
5.a	01000	0001 **1**	**0**	11000	Res=Res-Muldor
5.b	00100	0000 **1**	**1**		Res&Muldo&Q0 >> 1
6	00010	0000 **0**	**1**		Res&Muldo&Q0 >> 1
	00010	00000			

Resultado: 00010 00000

SOLUCIONES TEMA 1-3

Problema 1-3-1 (HP96, ex.3.1, pp.214). Utilizar el siguiente fragmento de código:

```
loop:   LW   R1,0(R2)
        ADDI R1,R1,#1
        SW   R1,0(R2)
        ADDI R2,R2,#4
        SUB  R4,R3,R2
        BNZ  R4, loop
```

Suponer que el valor inicial de R3 es R2+396. Suponer también que se utiliza la implementación segmentada del procesador DLX32p de cinco etapas para enteros (IF, ID, EX, MEM, WB), y que todos los accesos a memoria cache son aciertos. Además, suponer que los saltos se resuelven y actualizan el PC al final de la etapa MEM, y que no existen saltos retardados.

a. Mostrar la temporización de esta secuencia de instrucciones para la segmentación del DLX32p suponiendo que no se utiliza anticipación y que en un mismo ciclo de reloj se puede realizar una lectura y escritura en el banco de registros. Suponer también que en los saltos condicionales se predice que el salto es no-tomado, y si la predicción falla, se eliminan las instrucciones que se han introducido posteriormente al salto. Si todas las referencias a memoria aciertan en la cache, ¿cuántos ciclos tarda el bucle en ejecutarse?

Solución:

Debemos determinar el número de ciclos que tarda en ejecutarse el código anterior en un procesador con las siguientes características:

- Procesador segmentado de 5 etapas con una única unidad funcional para enteros.
- Todos los accesos a memoria cache son aciertos, lo que quiere decir que el tiempo de acceso a memoria es de un solo ciclo.
- El salto se resuelve en la etapa MEM (un ciclo después de el que hemos estudiado en clase).
- No hay saltos retardados y se predice "salto no tomado", por lo que después de un salto comienza a ejecutarse siempre la siguiente instrucción, pero se eliminará del cauce si el salto fuera tomado.
- Se puede leer y escribir en el banco de registros es un mismo ciclo de reloj; por lo tanto, permite la anticipación de un dato que genera una instrucción que se encuentra en la etapa WB a otra instrucción que requiere de ese dato y se encuentra en la etapa ID.
- No existe ningún otro tipo de anticipación de datos.

A continuación, se muestra el diagrama temporal con la evolución de las instrucciones dentro del cauce del procesador. En el eje horizontal están representados los ciclos de reloj, y en el eje vertical la instrucción que se ejecuta. El número que aparece a la izquierda de la instrucción es el orden que le corresponde en el código fuente. Por ejemplo, la séptima instrucción en ejecutarse será la instrucción número 1 del código por tratarse de un bucle. Para cada instrucción y cada ciclo de reloj, el diagrama temporal indica en qué etapa de la segmentación se encuentra: IF representa a la etapa de búsqueda, ID a la de decodificación, EX a la de ejecución, M a la de acceso a la memoria de datos, y WB a la de post-escritura. Un signo de admiración ("!") junto a las siglas de una etapa de segmentación representa que la instrucción se encuentra parada en esa etapa debido a dependencias de datos, de control, o bien porque el cauce se encuentre paralizado por alguna instrucción anterior.

Ciclos	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
1. LW R1,0(R2)	IF	ID	EX	M	WB														
2. ADDI R1,R1,#1		IF	ID!	ID!	ID	EX	M	WB											
3. SW R1,0(R2)			IF!	IF!	IF	ID!	ID!	ID	EX	M	WB								
4. ADDI R2,R2,#4						IF!	IF!	IF	ID	EX	M	WB							
5. SUB R4,R3,R2									IF	ID!	ID!	ID	EX	M	WB				
6. BNZ R4, loop										IF!	IF!	IF	ID!	ID!	ID	EX	M	WB	
1. LW R1,0(R2)													IF!	IF!	IF!	IF!	IF!	IF'	

17 ciclos

IF' corresponde a una instrucción distinta a la que se introduce en el cauce y que se representa por IF.

1. LW R1,0(R2) :

   Carga en R1 el dato contenido de la dirección '0(R2)' de memoria. Se ejecuta en 5 ciclos sin ninguna penalización.

2. ADDI R1,R1,#1 :

   Tiene una dependencia de datos con la instrucción anterior, y hasta el ciclo 5 no estará disponible el valor en R1 por lo que no podrá leerlo hasta ese momento. En este ciclo 5 podrá leer el dato, ya que el banco de registros permite leer y escribir en un mismo ciclo. Después de este ciclo la instrucción evoluciona en el cauce sin ninguna parada.

3. SW R1,0(R2) :

   Entra a ejecución en el ciclo 3 pero se detiene inmediatamente debido a que en la etapa ID se encuentra una instrucción anterior parada. En el ciclo 5 ya puede avanzar, pero en el ciclo 6 debe pararse otra vez debido a una dependencia de datos en R1 con la instrucción anterior ADDI. Como ocurrió antes, no puede continuar ejecutándose hasta que la instrucción 2 se encuentre en la etapa de WB, lo cual ocurre en el ciclo 8.

4. ADDI R2,R2,#4 :

   Entra a ejecución en el ciclo 6 pero se para inmediatamente debido a que en la etapa ID se encuentra parada una instrucción anterior. No tiene ninguna dependencia de datos con otra instrucción anterior.

5. SUB R4,R3,R2 :

   Tiene una dependencia de datos con la instrucción anterior número 4 en R2, por lo que detiene su ejecución durante los ciclos 10 y 11.

6. BNZ R4, loop :

   Entra a ejecución en el ciclo 10 pero se para inmediatamente debido a que en la etapa ID se encuentra parada una instrucción anterior. Además, tiene una dependencia de datos con la instrucción anterior en R4. Por todo ello, se detiene su ejecución durante los ciclos 10 y 11, 13 y 14. Cuando esta instrucción se encuentra en la etapa MEM es cuando se calcula el valor que se inicializará en el registro Contador de Programa, lo cual se realiza en el ciclo 18.

En el ciclo 13 entrará a ejecución la instrucción 7 del código fuente, pero al llegar la instrucción de salto a la etapa MEM (ciclo 17) se determina en la instrucción de salto condicional la dirección de la siguiente instrucción que debe entrar en el cauce: la instrucción 1. Por tanto, se elimina la instrucción 7 del cauce, y en el ciclo 18 comienza a ejecutarse de nuevo la instrucción 1.

Cada iteración comprende 17 ciclos (ver tabla anterior), contabilizando el ciclo 18 como perteneciente a la siguiente iteración, en el que se solapan la etapa WB de la instrucción 6 y la IF de la instrucción 1.

Este solapamiento se produce en todas las iteraciones excepto en la última iteración en la que no hay solapamiento. Por ello, cada iteración consta de 17 ciclos, menos la última que se compone de 18 ciclos.

La condición de salida del bucle consiste en que R3 sea igual a R2 (R3-R2=0). Inicialmente R3=R2+396 y en cada iteración vamos aumentando R2 de 4 en 4. ¿Cuántas veces podemos sumar 4 a R2 para llegar a R2 + 396?: 396/4 = 99. Es decir, el bucle se repetirá 99 veces.

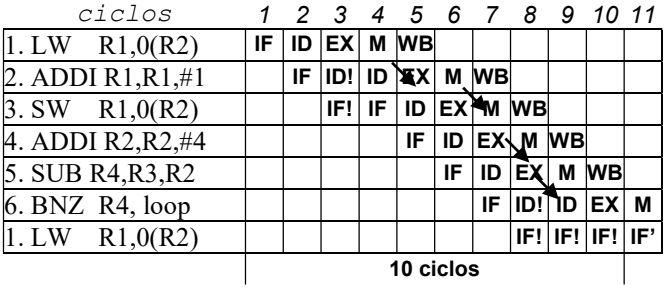
Calculamos entonces el número de ciclos ejecutados de la siguiente forma:
17 ciclos/iteración × 99 iteraciones + 1 ciclo (ya que en la última iteración se ejecutan 18 ciclos en lugar de 17)

$t = 17 \times 99 + 1 = 1684$ ciclos

b. Mostrar la temporización de esta secuencia de instrucciones para el DLX32p segmentado suponiendo la aplicación de la técnica de anticipación. Suponer que en los saltos condicionales se predice que el salto es no tomado. Además, suponer que los saltos comprueban la condición en la etapa ID y se resuelven y actualizan el PC al final de la etapa EX, y que no existen saltos retardados. Si todas las referencias a memoria aciertan en la cache, ¿cuántos ciclos tarda el bucle en ejecutarse?

Solución:

Las condiciones son las mismas que para el apartado anterior sólo que ahora se introduce la técnica de anticipación de datos, que consiste en enviar el dato desde la etapa MEM o WB a la EX cuando sea necesario.



2. ADDI R1,R1,#1:
   Tiene una dependencia de datos en R1 con la instrucción anterior (LW). Pese a haber anticipación de datos, la ejecución debe detenerse un ciclo en la etapa ID, ya que al ser la instrucción anterior una instrucción de carga (LW), no se obtendrá el dato hasta la etapa MEM (ciclo 4). En el ciclo 5 se produce la anticipación del dato desde la etapa WB de LW a la EX de ADDI. La penalización ha sido de 1 solo ciclo, frente a los dos ciclos de penalización en la ejecución sin anticipación de datos.
3. SW R1,0(R2):
   Entra a ejecución en el ciclo 3 pero se detiene inmediatamente debido a que la etapa ID se encuentra paralizada por la instrucción anterior. En el ciclo 4 puede avanzar sin ninguna penalización, ya que a pesar de tener una dependencia de datos en R1 con la instrucción anterior, la anticipación permite que el dato esté disponible para esta instrucción en el ciclo 6.
5. SUB R4,R3,R2 :
   Tiene una dependencia de datos con la instrucción anterior en R2, pero no necesita detener su ejecución, ya que la anticipación de datos permite que el valor de R2 esté disponible en el ciclo 8.
6. BNZ R4, loop :

Tiene una dependencia de datos con la instrucción anterior en R4. Sí necesita detener su ejecución en la etapa ID porque la condición BNZ se comprueba en esa etapa ID y tiene que esperar a que R4 se haya generado. La anticipación de datos permite que el valor de R4 esté disponible en el ciclo 9 para calicular la dirección destino del salto en el ciclo 10. Cuando esta instrucción se encuentra en la etapa EX es cuando se calcula el valor que se inicializará en el registro Contador de Programa, lo cual se realiza en el ciclo 11.

Las únicas penalizaciones son las producidas por la dependencia de una instrucción de carga (LW), y los 3 ciclos de penalización que produce el salto tomado. Como se puede observar en la tabla anterior, cada iteración se ejecuta en 10 ciclos, excepto la última que lo hará en 12 ciclos.

$$t = 10 \times 99 + 2 = 992 \text{ ciclos} \quad \rightarrow \quad \text{Speed-Up} = 1684 / 992 = 1.7X$$

Conclusión: Sólo introduciendo anticipación, la mejora respecto al apartado (a) en el tiempo de ejecución es de un factor 1.7X, o alternativamente, se ha reducido el tiempo de ejecución en 41%=100 × (1684-992) / 1684. También podemos decir que el tiempo de partida del apartado (a) es un 70% mayor que el tiempo conseguido con las mejoras de este aparado (b): 70% = 100 × (1684 - 992) / 992.

c. Suponiendo que se tiene un procesador DLX32p segmentado cuyos saltos condicionales se resuelven ahora en la etapa ID y están retardados en 1 ciclo, y que además se ha implementado la técnica de anticipación, planifica las instrucciones del bucle incluyendo los saltos retardados. Para ello se pueden reordenar las instrucciones y modificar los operandos de las instrucciones, pero no se deben modificar los códigos de operación ni el número de instrucciones. Mostrar el diagrama temporal de los segmentos y el número de ciclos necesarios para ejecutar todo el bucle.

Solución:

El enunciado nos dice que:
- El salto se resuelve en la etapa ID en lugar de la etapa EX. Por lo tanto, existe un solo ciclo de penalización en vez de 2 ciclos.
- Salto retardado en un ciclo; quiere decir que la siguiente instrucción al salto siempre se ejecuta.

