

SOLUCIONES

C1.- Responda breve y concisamente a las siguientes preguntas. Extender innecesariamente las respuestas o contestar a lo que no se pregunta se puntuará negativamente.

C1.1. ¿Hay algún tipo de caché que no necesite algoritmos de reemplazamiento (tipo LRU o FIFO)? En caso afirmativo, indicar cuál.

Sí, las cachés de correspondencia directa.

C1.2. En cierto sistema, la unidad TLB tarda 1 ciclo de reloj, las tablas de traducción de páginas tardan 3 ciclos, la caché tarda 1 ciclo y la memoria principal tarda 5 ciclos. Indicar el número de ciclos que se tarda en conseguir el dato para los siguientes supuestos:

1. Fallo en TLB, acierto en tablas de traducción y acierto en caché real.
2. Acierto en TLB y acierto en caché real.
3. Fallo en TLB, acierto en tablas de traducción y fallo en caché virtual.
4. Acierto en TLB y acierto en caché virtual.

1. $1 \text{ (TLB)} + 3 \text{ (TT)} + 1 \text{ (C)} = 5 \text{ ciclos}$

2. $1 \text{ (TLB)} + 1 \text{ (C)} = 2 \text{ ciclos}$

3. $1 \text{ (TLB)} + 3 \text{ (TT)} + 5 \text{ (MEM)} + 1 \text{ (C)} = 10 \text{ ciclos}$

4. 1 ciclo en la caché

C1.3. Sea un procesador segmentado en cinco etapas: captura, decodificación, ejecución, acceso a memoria de datos, y escritura en registros. Indicar si deben registrarse las siguientes señales de control y, en caso afirmativo, por cuántos registros pasan.

1. Habilitación de escritura en registros internos.
2. Control de la ALU.
3. Habilitación de lectura en memoria de datos.

1. Sí, pasan por 3 registros

2. Sí, pasa por 1 registro

3. Sí, pasan por 2 registros

C1.4. Resuma en una frase en qué consiste un bus de ciclo partido. Indique cuál es su principal ventaja y su principal inconveniente (sólo una ventaja y un inconveniente).

Un bus de ciclo partido es aquel en que se puede comenzar una transacción sin haber terminado la anterior. La principal ventaja es que se aumenta el ancho de banda y el principal inconveniente la complejidad de control.

C1.5 ¿Qué barreras a la anticipación de cargas se eliminan gracias a la carga especulativa? ¿Y gracias al avance de la carga?

La carga especulativa evita las barreras de los saltos, y el avance de la carga la de los almacenamientos (store).

C2.- Sea un sistema de memoria virtual segmento-paginada con una estructura similar a la del Pentium. El sistema tiene las siguientes características: direcciones de 8 bits para el segmento y de 32 bits para el desplazamiento. La memoria real es de 4 Gbytes y el tamaño de páginas único es de 1kbytes. La MMU del sistema dispone de tres elementos:

1.- Un TLB completamente asociativo con 16 entradas.

2.- En caso de fallo en el TLB, una tabla de correspondencia directa para los descriptores de segmento que siempre contiene el segmento buscado. Con este descriptor de segmento, sumando el desplazamiento, se obtiene una dirección lineal de 32 bits.

3.- Para la traducción de la dirección lineal en una dirección real, dispone de una tabla directorio de páginas para el primer nivel con una memoria de correspondencia directa de 64 entradas. El resto de los niveles se encuentran en la memoria principal donde para cada tabla de traducción se necesita una página del sistema.

Sabiendo que el tamaño de un descriptor de cualquier tipo y en cualquier estructura es de 4 bytes, se pide, justificando la respuesta, el tamaño de los tres elementos de la MMU, así como la memoria real necesaria para guardar todas las tablas de páginas.

a) Tamaño del TLB en la MMU:

SEGMENTO	DESPLAZAMIENTO	
	PAGINA	OFFSET
8 bits	22 bits	10 bits

El TLB CA compara al tiempo el segmento (8b) y la página virtual (22b) y extrae el marco de página real (22b), como cualquier descriptor es de 32 bits, el tamaño del TLB será:

16 entradas x (8+22) bits = 480 bits CAM + 16 comparadores de 30 bits.