Reordenamos el código de la siguiente forma:
- Introducimos las instrucciones 4 y 5 entre la 1 y la 2 para evitar la detención por dependencia de datos en R1.
- Aprovechamos el ciclo que se necesita para el cálculo de la dirección de salto (salto retardado) moviendo la instrucción de almacenamiento (3) después del salto (6), ya que la siguiente instrucción al salto siempre se ejecuta.

ciclos	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*
1. LW R1,0(R2)	**IF**	**ID**	**EX**	**M**	**WB**							
4. ADDI R2,R2,#4		**IF**	**ID**	**EX**	**M**	**WB**						
5. SUB R4,R3,R2			**IF**	**ID**	**EX**	**M**	**WB**					
2. ADDI R1,R1,#1				**IF**	**ID**	**EX**	**M**	**WB**				
6. BNZ R4, loop					**IF**	**ID**	**EX**	**M**	**WB**			
3. SW R1,-4(R2)						**IF**	**ID**	**EX**	**M**	**WB**		
1. LW R1,0(R2)							**IF**	**ID**	**EX**	**M**	**WB**	
	6 ciclos											

Con la reordenación se ha conseguido que no haya ningún ciclo de parada. Cada iteración se ejecuta en 6 ciclos, excepto la última que lo hará en 10.

$$t = 6 \times 99 + 4 = 598 \text{ ciclos} \quad \rightarrow \quad \text{Speed-Up} = 1684 / 598 = 2.8X$$

Conclusión: La mejora respecto al apartado (a) en el tiempo de ejecución es de un factor 2.8X, o alternativamente, se ha reducido el tiempo de ejecución en 64%=100 × (1684-598) / 1684. También podemos decir que el tiempo de partida del apartado (a) es un 180% mayor que el tiempo conseguido con las mejoras de este aparado (c): 180% = 100 × (1684 - 598) / 598.

Problema 1-3-2 (HP96, ex.3.3, pp.216). Exploraremos aquí las técnicas de segmentación para una arquitectura memoria-registro. La arquitectura tiene dos formatos de instrucción: un formato registro-registro y un formato registro-memoria. Existe un modo de direccionamiento a memoria para las operaciones registro-memoria que es del tipo: inmediato + registro base.

Se dispone de un conjunto de operaciones ALU con los siguientes formatos:

Formato 1: `ALUop Rdest, Rsrc1, Rsrc2`
Formato 2: `ALUop Rdest, Rsrc1, MEM`

Donde `ALUop` representa a uno de las siguientes operaciones: Add, Subtract, And, Or, Load (ignorando Rsrc1 en el formato 2), Store. `Rsrc` o `Rdest` son registros. MEM representa al par (registro base, inmediato).

Los saltos condicionales utilizan una comparación entre el contenido de dos registros en la etapa ALU2, y el salto se realiza relativo al Contador de Programa (PC). Suponer que la máquina es segmentada y escalar. Por lo tanto, una nueva instrucción comienza a ejecutarse cada ciclo de reloj. Su microarquitectura segmentada evoluciona ciclo a ciclo de la siguiente manera (parecida al VAX 8700):

CLK1	CLK2	CLK3	CLK4	CLK5	CLK6	CLK7	CLK8	CLK9	CLK10	CLK11
IF	RF	ALU1	MEM	ALU2	WB					
	IF	RF	ALU1	MEM	ALU2	WB				
		IF	RF	ALU1	MEM	ALU2	WB			
			IF	RF	ALU1	MEM	ALU2	WB		
				IF	RF	ALU1	MEM	ALU2	WB	
					IF	RF	ALU1	MEM	ALU2	WB

La primera etapa de la ALU (ALU1) se utiliza para el cálculo de la dirección de las referencias a memoria y de las direcciones de salto. El segundo ciclo ALU (ALU2) se utiliza para realizar las operaciones ALU, así como las comparaciones asociadas a los saltos. RF es la etapa de decodificación y de búsqueda de los operandos. Suponer que se pueden realizar lecturas y escrituras al mismo registro en el mismo ciclo de reloj.

a. Encontrar el número necesario de sumadores y mostrar una combinación de instrucciones y etapas de segmentación que justifique la respuesta. Sólo se requiere mostrar una sola combinación que requiera todos los sumadores.

Solución:

Necesitará 3 sumadores:
- uno en la etapa IF para el cálculo del PC+4
- uno en la etapa ALU1 para el cálculo de la dirección de memoria o la dirección destino del salto.
- uno en la etapa ALU2 para realizar las operaciones de las instrucciones aritméticas.

En la siguiente tabla se puede observar una combinación de instrucciones que requieren utilizar de los tres sumadores en un mismo ciclo de reloj (ciclo 5).

ciclos	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*
1. ADDI R2,R2,R5	IF	RF	ALU1	M	ALU2	WB				
2. SUB R4,R3,R2		IF	RF	ALU1	M	ALU2	WB			
3. LW R1,0(R2)			IF	RF	ALU1	M	ALU2	WB		
4. ADDI R1,R1,R5				IF	RF	ALU1	M	ALU2	WB	
5. BNE R3,R2, loop					IF	RF	ALU1	M	ALU2	WB

b. Encontrar el número de puertos de lectura y de escritura que se necesitan del banco de registros, así como el número de puertos de lectura y escritura que se requieren de la memoria.

Solución:

El banco de registros necesitará 2 puertos de lectura, para leer Rsrc1 y Rsrc2 o el registro base de la dirección de memoria (MEM), dependiendo del formato de instrucción, en la etapa RF. Y un puerto de escritura para escribir en el registro destino en la etapa WB.

La memoria necesitará 2 puertos de lectura, suponiendo una memoria conjunta de datos e instrucciones; un puerto para leer la instrucción en la etapa IF, y otro puerto para acceder a la posición de memoria que corresponda para las instrucciones de carga y ALUop de formato 2, en la etapa MEM; y un solo puerto de escritura, para las instrucciones de instrucciones de almacenamiento que escriben en memoria en la etapa MEM.

c. Determinar las anticipaciones de datos que se necesitan entre cualquiera de las ALUs. Suponer que existen ALUs separadas para las etapas ALU1 y ALU2. Especificar las conexiones necesarias entre las ALUs para implementar las anticipaciones y así reducir las paradas del flujo de segmentación.

Solución:

Hay dos casos de dependencias que pueden originar penalizaciones:
- Caso 1: El resultado de una operación que se calcula en la ALU2 y lo necesita otra instrucción para realizar otra operación en ALU2 (ambos tipos de instrucciones AluOp, saltos condicionales)
- Caso 2: El dato resultado de una operación que se calcula en la ALU2 y lo necesita otra instrucción para realizar otra operación en ALU1 (AluOp en formato 2, LW, SW, saltos)

La otra posible dependencia se podría deber a las direcciones de memoria calculadas en ALU1, pero como no deben guardarse en registros, no generan dependencias. Estas direcciones las utiliza la misma instrucción que la generó en la etapa de acceso a memoria.

Las combinaciones de instrucciones que hay que estudiar para detectar los riesgos por dependencias son las instrucciones: i, i+1, i+2, e i+3. Entre las instrucciones i e i+4 no existen riesgos debido a que el banco de registro permite una lectura y escritura en el mismo ciclo de reloj.

En el Caso 1, la dependencia se resuelve implementando anticipación desde ALU2 a ALU2 entre la instrucción i y las instrucciones dependientes siguientes: i+1, i+2, i+3. Para ello, se utilizan registros que guarden los resultados de ALU2 para los ciclos siguientes. En la siguiente tabla se muestran un ejemplo de anticipaciones, la cuales se representan con flechas.

Ciclos	1	2	3	4	5	6	7	8	9	10
1. ADDI R2, R2, #4	IF	RF	ALU1	M	ALU2	WB				
2. SUB R4, R3, R2		IF	RF	ALU1	M	ALU2	WB			
3. AND R1, R3, R2			IF	RF	ALU1	M	ALU2	WB		
4. OR R5, R3, R2				IF	RF	ALU1	M	ALU2	WB	
5 AND R5, R5,R2					IF		RF	ALU1	M	ALU2	WB

En el Caso 2, no es posible resolver por completo la dependencia usando anticipación. Cuando existe esta dependencia entre dos instrucciones consecutivas, la instrucción más nueva se encuentra en la etapa ALU1 un ciclo antes que la instrucción más vieja se encuentre en la etapa ALU2. Por ello, debe pararse el cauce hasta que la instrucción más vieja haya generado el dato necesario. En esta situación, la penalización es de dos ciclos de reloj. La siguiente tabla muestra esta situación entre las instrucciones 1 y 2.

Ciclos	1	2	3	4	5	6	7	8	9	10	11	12
1. ADDI R2, R2, #4	IF	RF	ALU1	M	ALU2	WB						
2. SUB R4, R3, 0(R2)		IF	RF	ALU1!	ALU1!	ALU1	M	ALU2	WB			
3. ADDI R3, R1, 0(R2)			IF	RF!	RF!		RF	ALU1	M	ALU2	WB	
4. SW R2, 0(R4)				IF!	IF!	IF	RF	ALU1!	ALU1	M	ALU2	WB

Cuando la dependencia de este tipo 2 aparece entre la instrucción i e i+2 y se implementa la anticipación cuando es posible desde ALU2 a ALU1, el número de ciclos de parada se reduce a 1. En la tabla anterior podemos observar un ejemplo para esta segunda situación del Caso 2 entre las instrucciones 2 y 4.

d. Mostrar todos los requerimientos de anticipación de datos que son necesarios cuando las unidades fuentes o destinos no son ALUs.

Solución:

En la siguiente tabla se describen los distintos casos de dependencias que solicita el apartado.

			Posibles riesgos entre instrucciones y solución		
Caso	**Dependencia**	**Descripción**	**i→i+1**	**i→i+2**	**i→i+3**
1	MEM→ALU1	Carga dato en MEM para computar dirección efectiva en ALU1	parada	Anticipa desde MEM	Anticipa desde WB
2	MEM→ALU2	Carga dato en MEM para realiza operación en ALU2	Anticipación desde ALU2 a ALU2	Anticipación desde WB a ALU2	Anticipación desde ALU2 a ALU1
3	MEM→MEM	Carga dato en MEM para realizar almacenamiento en MEM	Anticipación desde MEM a MEM	Anticipación desde MEM a ALU1	Anticipación desde ALU2 a ALU1
4	ALU2→ MEM	Realizar operación en ALU2 para almacenar dato en MEM	parada	Anticipación desde ALU2 a MEM	Anticipación desde ALU2 a ALU1

En las siguientes tablas se pueden observar respectivamente ejemplos de los casos 1, 2, 3 y 4.

Ciclos	1	2	3	4	5	6	7	8	9
1. LW R1,0(R2)	IF	RF	ALU1	M	ALU2	WB			
2. LW R3,0(R1)		IF	RF	ALU1!	ALU1	M	ALU2	WB	

Ciclos	1	2	3	4	5	6	7	8	9	10
1. LW R1,0(R2)	IF	RF	ALU1	M	ALU2	WB				
2. SUB R4, R3, R1		IF	RF	ALU1	M	ALU2	WB			

Ciclos	1	2	3	4	5	6	7	8	9
3. LW R2,0(R3)			IF	RF	ALU1	M	ALU2	WB	
4. SW R2, 0(R5)				IF	RF	ALU1	M	ALU2	WB

Ciclos	1	2	3	4	5	6	7	8	9	10
3. ADD R1, R3, R2			IF	RF	ALU1	M	ALU2	WB		
4. SW R1, 0(R2)				IF	RF	ALU1	M!	M	ALU2	WB

e. Mostrar el resto de las dependencias de datos que no puedan resolverse por anticipación y que impliquen al menos que una de las unidades fuente o destino no sea una ALU. Determinar el número de ciclos de paradas necesarios.

Solución:

Los siguientes casos son los que requieren parada del cauce, y además una de las unidades funcionales implicadas no es alguna de las ALU.

Caso	Dependencia	Descripción	i→i+1
1	MEM→ALU1	Carga dato en MEM para computar dirección efectiva en ALU1	parada
2	ALU2→ MEM	Realizar operación en ALU2 para almacenar dato en MEM	parada

A continuación, se muestra un ejemplo para el Caso 1. Si el dato se obtiene en la etapa MEM (a través de una instrucción LW) y se necesita en ALU1 (por alguna de las siguientes instrucciones: AluOp en formato 2, LW, SW, saltos) se necesita un ciclo de parada. Con la flecha se representan el camino de anticipación que permite enviar el dato después de realizar la parada.

Ciclos	1	2	3	4	5	6	7	8
1. LW R1,0(R2)	IF	RF	ALU1	M	ALU2	WB		
2. SUB R4, R3, 0(R1)		IF	RF	ALU1!	ALU1	M	ALU2	WB

A continuación, se muestra un ejemplo para el Caso 2. Si el dato se calcula en la etapa ALU2 (AluOp) y se necesita en la etapa MEM (SW), sigue necesitando un ciclo de parada.

Ciclos	1	2	3	4	5	6	7	8
1. SUB R2, R3, R1	IF	RF	ALU1	M	ALU2	WB		
2. SW R2, 0(R5)		IF	RF	ALU1	M!	M	ALU2	WB

f. Mostrar todos los tipos de dependencias de control a través de ejemplos y especifica la longitud de la parada.

Solución:

Saltos Incondicionales: Los saltos incondicionales calculan la dirección destino del salto en la etapa ALU1. Por lo tanto, hasta el ciclo 4 de la instrucción de salto no se conocer la dirección destino del salto, esta es la razón por la que se necesitan 2 ciclos de parada. En la siguiente tabla se puede observar un ejemplo.

Ciclos	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*
1. J salto	**IF**	**RF**	**ALU1**	**M**	**ALU2**	**WB**			
8. salto: SUB R4, R3, 0(R1)		**IF!**	**IF!**	**IF'**	**RF'**	**ALU1'**	**M'**	**ALU2'**	**WB'**

Saltos Condicionales: Los saltos condicionales requieren comparar dos registros en la etapa ALU2; es decir, el salto se resuelve en la etapa ALU. Por lo tanto, hasta el ciclo 6 de la ejecución de un salto no se puede conocer la dirección de la siguiente instrucción a un salto. Esto implica que habría cuatro instrucciones que podrían insertarse en el cauce antes de saber si se salta o no. Cuando se produce el salto, la penalización es de cuatro ciclos como se puede observar en la siguiente tabla.

Ciclos	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*
1. BEQ R1,R2 salto	**IF**	**RF**	**ALU1**	**M**	**ALU2**	**WB**					
8. salto: SUB R4, R3, 0(R5)		**IF!**	**IF!**	**IF!**	**IF!**	**IF'**	**RF'**	**ALU1'**	**M'**	**ALU2'**	**WB'**

Problema 1-3-3 (HP96, example, pp.137). Suponer que un procesador multiciclo no segmentado tiene un ciclo de reloj de 10 ns. y que utiliza 4 ciclos para operaciones ALU y saltos condicionales, y 5 ciclos para operaciones de memoria. Suponer que las frecuencias relativas de estas operaciones son 40%, 20% y 40% respectivamente. Suponer que debido al "clock skew" de la señal de reloj y al "setup" de los registros de segmentación, el procesador segmentado aumenta en 1 ns el ciclo de reloj. ¿Cuánto Speed-Up en el tiempo de ejecución de las instrucciones se obtendrá con la segmentación respecto a la implementación multiciclo?

Solución:

La mejora de prestaciones (Speed-Up) se cuantifica con la ecuación del tiempo: $t_{cpu} = N \times CPI \times T$. Para ello, tenemos que comparar el tiempo de CPU de los dos procesadores: multiciclo ($t_{cpu\text{-}multiciclo}$) y segmentado ($t_{cpu\text{-}segmentada}$).

Para la máquina multiciclo:
$T_{multiciclo}$ (tiempo del periodo de reloj) = 10 ns
$CPI_{multiciclo}$ (número promedio de ciclos de reloj por instrucción) = $4 \times (0.4 + 0.2) + 5 \times 0.4 = 4.4$ ciclos.

Para la máquina segmentada:
$T_{segmentada}$ = 11 ns (debido a la propagación de la señal de reloj (clock skew) y al tiempo de estabilizaciñon (setup))
$CPI_{segmentada}$ = 1 ciclo promedio por instrucción

Speed-Up = $t_{cpu\text{-}multiciclo} / t_{cpu\text{-}segmentada} = N \times CPI_{multiciclo} \times T_{multiciclo} / N \times CPI_{segmentada} \times T_{segmentada}$

Speed-Up = $N \times 4.4 \times 10 / N \times 1 \times 11 = 44 / 11 = 4X$

Conclusión: El aumento de prestaciones de la máquina segmentada con respecto a la multiciclo es de un factor 4X, o lo que es lo mismo, la reducción del tiempo de ejecución que proporciona la máquina segmentada es del 75% = 100 × (44 ns – 11 ns) / 44 ns = 75%. También podemos decir que el tiempo de ejecución en la máquina multiciclo respecto a la segmentada es de un 300% = 100 × (44 ns – 11 ns) / 11 ns.

Problema 1-3-4 (HP96, ex.3.4, pp.217). Suponer que se tiene una situación como la del problema anterior en la que el overhead en cada uno de los segmentos del procesador es de 1 ns. Suponer también que cada una de las 5 etapas en que está dividido el procesador tarda 10 ns. sin incluir el overhead. Dibujar el Speed-Up de la máquina segmentada en relación a la no segmentada a medida que el número de segmentos se incrementa hasta un total de 20, considerando sólo el impacto del overhead y suponiendo que el trabajo puede ser homogéneamente dividido a medida que el número de etapas aumenta (lo cual no es completamente cierto). Dibujar además el Speed-Up perfecto que se obtendría si no existiera overhead.

Solución:

Inicialmente el procesador segmentado está dividido en 5 etapas, cada una de 10 ns de duración. Por ello, 50 ns es el tiempo de ejecución de una instrucción al atravesar todas las etapas de segmentación. Iremos dividiendo la segmentación en más etapas, de forma que el tiempo de ciclo (o periodo del reloj) será:

$T_{segmentada}$ = 50 ns / número de etapas.

A este cálculo hay que sumarle en cada ciclo de reloj un overhead de 1 ns, debido al tiempo que necesita cada registro de segmentación para considerar que en su entrada el dato se encuentra estable. Por lo tanto, el periodo de reloj de la máquina segmentada es:

$T_{segmentada}$ = 1 + (50 ns / número de etapas).

Calcularemos para cada número de etapas posibles dentro del procesador segmentado, de 1 á 20, el tiempo de ciclo, tanto con como sin overhead, para luego calcular el Speed-Up con respecto a la máquina multiciclo y representarlo en forma de gráfica.

$Speed\text{-}Up_{overhead} = t_{cpu\text{-}multiciclo} / t_{cpu\text{-}segmentada} = N \times CPI_{multiciclo} \times T_{multiciclo} / N \times CPI_{segmentada} \times T_{segmentada}$

Suponemos un CPI mínimo para el procesador segmentado, $CPI_{segmentada}$ = 1 ns. Calculamos ahora el $Speed\text{-}Up_{overhead}$ para cada número de etapas. Utilizando los datos del problema anterior:

$Speed\text{-}Up_{overhead}$ = 44 ns / [1 + (50 ns / número de etapas)]

Etapas = 1:
- sin overhead: $T_{segmentada} = 50/1 = 50$ ns → $Speed\text{-}Up_{ideal} = 44 / 1 \times 50 = 0.88X$
- con overhead: $T_{segmentada} = (50/1)+1 = 51$ ns → $Speed\text{-}Up_{overhead} = 44 / 51 = 0.86X$

Etapas = 5:
- sin overhead: $T_{segmentada} = 50/5 = 10$ ns → $Speed\text{-}Up_{ideal} = 44 / 10 = 4.4X$
- con overhead: $T_{segmentada} = (50/5)+1 = 11$ ns → $Speed\text{-}Up_{overhead} = 44 / 11 = 4X$

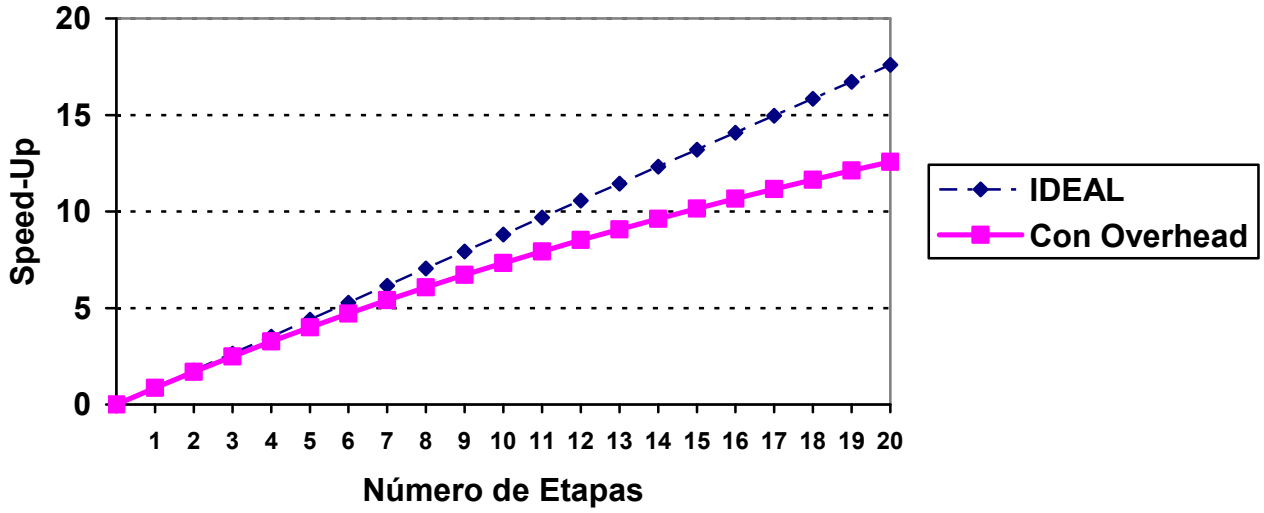
Etapas = 10:
- sin overhead: $T_{segmentada} = 50/10 = 5$ ns → $Speed\text{-}Up_{ideal} = 44 / 5 = 8.8X$
- con overhead: $T_{segmentada} = (50/10)+1 = 6$ ns → $Speed\text{-}Up_{overhead} = 44 / 6 = 7.33X$

Etapas = 15:
- sin overhead: $T_{segmentada} = 50/15 = 3.33$ ns → $Speed\text{-}Up_{ideal} = 44 / 3.33 = 13.2X$
- con overhead: $T_{segmentada} = (50/15)+1 = 4.33$ ns → $Speed\text{-}Up_{overhead} = 44 / 4.33 = 10.15X$

Etapas = 20:

- sin overhead: $T_{segmentada} = 50 / 20 = 2.5$ ns $\rightarrow$ $Speed\text{-}Up_{ideal} = 44 / 2.5 = 17.6X$
- con overhead: $T_{segmentada} = (50/20) + 1 = 3.5$ ns $\rightarrow$ $Speed\text{-}Up_{overhead} = 44 / 3.5 = 12.57X$

Se representan ahora estos resultados en la siguiente gráfica. Para confeccionar la gráfica se han usado también los valores intermedios.



Conclusión: Se puede observar una desviación entre las curvas correspondiente al $Speed\text{-}Up_{ideal}$ y $Speed\text{-}Up_{overhead}$. El Speed-Up calculado sin overhead ($Speed\text{-}Up_{ideal}$), aumenta en forma lineal, mientras que el Speed-Up con overhead ($Speed\text{-}Up_{overhead}$) describe una curva. Esto indica que en la realidad el aumento de prestaciones cada vez va siendo menor al aumentar el número de etapas de segmentación.

Problema 1-3-5 (HP96, ex.3.5, pp.217). Un procesador se denomina "subsegmentado" si se pueden combinar varias etapas de segmentación sin cambiar apreciablemente el comportamiento respecto a las paradas del flujo de ejecución de las instrucciones. Suponer que en el procesador DLX23p segmentado de 5 etapas se han unificado las etapas EX y MEM para lo que el ciclo de reloj se ha alargado en un 50%. ¿Cuánto de más rápido sería el procesador DLX32p convencional respecto al procesador subsegmentado para código sobre datos enteros? Utilizar los datos para el programa gcc (el 4% de las instrucciones producen una parada del flujo debido a saltos condicionales, y un 5% producen una parada del flujo de instrucciones debido a LW). Suponer que los saltos condicionales se resuelven en la etapa ID.

Solución:

Para el procesador segmentado sin "subsegmentación" las instrucciones de salto condicional (4% de la sinstrucciones) producen 1 ciclo de penalización (se resuelve en la etapa ID), y las dependencias debidas a LW (5% de las instrucciones) también producen una detención de 1 ciclo de reloj. En la siguiente tabla se observa un ejemplo de combinación de instrucciones donde la penalización debida a la carga (LW) se produce en el ciclo 4; y la instrucción ADDI se para un ciclo de reloj en su etapa ID (representado por ID!). Por otro lado, la penalización debido al salto condicional se produce en el ciclo 6. Hasta el ciclo siguiente no se puede conocer el contador de programa de la siguiente instrucción. Por eso, en el ciclo 6 la instrucción que ha entrado en el cauce y que es la ADDI está parada (representado por "IF!").

Ciclos	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*
1. ADDI R2,R2,#4	IF	ID	EX	M	WB						
2. LW R1,0(R2)		IF	ID	EX	M	WB					
3. ADDI R1,R1,#1			IF	ID!	ID	EX	M	WB			
4. BNZ R4, loop				IF!	IF	ID	EX	M	WB		
1. ADDI R2,R2,#4						IF!	IF'	ID'	EX'	M'	WB'
	11 ciclos										

Por todo ello, CPI = 1 + 1 × 0,04 + 1 × 0,05 = 1.09 ciclos

Para el procesador "subsegmentado" ya no hay penalización para las instrucciones LW; sólo se producen penalizaciones por las instrucciones de salto. En la siguiente tabla se muestra el ejemplo de la tabla anterior, en la que la etapa unificada se representa como "EXM".

Ciclos	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*
1. ADDI R2,R2,#4	IF	ID	EXM	WB					
2. LW R1,0(R2)		IF	ID	EXM	WB				
3. ADDI R1,R1,#1			IF	ID	EXM	WB			
4. BNZ R4, loop				IF	ID	EXM	WB		
1. ADDI R2,R2,#4					IF!	IF'	ID'	EXM'	WB'
	9 ciclos								

Por todo ello, CPI = 1 + 1 × 0,04 = 1.04 ciclos

Speed-Up = $t_{cpu\text{-}subsegmentado} / t_{cpu\text{-}original} = N \times CPI_{subsegmentada} \times T_{subsegmentado} / N \times CPI_{original} \times T_{original}$

Denominamos: $T_{original} = T$.