16 entradas x 32 bits = 512 bits = 64 bytes SRAM

b) Tamaño de la tabla de descriptores de segmento en la MMU:

La tabla de descriptores tiene 2^8 entradas por lo tanto su tamaño es:

2^8 entradas x 4 bytes = 1024 = 2^{10} bytes SRAM.

c) Tamaño de la tabla directorio de primer nivel en la MMU:

La tabla directorio tiene 2^6 entradas por lo tanto su tamaño es:

2^6 entradas x 4 bytes = 256 = 2^8 bytes SRAM.

d) La memoria real necesaria para guardar todas las tablas de páginas:

Una página de 1 kbyte, contiene 256 descriptores, por lo tanto la dirección lineal de 32 bits se subdivide en:

N1	N2	N3	OFFSET
6	8	8	10

Hay 2^6 tablas del nivel N2 en memoria real que ocupan $2^8 \times 2^2$ bytes $\Rightarrow 2^{16}$ bytes = 64 kbytes.

Hay $2^6 \times 2^8$ tablas del nivel N3 en memoria real que ocupan $2^8 \times 2^2$ bytes $\Rightarrow 2^{24}$ bytes = 16 Mb = 16.384 kb

En total tenemos 16.448 kbytes en la memoria real.

C3.- Se tiene un procesador segmentado con ejecución en orden (emisión y finalización en orden). La segmentación se produce en 5 etapas: (1) captura de instrucción, (2) decodificación y lectura de operandos de registros, (3) ejecución, (4) acceso a memoria de datos, (5) escritura en registros. Dicho procesador no dispone de adelantamiento de datos. Para cierto programa, el porcentaje de uso de cada tipo de instrucción es el reflejado en la siguiente tabla:

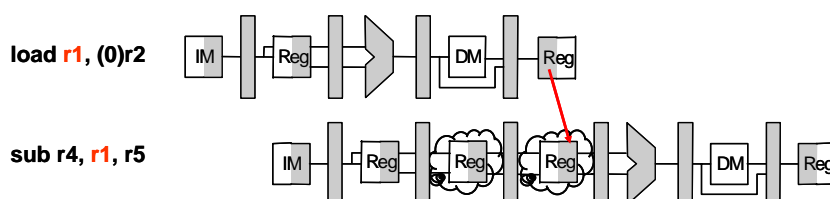
Tipo instr.	ENTEROS	SALTOS	LOAD	STORE
% uso	70%	5%	15%	10%

Con dicho programa y procesador se ha medido un CPI promedio de 3. Se pretende añadir adelantamiento de datos para reducir el CPI.

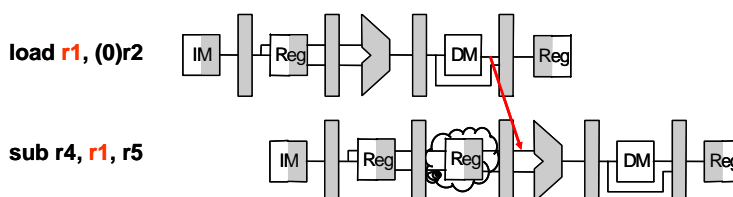
El adelantamiento de datos se aplica sólo a los datos obtenidos con cargas (LOAD). Suponga que en todos los casos de adelantamiento de datos se tiene éxito en caché (no hay demoras en memoria de datos) y que hay riesgos RAW con instrucciones en la siguiente posición de memoria el 20% de las veces y con instrucciones dos posiciones de memoria después el 15% de las veces. Por simplicidad, se puede suponer que ambas circunstancias nunca suceden a la vez. Calcular el nuevo CPI.

SOLUCIÓN:

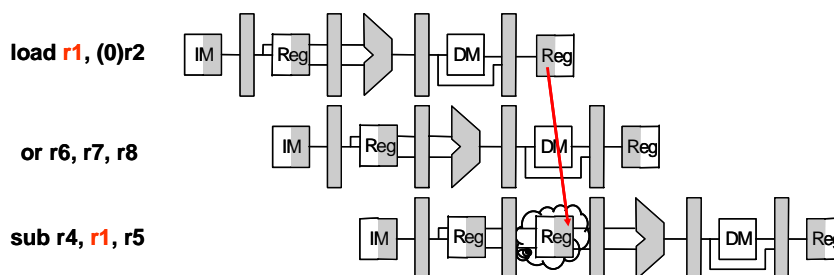
a) Si el riesgo se produce con la instrucción 1 posición después, sin adelantamiento de datos se pierden 2 ciclos como se observa en la figura.