Según nos dice el enunciado: $T_{subsegmentado} = 1.5\ T_{original} = 1.5\ T$.

Sustituyendo en la ecuación del Speed-Up:

Speed-Up = N x 1.04 x (T x 1.5) / N x 1.09 x T = 1.56 / 1.09 = 1.43X

Conclusión: El procesador DLX32p convencional es un factor 1.43X más rápido que el subsegmentado. Es decir, aunque se producen menos penalizaciones por dependencias de datos de las instrucciones de carga en el procesador subsegmentado, el aumento del periodo de reloj (T) no es compensado por la reducción de las penalizaciones.

Problema 1-3-6 (HP96, ex.3.9, pp.217). Suponer que las frecuencias de los distintos tipos de instrucciones de salto son las siguientes:

Saltos condicionales: 20% (en el 60% se produce salto tomado)
Saltos incondicionales y llamadas a rutinas: 5%

Estamos examinando aquí un procesador con 4 etapas de segmentación (IF, ID, EX, WB) donde los saltos incondicionales se resuelven al final de la segunda etapa (ID) y los saltos condicionales se resuelven al final de la tercera etapa (EX). Suponiendo que sólo la primera etapa de segmentación (IF) se puede realizar independientemente de si el salto se realiza o no e ignorando las otras etapas de la segmentación, ¿cuánto más rápida sería la máquina sin dependencia de control por saltos?

Solución:

Aclaraciones previas a la solución:

Según el enunciado del problema, el 20% de las instrucciones son saltos condicionales, de los cuales, el 60% produce salto tomado y el 40% produce salto no tomado. De saltos incondicionales y llamadas a rutinas tenemos un 5%.

El enunciado nos dice también que la primera etapa de segmentación (IF) se puede realizar independientemente de si el salto se realiza o no e ignorando las otras etapas de la segmentación. Esto nos indica que la instrucción entra siempre en la etapa de búsqueda.

Si la instrucción es un salto condicional se queda parada hasta que se resuelva en la tercera etapa. Si el salto es no tomado, la instrucción que está en IF evoluciona normalmente. En este caso la penalización es de 1 ciclo de reloj. Si el salto condicional es tomado, la penalización es de dos ciclos de reloj. Si el salto es incondicional, cuando se encuentre en la segunda etapa (ID), una nueva instrucción puede entrar en la etapa IF, pero hay que desecharla. En este caso, la penalización es de un ciclo de reloj.

Por lo tanto, las penalizaciones se resumen a continuación:
- Penalización saltos condicionales tomados: 2 ciclos (se resuelven en la 3º etapa)
- Penalización saltos condicionales no tomados: 1 ciclo (se resuelven en la 3º etapa mientras la siguiente instrucción está en la primera etapa)
- Penalización saltos incondicionales 1 ciclo (se resuelven en la 2º etapa)

El problema nos pide una relación del tiempo de ejecución entre las distintas maquinas. No se modifica el periodo de reloj, tampoco el número de instrucciones a ejecutar; y por tanto, sólo tenemos que calcular la relación entre CPIs.

En la máquina sin dependencias de control, el CPI es ideal por ser una maquina segmentada. Por tanto en cada ciclo de reloj entra y sale una instrucción: CPI = 1.

Desarrollo de la solución:

$CPI_{ideal} = 1$

Suponemos que no existen saltos retardados.

$CPI_{real} = CPI_{ideal}$ + penalización de los saltos incondicionales + penalización de los saltos condicionales en los que no se produce salto + penalización de los saltos condicionales en los que si se produce salto

Saltos incondicionales: corresponde al 5% y siempre introducen un ciclo de penalización; penalización = 0.05×1

Saltos condicionales en los que no se produce salto: corresponde al 40% del 20% de las instrucciones, y siempre introducen un ciclo de penalización; penalización = $0.2 \times 0.4 \times 1$

Saltos condicionales en los que si se produce salto: corresponde al 60% del 20% de las instrucciones, y siempre introducen un ciclo de penalización; penalización = $0.2 \times 0.6 \times 2$

$CPI_{real} = 1 + 1 \times 0.05 + 1 \times 0.2 \times 0.4 + 2 \times 0.2 \times 0.6 = 1.37$ ciclos

Speed-Up = $CPI_{real} / CPI_{ideal} = 1.37X$

Conclusión: el tiempo de ejecución del programa en una máquina con dependencias de control es un 37% = $100 \times (CPI_{real} - CPI_{ideal}) / CPI_{ideal}$ más largo que en la máquina que no experimenta ciclos de penalización por dichas dependencias.

PROBLEMA 1-3-7 (HP96, ex.3.10, pp.218; Examen Final 27-1-2006). Suponer que un procesador dispone de las siguientes etapas de segmentación:

Etapa	Función
1 (IF)	Búsqueda de instrucción
2 (ID)	Decodificación y Búsqueda de Operandos
3 (EX)	Ejecución, Acceso memoria, Resolución de saltos, Escritura de resultados

Todas las dependencias de datos se restringen a la interrelación entre el registro escrito en la etapa 3 (EX) de la instrucción *i* y la lectura de registro de la instrucción *i+1* en la etapa 2 (ID) antes que la instrucción *i* se haya completado. La probabilidad de que aparezca esta dependencia es de *1/p*, es decir p^{-1}. La probabilidad representa a la frecuencia de aparición de la dependencia entre instrucciones y puede considerarse de forma equivalente al porcentaje de instrucciones ejecutadas cuando se calcula el CPI; es decir, porcentaje de instrucciones que penalizan = p^{-1}.

Se está considerando un cambio en la organización del procesador que resolvería los saltos y escribiría en el banco de registros el resultado de la instrucción en una cuarta etapa de la segmentación (WB). Esto reduciría el tamaño del periodo de reloj en una cantidad *d*. Es decir, si el periodo original es *T*, el periodo que resultaría a consecuencia del cambio sería *T-d*. Esta modificación origina la aparición de un nuevo riesgo de penalización por dependencia de datos entre las instrucciones *i* e *i+2*. La probabilidad que aparezca una dependencia de datos entre la instrucción *i* y la instrucción *i+2* es p^{-2}. Suponer que:

- **El valor de p^{-1} incluye los casos de las dependencias tanto entre *i* e *i+1* como *i* e *i+2***
- **Las instrucciones de salto también se resolverían en esta cuarta etapa (WB)**
- **El banco de registros no permite la escritura y lectura de un mismo registro en el mismo ciclo de reloj**

a.) Suponer que <u>no se añade hardware adicional</u> para aplicar anticipación a la cuarta etapa. Considerar exclusivamente las dependencias de datos, y encontrar el <u>límite inferior</u> de *d* que ocasiona que el cambio a 4 etapas proporcione un menor tiempo de ejecución que en el caso en el que el procesador dispone de 3 etapas.

Solución:

<u>Aclaraciones previas a la solución:</u>

En la maquina con tres etapas, una instrucción escribe el resultado de una operación en la etapa 3. La dependencia de datos se produce cuando la siguiente instrucción está en la etapa 2 e intenta leer el resultado generado por la instrucción anterior.

El enunciado considera la opción de introducir una cuarta etapa que resolviera los saltos y escribiera los resultados en el banco de registros. Se reduciría por tanto el periodo de reloj, ya que reducimos la complejidad del hardware de la última etapa en la máquina original, lo cual conduce a una reducción del periodo de reloj.

Para que la nueva máquina de 4 etapas fuera rentable tendría que exhibir menor tiempo de ejecución que la maquina de 3 etapas, o lo que es lo mismo, el Speed-Up tiene que se mayor que 1. El enunciado no nos da el número de instrucciones, y tampoco nos dice que se modifican en número. Por tanto, podemos eliminar el número de instrucciones de la relación de tiempos que permite calcular el Speed-Up. El periodo de reloj y el CPI sí se modifican porque la ruta crítica del hardware ha variado y aparecen nuevas penalizaciones. Por tanto, el Speed-Up sólo relaciona CPIs y periodos de reloj (T). El CPI base

de ambas maquinas es 1, ya que son segmentadas. Lo que cambia son los ciclos de penalización por instrucción. La maquina original tiene un periodo de reloj de T, y el periodo de la nueva máquina es T-d.

Como indica el enunciado, el banco de registros no permite una escritura y lectura en el mismo ciclo de reloj sobre el mismo registro, y también se dice que no existe anticipación.

Desarrollo de la solución:

En la siguiente tabla se observan las posibles dependencias de datos, las frecuencias a las que se producen, y los ciclos de penalización en cada procesador, tanto para el procesador original de tres etapas como para el nuevo de cuatro etapas.

		Penalización (ciclos)	
Dependencias de datos entre instrucciones	Frecuencia	Procesador 3 etapas	Procesador 4 etapas
$i \rightarrow i+1$	p^{-1}	1	2
$i \rightarrow i+2$	p^{-2}	0 (Nota 1)	1
$i \rightarrow i+1$ & $i \rightarrow i+2$	p^{-1} (Nota 2)	1	2

Nota 1: no existe penalización por esta dependencia en el procesador de 3 etapas

Nota 2: una vez resuelta la dependencia $i \rightarrow i+1$, desaparece la penalización debida a la dependencia $i \rightarrow i+2$. Por lo tanto esta dependencia está incluida en el primer caso "$i \rightarrow i+1$" y no se considera como caso aparte.

Se describe ahora el contenido de esta tabla.

- Primera Fila: La DEPENDENCIA DE DATOS "$i \rightarrow i+1$" ocurre con una frecuencia p^{-1}
  - Procesador de 3 etapas: Una instrucción i escribe en la etapa 3 y otra instrucción i+1 lee en la etapa 2. Esto no puede ocurrir porque no se puede leer y escribir en el mismo ciclo de reloj en el banco de registros. Por tanto, debe haber un ciclo de parada en la instrucción i+1.
  - Procesador de 4 etapas: Tiene dos ciclos de penalización porque la instrucción i escribe el resultado en la cuarta etapa, mientras la instrucción siguiente i+1 lo intentará mientras se encuentra en la segunda etapa. Por tanto, i+1 deberá esperar dos ciclos de reloj si quiere leerlo correctamente.

- Segunda Fila: La DEPENDENCIA DE DATOS "$i \rightarrow i+2$" ocurre con una frecuencia p^{-2}
  - Procesador de 3 etapas: No tiene sentido porque la instrucción i ya habrá escrito el resultado en el banco de registros cuando la instrucción i+2 tiene que leerlo.
  - Procesador de 4 etapas: Hay un ciclo de penalización en esta maquina porque la instrucción i no puede escribir cuando está en la cuarta etapa a la vez que la instrucción i+2 quiere leer en el banco de registros.

- Tercera Fila: La DEPENDENCIA DE DATOS "$i \rightarrow i+1$" e "$i \rightarrow i+2$"
  - "$i \rightarrow i+2$" indica una dependencia de datos en la que una instrucción i+2 estaría en la segunda etapa intentado leer operandos que otra instrucción i que entró dos ciclos antes estaría escribiendo. Si resolvemos el riesgo por dependencia de datos entre las instrucciones i e i+1, también queda resuelta la de i e i+2. Por tanto, este tercer tipo de penalizaciones está incluido en el primer tipo (primera fila).

La mejora de prestaciones se cuantifica con la ecuación del tiempo: $t_{cpu} = N \times CPI \times T$. Tenemos que comparar el tiempo de CPU de los dos procesadores: segmentado de 3 etapas (t_{cpu-3}), y segmentado de 4 etapas (t_{cpu-4}). Analizaremos las condiciones en las que:

$t_{cpu-3} > t_{cpu-4} \Rightarrow$

$(CPI_{base-3} + CPI_{penalizacion-3}) \times T_3 > (CPI_{base-4} + CPI_{penalizacion-4}) \times T_4$,, $T_3=T$,, $T_4=T-d$
Los ciclos de penalización se toman de la tabla anterior.

$CPI_{penalizacion-3} = 1 \times p^{-1} + 0 \times p^{-2} = 1/p$

$CPI_{penalizacion-4} = 2 \times p^{-1} + 1 \times p^{-2} = (2 \times p + 1) / p^2$

$t_{cpu-3} > t_{cpu-4} \Rightarrow (1 + 1/p) \times T > [1 + ((2 \times p + 1)/p^2)] \times (T-d) \Rightarrow d > T/(p+1) \Rightarrow d_{min} = T/(p+1)$

Conclusión: *d* debe ser superior a un valor mínimo para compensar el aumento de CPI debido a las mayores penalizaciones que surgen por dependencias de datos cuando aumenta el número de etapas de segmentación del procesador.

b.) Suponer ahora que hemos utilizado anticipación para eliminar las penalizaciones extras que aparecen a raíz del cambio a cuatro etapas de segmentación y que son originadas por las dependencias de datos "i → i+2". Es decir, que para todas las dependencias de datos, la longitud efectiva de la ruta segmentada es de tres etapas. Este nuevo diseño no tiene por qué ser mejor debido al impacto de las dependencias de control que se originan en una ruta de 4 segmentos frente a una de 3 segmentos. Suponer que en ambos procesadores, la siguiente instrucción a un salto puede entrar y permanecer "sólo" en la primera etapa (IF) antes que se decida si el salto se produce o no. Queremos analizar el impacto de las dependencias de control por saltos antes de que segmentemos más la ruta de datos para saber si se va a conseguir mayor nivel de prestaciones. Encontrar un límite superior para el porcentaje de saltos en el programa en función de la relación entre *d* y el ciclo de reloj original (T), de tal forma que la modificación de la ruta de datos proporciona mejor nivel de prestaciones. En el caso de que *d* represente un 10% de reducción del periodo de reloj, ¿cuál es el valor máximo del porcentaje de saltos condicionales antes de que un mayor número de segmentos produzca una disminución del nivel de prestaciones? Suponer que el porcentaje de saltos condicionales tomados es del 60%.

Solución:

Aclaraciones previas a la solución:

Al utilizar la anticipación en el procesador de cuatro etapas, van a desaparecer las penalizaciones por las dependencias i→i+2. Sin embargo, con la utilización de la anticipación no van a desaparecer todas las dependencias de datos, y además se incluyen ahora las dependencias de control. En estos casos, la probabilidad de que aparezca una dependencia de control coincide con la probabilidad de que una instrucción sea de salto respecto al total de instrucciones.

Desarrollo de la solución:

Las penalizaciones que aparecen por dependencias de control son las siguientes:

- Procesador de 3 etapas: Los saltos tomados se resuelven en la etapa 3, y por ello existen 2 ciclos de penalización. Los saltos no tomados se resuelven en la tercera etapa. En la primera etapa ya ha entrado una instrucción y por ello, tenemos un ciclo de penalización.

- Procesador de 4 etapas: Se resuelve una etapa mas tarde, y por ello, las penalizaciones de los saltos tomados y no tomados son de 3 y 2 ciclos respectivamente.

Las penalizaciones por dependencias de control se resumen en la siguiente tabla.

		Ciclos de Penalización	
Tipo de salto condicional	Frecuencia por instrucción	Procesador 3 etapas	Procesador 4 etapas
Tomado	60% × b	2	3
No tomado	40% × b	1	2

b: porcentaje de saltos condicionales respecto al total de instrucciones

Las penalizaciones por dependencias de datos se muestran en la siguiente tabla.

		Penalización (ciclos)	
Dependencias de datos entre instrucciones	Frecuencia	Procesador 3 etapas	Procesador 4 etapas
$i \rightarrow i+1$	p^{-1}	1	1 (reducida por anticipación)
$i \rightarrow i+2$	p^{-2}	0	0 (reducida por anticipación)

Se describe ahora el contenido de esta tabla.

- DEPENDENCIAS DE DATOS $i \rightarrow i+1$
  - Procesador 3 etapas: No tenemos anticipación y la penalización es la misma que en el apartado anterior.
  - Procesador 4 etapas: Gracias a la anticipación la penalización se reduce a 1 ciclo.

- DEPENDENCIA DE DATOS $i \rightarrow i+2$
  - Procesador de 3 etapas: No existen penalizaciones.
  - Procesador de 4 etapas: Gracias a la anticipación en el procesador de 4 etapas vamos a reducir a 0 ciclos la penalización.

Calculamos ahora los CPI de cada procesador:

$CPI_{penalizacion\text{-}3} = 1 \times p^{-1} + 0 \times p^{-2} + 2 \times 0.6 \times b + 1 \times 0.4 \times b = (1 + 1.6 \times b \times p) / p$

$CPI_{penalizacion\text{-}4} = 1 \times p^{-1} + 0 \times p^{-2} + 3 \times 0.6 \times b + 2 \times 0.4 \times b = (1 + 2.6 \times b \times p) / p$

Aplicamos ahora la condición de aumento de prestaciones para el procesador de cuatro etapas :

$t_{cpu\text{-}3} > t_{cpu\text{-}4} \Rightarrow (1 + (1 + 1.6 \times b \times p / p)) \times T > (1 + (1 + 2.6 \times b \times p / p)) \times (T\text{-}d) \Rightarrow$

$b < (d \times p + d) / (T \times p - 2.6 \times d \times p)$

Si $d = 0.1 \times T \quad \Rightarrow \quad b < 0.135 \times (p+1) / p$

Conclusión: "$b = 0.135 \times (p+1) / p$" es el porcentaje de saltos máximo que debe disponer un programa para que un procesador segmentado de cuatro etapas sea de mejores prestaciones que uno de 3 etapas cuando la reducción del periodo de reloj requerida es del 10%.

PROBLEMA 1-3-8 (fa98-preq_soln.pdf, Question 2). La siguiente secuencia de código corresponde al ensamblador DLX. Suponer que se va a ejecutar en un procesador con 5 etapas de segmentación y saltos retardados de 1 ciclo.

```
0:          add  r6,r0,r0
4:          lw   r1,16(r19)
8:          lw   r2,32(r19)
12:  loop:  slli r4,r2,#3
16:         addu r5,r4,r1
20:         lw   r4,0(r5)
24:         add  r6,r6,r4
28:         bnez r2,loop
32:         addi r2,r2,-1
36:         j    exit
40:         add  r10,r0,r0
44:  exit:  addi r11,r10,#12
```

Identifica todas las dependencias de datos que aparecen en este programa, y clasifícalas en uno de los siguientes tipos: RAW, WAR, WAW. Observar que se está pidiendo que identifiques todas las dependencias del programa y no sólo las dependencias de la segmentación.

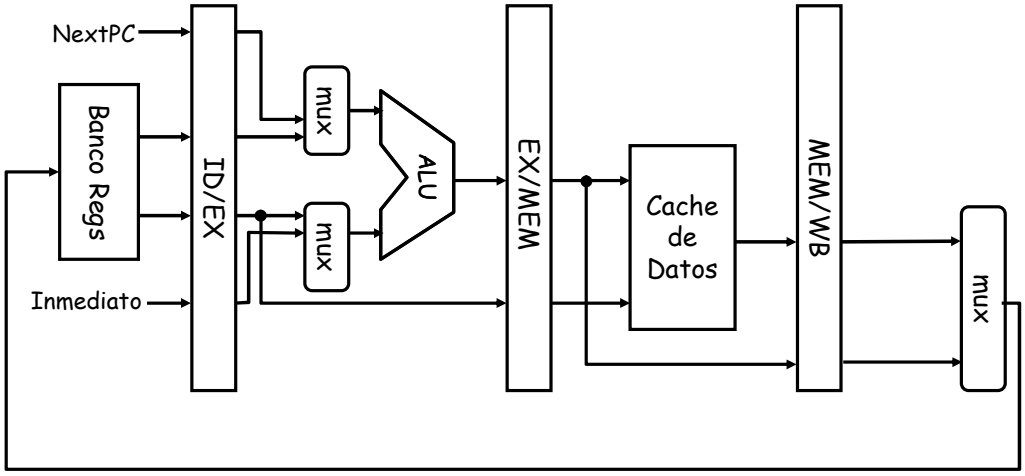
Para cada dependencia, determinar si es necesario aplicar la técnica de anticipación (la del procesador DLX segmentado en 5 etapas), o si se requiere parar el flujo de ejecución de la segmentación aunque se utilice anticipación. Suponer que un registro puede ser actualizado y leído en el mismo ciclo de reloj sin necesidad de anticipación, y por lo tanto no produce parada del flujo de instrucciones.

Para resolver este problema, rellena la tabla que aparece a continuación. "Instrucción Fuente" se refiere a la que escribe en el registro, "Instrucción Destino" a la que lee el registro. Para cada una de ellas, escribir el número de posición de memoria que aparece a la izquierda. "Registro" representa al registro involucrado en la dependencia.

Solución:

Instr. Fuente	Inst. Destino	Tipo Depend	Registro	Anticipación	Parada
32	32	WAR	R2	NO	NO
40	44	RAW	R10	SI	NO
0	24	RAW	R6	NO	NO
4	16	RAW	R1	NO	NO
8	12	RAW	R2	SI	SI
8	28	RAW	R2	NO	NO
8	32	RAW	R2	NO	NO
12	16	RAW	R4	SI	NO
16	20	RAW	R5	SI	NO
20	16	WAR	R4	NO	NO
20	24	RAW	R4	SI	SI
24	24	WAR	R6	NO	NO
32	12	WAR	R2	NO	NO
32	28	WAR	R2	NO	NO
40	44	RAW	R10	SI	NO
8	32	WAW	R2	NO	NO
12	20	WAW	R4	NO	NO
0	24	WAW	R6	NO	NO

PROBLEMA 1-3-9 (preq-soln.pdf, 2). **Este problema analiza las dependencias de una ruta de datos segmentada.**
a) Existen 3 tipos diferentes de dependencias de datos: RAW, WAR, y WAW. Definirlas, proporcionando una corta secuencia de código que ilustre cada una de ellas. Indicar brevemente para cada tipo de dependencia, cuál es la técnica que permite eliminarla.



Solución:

Dependencia RAW: Es una dependencia en la que existe una instrucción previa que genera un resultado y una instrucción a posteriori que utiliza dicho resultado. Para eliminarla podemos utilizar anticipación para instrucciones aritméticas o una burbuja para las instrucciones de carga. Ejemplo:

add **r1**, r2, r3
sub r4, **r1**,r2

Dependencia WAR: Es una anti-independencia en la que hay una instrucción más vieja que lee un registro y una más nueva que intenta escribir en ese registro. El riesgo surge cuando la nueva instrucción escribe antes de que la más vieja lea. Para eliminarla, en el procesador segmentado tenemos que leer lo más pronto posible y escribir lo más tarde posible. Ejemplo:

add r1, **r2**, r3
sub **r2**, r4, r2

Dependencia WAW: Se produce cuando dos instrucciones consecutivas escriben en el mismo registro. El problema surge cuando la instrucción nueva escribe en el registro antes que la más vieja. El problema se resuelve haciendo que las instrucciones se ejecuten en el orden que establece el programa. Ejemplo:

add **r1**, r2, r3
sub **r1**, r2, r3

b) ¿Qué son las dependencias de control? Mencionar dos de las técnicas que se utilizan para resolverlas.

Solución:

Las dependencias de control son las situaciones que aparecen en la ejecución de los programas en las que ciertos tipos de instrucciones ocasionan un salto no secuencial del contador de programa. El problema surge de que se tarda un tiempo en averiguar qué instrucción se ejecuta después del salto. Hay

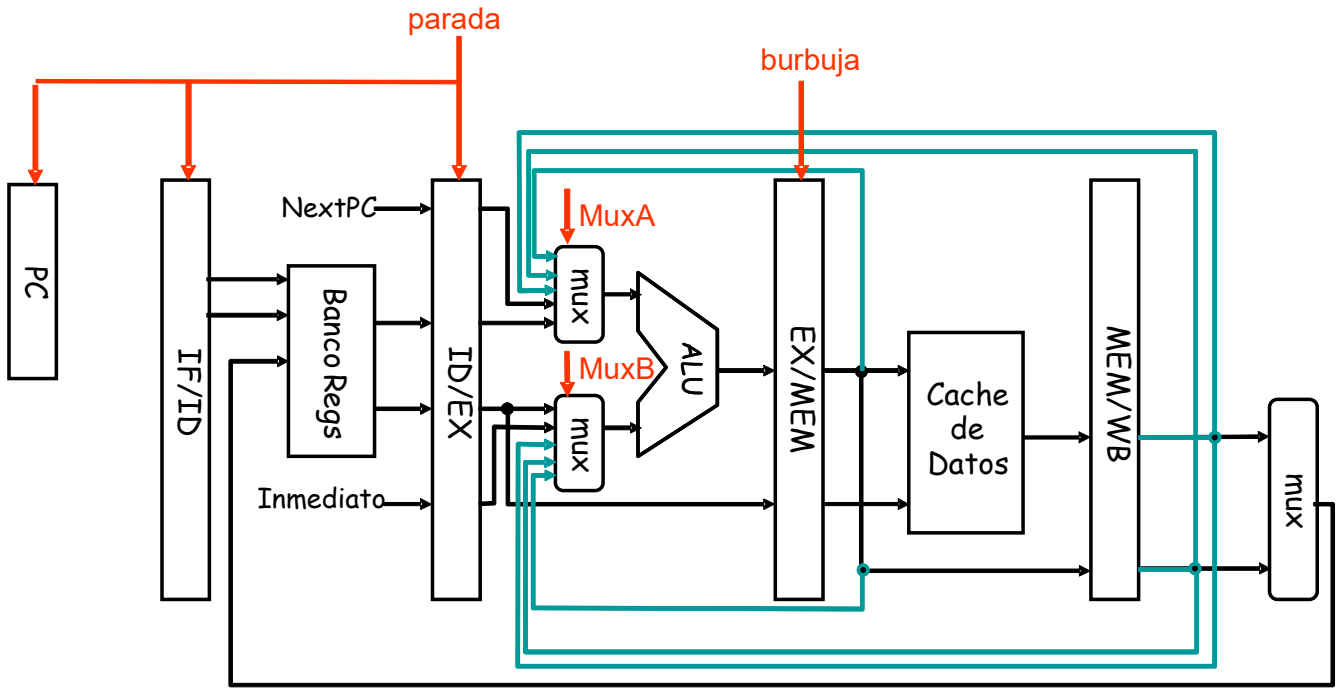
instrucciones que se ejecutan después del salto que no se saben cuáles son, lo cual ocasiona penalizaciones. Para resolver este tipo de dependencia se utilizan las siguientes técnicas:

- Parada del flujo de instrucciones mientras la instrucción de salto decide a dónde saltar.
- Predecir que si una instrucción de salto va a saltar o no.
- Ejecutar las dos ramas por donde puede dirigirse la ejecución del programa cuando se ejecuta un salto.
- La utilización de saltos retardados, lo cual requiere que se soporte en la arquitectura del repertorio de instrucciones. Este tipo de saltos implica que una o varias instrucciones que se encuentran en la memoria de programa después del salto siempre se ejecutan.

c) Dibuja y describe modificaciones simples a la ruta de datos de la figura anterior que se necesitan para eliminar algunas de las penalizaciones por dependencias de datos. Incluir las señales de control necesarias.

Solución:

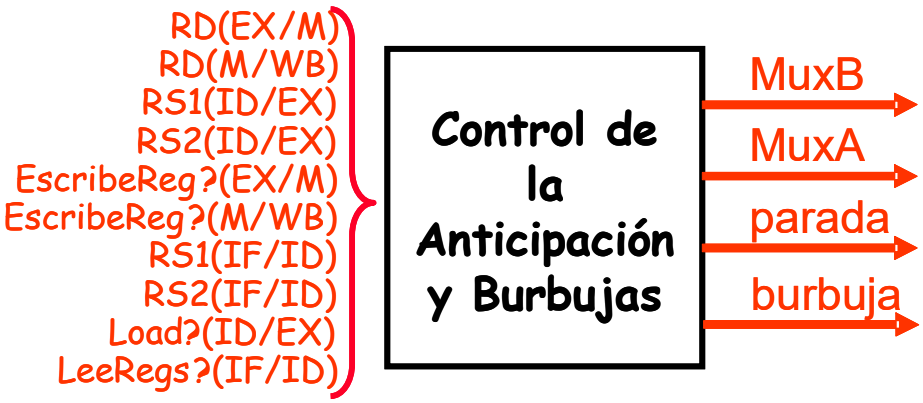
A la ALU le ampliamos los multiplexores, que es por donde entran los datos provenientes de la etapa de decodificación para realizar la operación pertinente. Con ello, podremos realizar anticipaciones de datos provenientes bien de la salida de la ALU o desde la etapa de WB hacia la entrada de la ALU. Esto se traduce en seis nuevos caminos de anticipación, como se puede observar en la siguiente figura con las líneas en azul. Adicionalmente, para solucionar las situaciones en las que no se pueden anticipar los datos se añaden varias señales de control que en la siguiente figura aparecen en rojo. Las nuevas señales son las de *parada*, que no permite la evolución de las instrucciones que se encuentran en las etapas IF, e ID; y la señal *burbuja*, que no permite la evolución de la instrucción que se encuentra en la etapa EX, y genera una instrucción `nop` para que no quede replicada la instrucción que se encuentra en la etapa EX. Aparte de esto, las señales de control de los multiplexores (*MuxA* y *MuxB*) se amplían para poder seleccionar entre un mayor número de entradas que aparecen en cada puerto de la ALU.



d) Las modificaciones anteriores de la ruta de datos dispondrán de un bloque que las controle. Dibuja en un diagrama las señales que requeriría tal módulo de control así como sin son de entrada o salida.

Solución:

En la siguiente figura se puede observar las señales de entrada y salida del módulo de control solicitado. Las señales de salida son aquellas que controlan los multiplexores que nos permitirán direccionar el dato que va a necesitar la instrucción cuando está en la etapa de ejecución (EX): *MuxA*, *MuxB*. Adicionalmente, otras dos señales permiten o paralizan la evolución de algunas de las instrucciones que se encuentran en el cauce del procesador en las etapas IF, ID, EX: parada, burbuja.



Para generar las anteriores señales de salida debemos examinar aquellas instrucciones que escriban en el banco de registros y examinar la dirección de los registros destino que guarden un resultado. Para ello, miramos qué registros destino se encuentran en la etapa MEM o WB para saber si coinciden con registros fuentes de alguna instrucción más nueva en la etapa de ejecución (EX). Se utilizará un bit "*EscribeReg?*" en vez de los 6 bits del código de operación para saber si la instrucción escribirá en un registro y para activar la escritura cuando llegue al banco de registro.

Otro caso que nos podemos encontrar es una dependencia de datos entre una instrucción de carga y la siguiente instrucción. Para ello, debemos mirar aquellas instrucciones de carga que se encuentran en la etapa MEM, comprobar que la siguiente instrucción necesita operandos del banco de registros, y mirar la etapa EX de la siguiente instrucción. Hay que comprobar que coinciden los registros destinos de la instrucción de carga y los registros fuentes de la siguiente instrucción, y comprobar que esta última lee operandos (*LeeRegs?*=1).

Para detectar las dependencias de datos y tomar decisiones sobre las señales de salida, son necesarias las siguientes señales de entrada:

- RD: dirección del registro destino de una instrucción que se encuentre bien en la etapa MEM o en la etapa WB. Para ello, el valor de RD se toman de los registros de segmentación EX/M o M/WB respectivamente.
- RS1, RS2: direcciones de los registros fuentes 1 y 2 de una instrucción que se encuentre bien en la etapa ID o en la etapa EX. Para ello, el valor de RS1 y/o RS2 se toma de los registros de segmentación IF/ID o ID/EX respectivamente.
- EscribeReg?: indica si la instrucción que se encuentra en la etapa MEM o en WB tiene por resultado final la escritura en el banco de registros. Este valor se toma de los registros de segmentación EX/M o M/WB, dependiendo de la etapa donde se encuentre la instrucción.

- LeeRegs?: indica si la instrucción que se encuentra en la etapa ID necesita leer el banco de registros. Esta señal se toma del registro de segmentación IF/ID.
- Load?: Esta señal indica que la instrucción es una carga de un dato desde la memoria de datos, y se toma del registro de segmentación ID/EX.

e) Explica cómo la unidad de resolución de dependencias reconoce y resuelve las dependencias de datos para cada una de las siguientes secuencias. Se deben utilizar las señales de entrada y salida del apartado d). La expresión que debe aparecer en la parte de reconocimiento de la dependencia debe hacerse con expresiones booleanas. La expresión que debe aparecer en la parte de resolución debe proporcionar valores para las señales de salida del apartado d).

`lw r1,0(r2)`
`add r3,r1,r2`
`add r5,r4,r6`
	`add r1,r2,r3`
	`add r5,r6,r7`
	`sub r4,r1,r2`
		`add r1,r2,r3`
		`sub r4,r4,r1`
		`add r5,r5,r3`

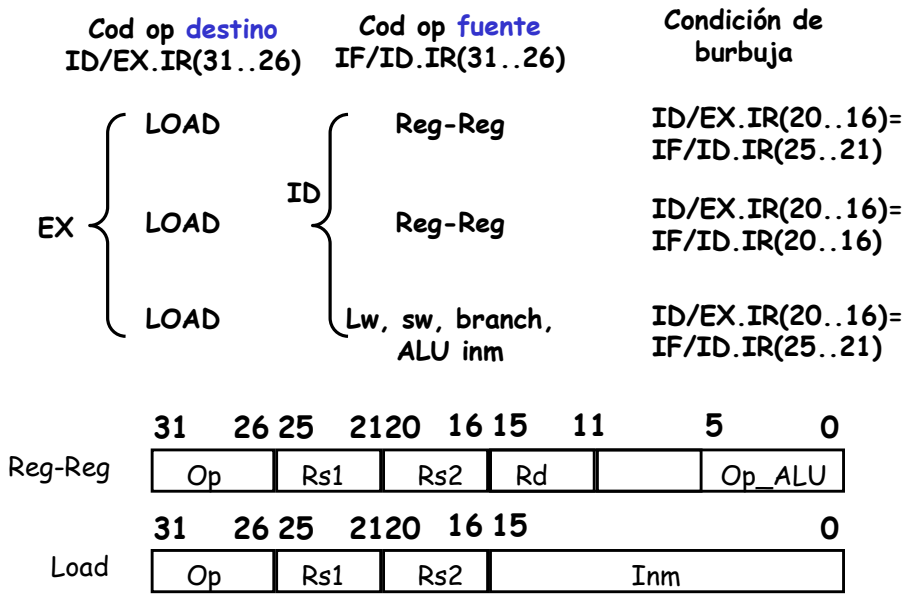
Solución:

Veamos en primer el primer caso en el que una instrucción de carga obliga a detener el flujo de algunas instrucciones que se introducen en el procesador posteriormente. En este caso una suma que necesita el registro r1 que la instrucción de carga utiliza para guardar el dato que trae desde la memoria de datos:

```
lw r1,0(r2)
add r3,r1,r2
add r5,r4,r6
```

Previamente establecemos la siguiente nomenclatura:
- IR: Registro de instrucción de 32 bits que forma parte de cada registro de segmentación. Por ejemplo ID/EX.IR corresponde al lugar donde se encuentra el código de instrucción en el registro de segmentación ID/EX.
- Load?(ID/EX) = 1 sii ID/EX.IR(31..26) == código de operación de la instrucción `lw`.
- LeeRegs?(IF/ID) = 1 sii IF/ID.IR(31..26) == código de operación de instrucciones: registro-registro, registro-inmediato, saltos condicional, carga, almacenamiento.
- RS2(ID/EX) = ID/EX.IR(20..16)
- RS1(IF/ID) = IF/ID.IR(25..21)
- RS2(IF/ID) = IF/ID.IR(20..16)

A continuación, establecemos las condiciones en las que el flujo de la instrucción siguiente a la carga y de las que vienen a continuación se para: En el ciclo siguiente a la detección de la dependencia "`lw(en EX)-add(en ID)`", la unidad de control activará la señal parada (parada=1) para que la instrucción `lw` que se encuentra en la etapa MEM evolucione a la siguiente etapa WB y las siguientes a ésta permanezcan en las etapas EX, ID e IF durante un ciclo de reloj. Adicionalmente, la señal burbuja se debe activar (burbuja=1) para que la instrucción `lw` no se quede en la etapa EX. Las condiciones para activar las señales parada y burbuja son las que aparecen en la siguiente figura.



Aparecen en la figura más casos del que realmente corresponde al ejemplo `lw-add-add`. Particularmente, el caso de este código ejemplo es el que aparece en primer lugar. En cada condición se analiza el código de operación de la instrucción que estando en la etapa EX escribe en el registro destino RD cuando llegue a la etapa WB (en este caso una instrucción de carga "LOAD"). Además se tiene que analizar el código de operación de la instrucción siguiente que se encuentra en la etapa ID. Esta instrucción puede ser cualquiera que inicialice "LeeRegs?(IF/ID) = 1".

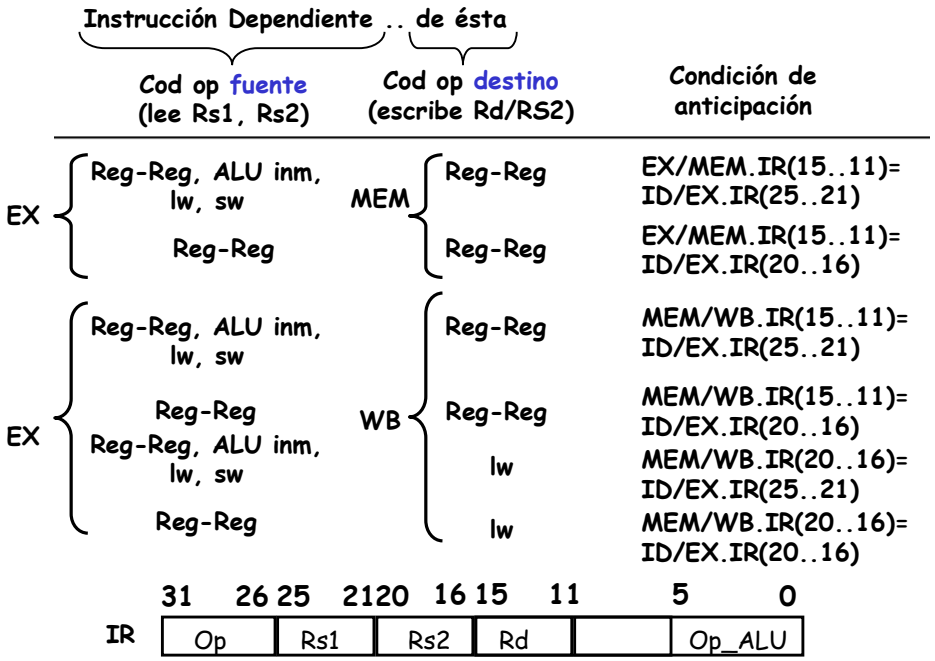
Veamos ahora los otros dos casos conjuntamente, en los cuales aparecen dependencias de datos entre una instrucción que genera un resultado y alguna de las dos siguientes que lo pretenden leer. En estas situaciones no existen paradas, sino la activación de caminos de anticipación hacia la ALU.

`add r1,r2,r3`
`add r5,r6,r7`
`sub r4,r1,r2`
	`add r1,r2,r3`
	`sub r4,r4,r1`
	`add r5,r5,r3`

Previamente establecemos la siguiente nomenclatura:

- EscribeReg(EX/M)? =1 sii EX/M.IR(31..26) == código de operación de instrucciones: registro-registro, registro-inmediato, carga; es decir, instrucciones que se encuentran en la etapa MEM y escriben en el banco de registros cuando se encuentren en la etapa WB.
- EscribeReg(M/WB)? =1 sii M/WB.IR(31..26) == código de operación de instrucciones: registro-registro, registro-inmediato, carga; es decir, instrucciones que se encuentran en la etapa WB y escriben en el banco de registros.
- RS1(ID/EX) = ID/EX.IR(25..21)
- RD(EX/M) = EX/M.IR(15..11)
- RD(M/WB) = M/WB.IR(15..11)

A continuación, establecemos las condiciones para activar la anticipación de datos hacia la entrada A de la ALU desde el registro M/WB en el primer código, y hacia la entrada B de la ALU desde el registro EX/M en el segundo código. En ambos casos, la unidad de control de la anticipación activará el valor de MuxA y MuxB correcto para encauzar el dato requerido. En la siguiente figura aparecen más casos de los correspondientes a los dos códigos ejemplo. Concretamente, los casos tercero y segundo corresponden a los ejemplos de esta parte del problema.

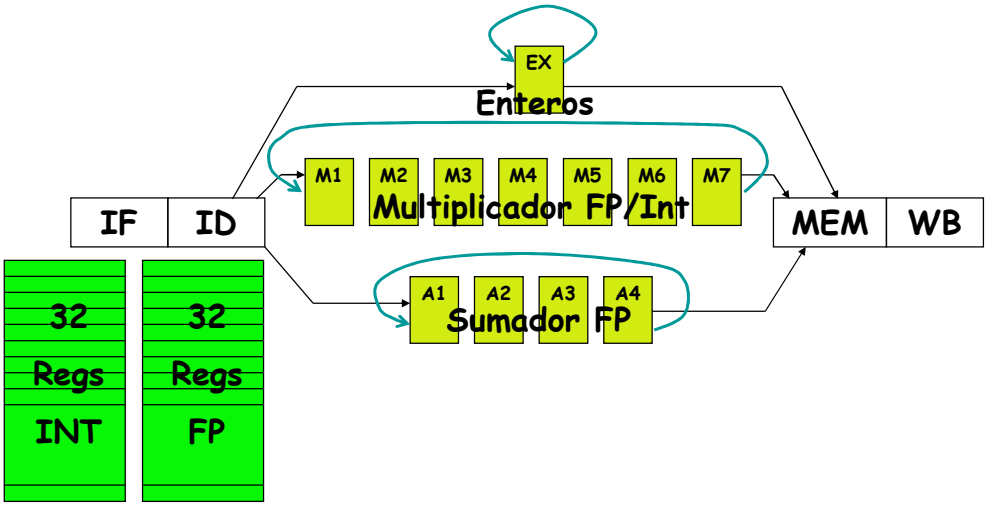


EJERCICIO ADICIONAL: Detectar en qué situaciones se deben activar a la vez las señales de control “parada” y “burbuja”.

PROBLEMA 1-3-10 (HP96, ex.3.11, pp.218). Construir una tabla con las condiciones de una ruta de datos FP segmentada para un procesador DLX32mf con múltiples unidades funcionales que originen las paradas del flujo de instrucciones por dependencias RAW. No considerar las divisiones, sólo las instrucciones Enteros/FP y FP/FP.

Solución:

Consideramos la microarquitectura del procesador DLX32mf que se observa a la derecha, en la que aparecen varias unidades funcionales de distinta latencia. Para que exista dependencia RAW, el registro que escribe la instrucción anterior, es el mismo que el que lee la nueva instrucción.



Utilizaremos la siguiente nomenclatura:

- ALU-FP: operaciones aritmético-logicas de punto flotante
- ALU-INT: operaciones aritmético-logicas de datos enteros
- ADDD: operación de suma/resta en punto flotante
- MULD: operaciones de multiplicación
- An: son los distintos segmentos del sumador, n= 1,2,3,4
- Mn: son los distintos segmentos del multiplicador, n= 1,2,…,7

En la siguiente tabla se muestran las **condiciones de parada** del flujo de instrucciones por RAW.

Combinación de Instrucciones	Instrucción que escribe resultado	Etapa de la instrucción que escribe resultado	Instrucción que lee el resultado	Etapa de la instrucción que lee el resultado
FP/FP	ADDD	A1	ALU-FP	ID
FP/FP	ADDD	A2	ALU-FP	ID
FP/FP	ADDD	A3	ALU-FP	ID
ENTEROS/FP	MULD	M1	ALU-FP, ALU-INT	ID
ENTEROS/FP	MULD	M2	ALU-FP, ALU-INT	ID
ENTEROS/FP	MULD	M3	ALU-FP, ALU-INT	ID
ENTEROS/FP	MULD	M4	ALU-FP, ALU-INT	ID
ENTEROS/FP	MULD	M5	ALU-FP, ALU-INT	ID
ENTEROS/FP	MULD	M6	ALU-FP, ALU-INT	ID
ENTEROS/FP	LD	EX	ALU-FP	ID
ENTEROS/FP	LW	EX	ALU-FP, ALU-INT	ID

PROBLEMA 1-3-14 (examen). Suponer que se dispone de un procesador cuyo propósito es que el flujo de instrucciones nunca se pare a no ser que los operandos no estén disponibles. Sus características son las siguientes:

- **Monoescalar (sólo se envía a la etapa EX una instrucción a la vez)**
- **Segmentación en orden con las siguientes etapas: 1 búsqueda (IF), 1 decodificación (ID), múltiples etapas de ejecución (EX1, EX2, ...), 1 postescritura (WB).**

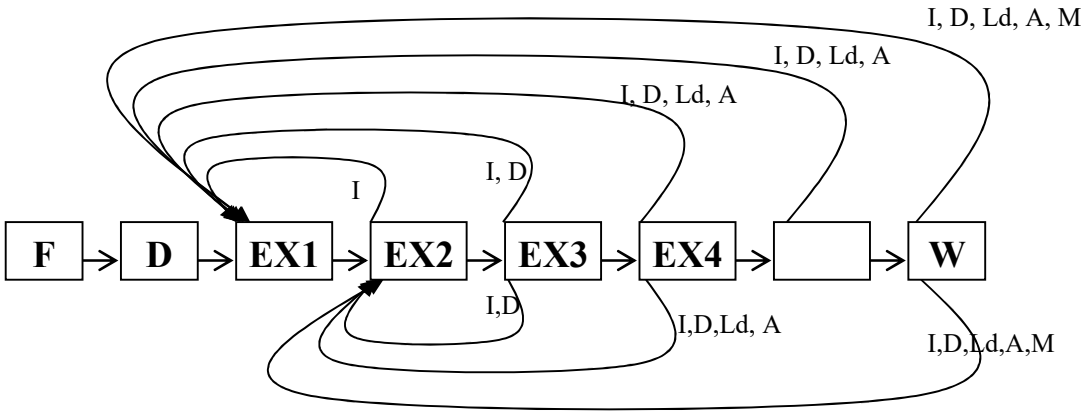
Suponer las siguientes latencias de la <u>única</u> etapa EX que se encuentra segmentada de forma tal que cada tipo de instrucción requiere los siguientes números de ciclos de reloj:

- **multf: 5 ciclos**
- **addf: 3 ciclos**
- **divf: 2 ciclos**
- **operaciones sobre enteros y saltos: 1 ciclo**
- **cargas/almacenamientos: 3 ciclos (incluido el cálculo de la dirección)**

Suponer que este procesador segmentado consiste en una secuencia lineal de etapas en las que las últimas etapas de EX equivalen a no-operaciones para las instrucciones que requieren un número inferior de ciclos en la etapa EX.

a) Dibujar los distintos segmentos y asignarles siglas distintas. Describir la función que se le asigna a cada segmento y mostrar a través de flechas los caminos de anticipación entre etapas. Etiqueta cada flecha con el tipo de instrucción que anticipará sus resultados a través de estas trayectorias. Por ejemplo, utilizar "Ld" para las cargas/almacenamientos, "M" para multf, "D" para divf, "A" para addf, e "I" para operaciones con enteros.

Solución:



Significados

Ld Carga (Load)
F Búsqueda próxima instrucción
D Decodificación
EX1 Operaciones sobre enteros, cálculo dirección para Carga/Almacenamiento, primera etapa de: Addf, Multf, Divf
EX2 Primera etapa de: Carga/Almacenamiento, última etapa de Divf, segunda etapa de: Addf, Multf
EX3 Última etapa de: Carga/Almacenamiento, Addf, tercera etapa de Multf
EX4 Cuarta etapa de Multf
EX5 Quinta etapa de Multf
W Writeback

b) ¿Cuántas instrucciones extra se requieren entre cada una de las siguientes combinaciones de instrucciones para evitar paradas (por ejemplo, suponer que una segunda instrucción utiliza un valor que proporciona la primera): divf/almacenamiento, carga/multf, 2 operaciones consecutivas sobre números enteros, multf/addf, addf/divf, enteros/almacenamiento?

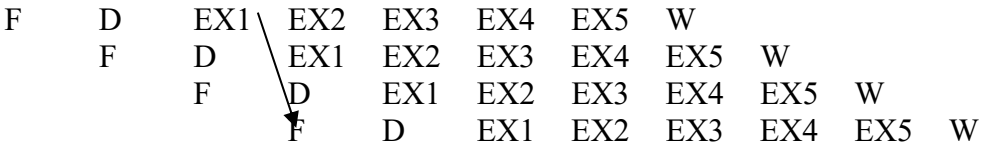
Solución:

Entre Divf y Store: 1
Entre Load y Multf: 2
Entre 2 operaciones sobre enteros: 0
Entre Multf y Addf: 4
Entre Addf y Divf: 2
Entre una operación sobre enteros y Store: 0

c) ¿Cuántas instrucciones se podrán ejecutar dentro de un salto retardado si la condición se establece comparando dos registros? Razonar la respuesta.

Solución:

Esta máquina tiene 2 ciclos de saltos retardados. Para llegar a esta conclusión, sólo hay que notar que el establecimiento de la condición de salto se realiza comparando 2 registros, es decir, realizando una resta. Este tipo de operación es de tipo enteros, por lo tanto, el resultado se obtiene al final de la etapa EX1. Esto ocasiona los 2 ciclos de retardo.



Por lo tanto pueden existir hasta 2 instrucciones entre la instrucción de salto y la primera instrucción destino del salto.

d) ¿Existen dependencias WAR y WAW?

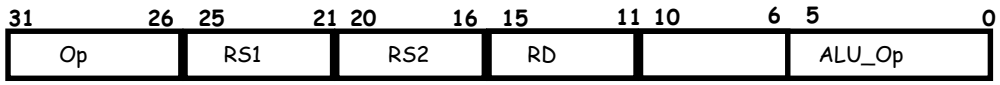
Solución:

No existen dependencias WAR y WAW ya que las instrucciones se lanzan a ejecutar en orden y se terminan de ejecutar en orden.

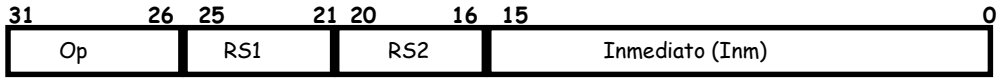
Pregunta 1-3-15 (examen). Describe las modificaciones de la ruta de datos segmentada del procesador DLX32p con unidad funcional de 1 ciclo para resolver la dependencia de datos siguiente utilizando anticipación:

```
ADD R1,R3,R4 ;
SW  R1,0(R2) ;
```

Formatos para las instrucciones REGISTRO-REGISTRO (R-R) y REGISTRO-INMEDIATO (R-I)



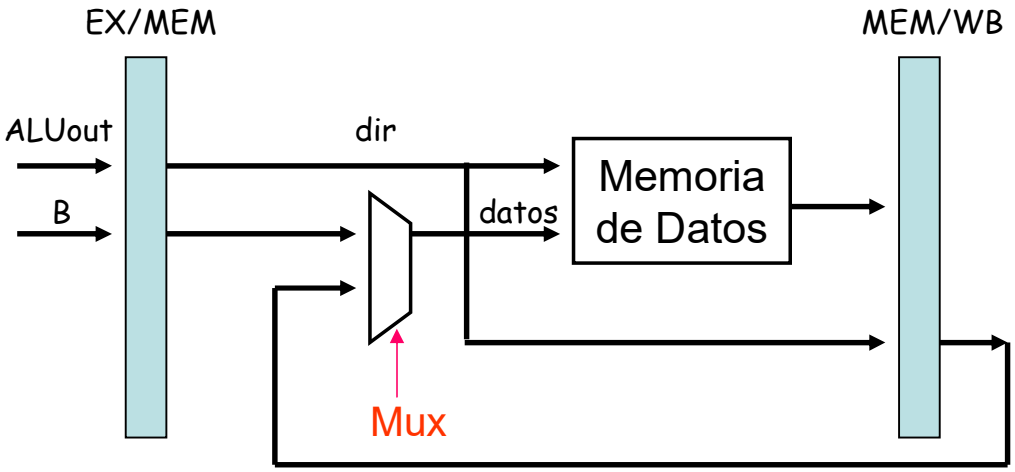
R-R



R-I

Solución:

Tan pronto como la ALU calcule la suma que se realiza en la primera instrucción se puede suministrar el resultado a la instrucción SW. Para esta solución, consideramos que la instrucción ADD llega a la etapa WB, mientras que SW se encuentra en la etapa MEM. La solución consiste en encaminar el resultado de ADD desde la etapa WB al puerto de datos de la memoria de datos en la etapa MEM. Para ello, se necesita añadir un multiplexor a la entrada de datos de la memoria de datos que se encuentra en la etapa MEM, así como control adecuado para manejarlo. En la siguiente figura se muestra un diagrama con la solución en la que se resalta la etapa MEM del procesador DLX32p.

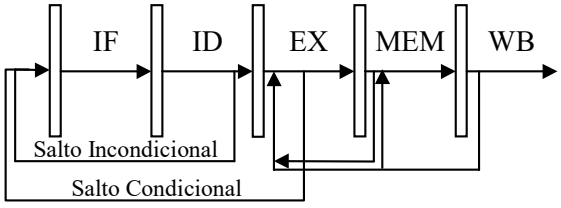


Es necesaria una unidad de control de esta anticipación que mostramos a continuación.

Las condiciones para detectar esta dependencia y activar la anticipación descrita son las siguientes:

- EX/MEM.IR[31..26]= código de operación (Cop) de "SW" en la etapa MEM.
- MEM/WB.IR[31..26]= código de operación (Cop) de: una instrucción Registro-Registro, o de una carga en la etapa WB.
- Se activa la anticipación sii EX/MEM.IR[20..16]=MEM/WB.IR[15..11] (EX/MEM.RS2=MEM/WB.RD)

Problema 1-3-16 (Examen Parcial 30/1/04). Procesador con Organización Segmentada Simple



Instrucción	Resol.	Semántica
BR label	ID	Salto incondicional a "label"
B*cc* Rx,label	EX	Salto condicional, si *cc*(Rx)
OP Rx,Ry,Rz	EX	(Rx OP Ry) → Rz
LDQ Rx,off(Ry)	MEM	Carga desde Memoria a Rx
STQ (Ry),Rx	MEM	Almacena Rx en Memoria

La figura superior muestra de forma simplificada la organización segmentada de 5 etapas de un procesador. Sólo se muestra la organización en etapas y los principales caminos de control y de datos entre las distintas etapas. Las instrucciones siempre comienzan a ejecutarse en orden y modifican el estado del procesador (registros y memoria) también en orden. Suponemos que no se producen fallos ni en la cache de datos ni en la de instrucciones. No existe predicción de saltos (es decir, se continúan obteniendo instrucciones de la cache de instrucciones suponiendo que los saltos no se toman).

El repertorio de instrucciones y la etapa en que cada tipo de instrucción se resuelve se muestran en la tabla que aparece a la derecha de la figura. Los saltos incondicionales se resuelven al final de la segunda etapa y los saltos condicionales al final de la tercera. Las operaciones aritméticas se resuelven al final de la tercera etapa. Las operaciones de acceso a memoria acaban al final de la cuarta etapa. En la quinta etapa se escribe el resultado de la operación en el registro destino (si es necesario). Las instrucciones no tienen que esperar a que sus operandos estén almacenados correctamente en el banco de registros, sino que pueden obtener el valor por caminos de anticipación justo al comenzar el ciclo siguiente a que se resuelva la instrucción anterior.

Sobre el procesador anterior se ejecuta el programa que se muestra a continuación.

```
1  LDQ     R5,0(R7)       ; R5 ← MEM(R7)
2  LDQ     R6,8(R7)       ; R6 ← MEM(R7+8)
3  LDQ     R1,-8(R6)      ; R1 ← MEM(R6-8)
4  BR      11             ; salta a instrucción 11
5  LDQ     R2,0(R6)       ; R2 ← MEM(R6)
6  SUBQ    R2,R1,R3       ; R3 ← R2-R1
7  BGE     R3, 9          ; if R3>=0 salta a 9
8  MOVQ    R1,R2          ; R1 ← R2
9  ADDQI   R6,8,R6        ; R6 ← R6+8
10 SUBQI   R5,1,R5        ; R5 ← R5-1
11 BNE     R5, 5          ; if R5!=0 salta a 5
12 LDA     R7,100(R7)     ; R7 ← 100+R7 (NO ACCEDE A MEMORIA)
13 STQ     R1,(R7)        ; MEM(R7) ← R1
```

(a) La secuencia de instrucciones ejecutadas es: 1,2,3,4,11,5,6,7,9,10,11,5,6,7,8,9,10,11,12,13. Realizar el grafo de dependencias de datos. Como existen instrucciones que se ejecutan más de una vez, distinguiremos cada una de ellas usando el subíndice a, b, c, ... Completar las siguientes tablas para la ejecución del programa (hasta que la última instrucción haya completado la última etapa). Son dos tablas equivalentes y, por lo tanto, hacer una de ellas inmediatamente determina la otra.

Solución:

Ciclo	*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*
1.R5←M(R7)	IF	ID	EX	**M**	WB																						
2.R6←M(R7+8)		IF	ID	EX	**M**	WB																					
3.R1←M(R6-8)			IF	ID!	ID	EX	**M**	WB																			
4.goto 11				IF!	IF	**ID**																					
11.R5≠0⇒ 5							IF	ID	**EX**																		
5.R2←M(R6)										IF	ID	EX	**M**	WB													
6.R3←R2-R1											IF	ID!	ID	**EX**	M	WB											
7.R3≥0⇒ 9												IF!	IF	ID	**EX**												
9.R6←R6+8																IF	ID	**EX**	M	WB							
10.R5←R5-1																	IF	ID	**EX**	M	WB						
11.R5≠0⇒ 5																		IF	ID	**EX**							
5.R2←M(R6)	IF	ID	EX	**M**	WB																						
6.R3←R2-R1		IF	ID!	ID	**EX**	M	WB																				
7.R3≥0⇒ 9			IF!	IF	ID	**EX**																					
8.R1←R2					IF	ID	**EX**	M	WB																		
9.R6←R6+8						IF	ID	**EX**	M	WB																	
10.R5←R5-1							IF	ID	**EX**	M	WB																
11.R5≠0⇒ 5								IF	ID	**EX**																	
12.R7←R7+100									IF	ID	**EX**	M	WB														
13.M(R7)←R1										IF	ID	EX	**M**														

En la anterior **tabla de evolución temporal** podemos observar lo siguiente:

- En el eje de ordenadas podemos encontrar las instrucciones tal y como nos dice que ejecuta la traza. En el eje de abscisas tenemos los ciclos de reloj. Como no caben el número de ciclos podemos observar que al llegar al ciclo 21 saltamos y empezamos desde el primer ciclo, por lo tanto a los 20 ciclos de reloj iniciales hay que sumarles otros 13 ciclos. En la segunda *tabla de evolución temporal* podemos ver todas las dependencias entre las instrucciones. Las flechas nos indica los caminos de anticipación de datos que resuelven alguna dependencia de datos entre las instrucciones.
- Podemos ver que la cuarta instrucción es un salto incondicional que tal y como dice el enunciado se resuelve en la etapa ID. Por lo tanto, la siguiente instrucción no puede entrar en el cauce hasta que la instrucción de salto incondicional llegue a la etapa ID.
- La instrucción 11 es un salto condicional que no se resuelve hasta la etapa EX por lo que la siguiente instrucción debe esperar hasta el ciclo 10 para entrar en el cauce.
- En la instrucción 7 pasa lo mismo que en el caso anterior; es decir, como el salto condicional se resuelve en la etapa EX, la siguiente instrucción debe esperar hasta este momento para entrar en el cauce.

En la **tabla de recursos** que es la que se muestra a continuación, la cual nos muestra en que ciclo se encuentra una instrucción determinada en cada etapa, el eje de ordenadas representa a las etapas de segmentación, y el eje de abscisas a los ciclos de reloj. Si encontramos una exclamación (“!”) quiere decir que la instrucción se encuentra parada. Si encontramos una flecha quiere decir que nos encontramos con una anticipación de datos.

Ciclo	*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*	*32*	*33*
IF	1	2	3	4!	4	5	11	12	13	5	6	7!	7	8	9	9	10	11	12	13	5	6	7!	7	8	9	10	11	12	13			
ID		1	2	3!	3	4		11	12		5	6!	6	7	8		9	10	11	12		5	6!	6	7	8	9	10	11	12	13		
EXE			1	2		3			11			5		6	7			9	10	11			5		6	7	8	9	10	11	12	13		
MEM				1	2		3						5		6				9	10				5		6		8	9	10		12	13
WB					1	2		3						5		6				9	10				5		6		8	9	10		12

(b) Para este fragmento de programa y para este procesador, ¿cuál es el promedio de instrucciones ejecutadas por ciclo (IPC = instrucciones por ciclo = 1/CPI)? Identificar cada uno de los casos que provocan que el IPC sea menor que uno. Para cada caso diferente, indicar el número de ciclos de penalización que provoca y dar una explicación de por qué se produce la penalización.

Solución:

La ejecución del programa tarda 33 ciclos en terminar, y se ejecutan un total de 20 instrucciones, por lo tanto como: IPC= nº instrucciones / nº ciclos

Tenemos que: IPC= 20 instrucciones / 33 ciclos = 0.6060

A continuación, se describen los tipos de penalizaciones y sus causas:

Tipos de Penalización	Penalización	Nº de veces que aparece
RAW 2 $\Rightarrow$ 3	1 ciclo	1
Dependencia de Control Incondicional en la instrucción 4:	1 ciclo	1
Dependencia de Control Condicional en la instrucción 11:	2 ciclos	2
RAW 5 $\Rightarrow$ 6 (2 veces) :	1 ciclo	2
Dependencia de Control Condicional en la instrucción 7:	1 ciclo	2

La cantidad de ciclos de penalización es la siguiente: Total Penalización= 1+1+2×2+1×2+1×2 = 10 ciclos

Clasificando los orígenes de las penalizaciones: 3 ciclos se deben por dependencias de datos, y 7 ciclos por dependencias de control.

Tiempo Total= 20 ciclos (1 por instrucción) + 10 ciclos de penalización + 3 ciclos (para vaciar el cauce) = 33 ciclos

No se considera la etapa de WB porque en la instrucción de almacenamiento porque realmente no hace nada en esta etapa.

(c) Suponer que no existieran caminos de anticipación de datos y que los resultados de una instrucción no pudieran ser utilizados por una instrucción posterior hasta haberse escrito en el banco de registros. Supondremos que mientras una instrucción se encuentra en la etapa WB, escribiendo el resultado en el banco de registros, otra instrucción en la etapa ID puede leer este resultado del banco de registros. ¿Cuántos ciclos de penalización se añadirían en el programa del apartado (a)? Suponemos que las instrucciones de salto siguen tratándose como en el apartado (a). ¿Qué porcentaje de pérdida se produciría?

Solución:

Las penalizaciones por dependencia de datos RAW con instrucciones de carga (LD) aumentan 1 ciclo de reloj. Pueden aparecer nuevas penalizaciones donde no había: dependencias entre instrucciones que tienen menos de 2 instrucciones entre ellas.

A continuación, se describen los tipos de penalizaciones por dependencias de datos y sus causas:

Tipos de Penalización	Penalización	Nº de veces que aparece
RAW 2 ⇒ 3	2 ciclos (1+1)	
RAW 5 ⇒ 6	2 ciclos (1+1)	2
RAW 6 ⇒ 7	2 ciclos (0+2)	2
RAW 10 ⇒ 11	2 ciclos (0+2)	2
RAW 12 ⇒ 13	2 ciclos (0+2)	

La cantidad de ciclos de penalización es la siguiente: Total Penalización por Dependencias de datos = 16 ciclos (3 + 13)

Dependencias de Control: 1 + 4 + 2 = 7 ciclos

Total Penalización: 16 + 7 = 23 ciclos (en el apartado anterior sólo había 10 ciclos de penalización)

Total ciclos= 20 ciclos (1 por instrucción) + 23 ciclos de penalización + 3 ciclos (para vaciar el cauce) = 46 ciclos

El número de instrucciones por ciclo que se ha ejecutado es: IPC= 20 / 46 = 0.435

Por tanto, Speed-Up = 0.435 / 0.6060 = 0.717X ,, Mejora = -28% = 100 × (0.43-0.6)/0.43

Es decir, existe una pérdida de prestaciones del 28.26%; en el apartado (b), la nueva máquina es un 28% peor que la del apartado (a).

Agradecimientos

En el curso 2005-2006, el estudiante David García Ortega redactó la solución de algunos de los problemas que aparecen en este documento. En el curso 2006-2007, los estudiantes Adaya S. Lorenzo León, Silvia Tejera Correa, Luis M. Matos Barroso, y Leidia Martel Monagas participaron en la redacción de las soluciones de este bloque de problemas.