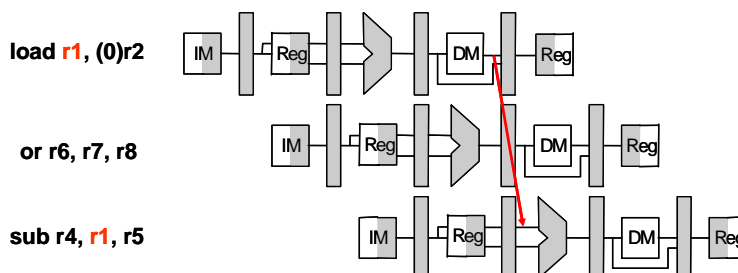
Sin embargo, con adelantamiento de datos se pierde sólo 1 ciclo (ver la siguiente figura), así que se gana 1 ciclo en estos casos.



Para riesgos con instrucciones 2 posiciones después, sin adelantamiento se pierde 1 ciclo (ver la siguiente figura).



Con adelantamiento no pierde ningún ciclo (ver la siguiente figura), así que se gana 1 ciclo.



La mejora en CPI es $0,15 \cdot 0,2 \cdot 1 + 0,15 \cdot 0,15 \cdot 1 = 0,03 + 0,0225 = 0,0525$

Por tanto, el nuevo CPI es $3 - 0,0525 = 2,9475$

C4.- Se dispone de un procesador RISC, con un CPI ideal de 1,6 ciclos (si se tiene en cuenta los fallos de caché y TLB), segmentado en las siguientes 10 etapas:

F1: Envío de dirección a la I-caché

F2: Recepción de la instrucción desde la I-caché

D1: Decodificación

RE: Lectura de operandos desde registros internos. Cálculo de dirección destino en saltos

A1: Comienza la operación en la ALU. Calcula la dirección efectiva para acceso a memoria.

A2: Ciclo intermedio de ejecución en operación con la ALU.

A3: Termina la operación con la ALU. Resuelve la condición.

M1: Envía la dirección a la D-caché

M2: Lee/Escribe el dato de/en la D-caché

WR: Escritura de resultados en registro

El procesador posee una instrucción de salto incondicional **JUMP** y cuatro instrucciones de saltos condicionales **JUMPZ**, **JUMPNZ**, **JUMPC**, **JUMPNC** que saltan en función de los flags de Zero y Carry respectivamente. De forma genérica llamaremos **JUMPX** a los saltos condicionales. Los flags son modificados por todas las instrucciones aritméticas y lógicas.

Considerar el siguiente programa en ensamblador que ha surgido de compilar el programa escrito en C que se muestra a continuación. El programa calcula cuántos números entre 1 y 1000 no son múltiplos de 7:

Código C

Ensamblador

	I1	load R1, 1	; R1 <= 1 (índice i)
	I2	xor R2, R2, R2	; R2 <= 0 (var nm)
for (i=1;i<1001; i++)	I3 L1:	Mod R3, R1, 7	; R3 <= R1 mod 7
{ if (i%7 != 0)	I4	jumpZ L2:	; salta si 0
nm ++;	I5	add R2, R2, 1	; R2 <= R2+1
};	I6 L2:	add R1, R1, 1	; R1 <= R1+1
	I7	sub R3, R1, #1000	; 1000 decimal
	I8	jumpNC L1	; salta por no carry
	I9	Stop	

Teniendo en cuenta únicamente los riesgos debidos a los saltos: se piden ciertos cálculos para el control de saltos en este programa usando las siguientes estrategias:

A) Sin predicción de saltos, el procesador se detiene a que se resuelva el salto.

B) Predicción estática: si el salto es hacia delante, se predice no efectivo y si es hacia detrás se predice efectivo.

C) Utilizando un predictor dinámico de saltos basado en 2 bits de control, es decir cambia la predicción cada dos errores de predicción consecutivos, tomando por defecto la opción NO efectiva cuando es hacia adelante y Efectivo cuando el salto hacia atrás (cada instrucción de salto tiene su propio predictor dinámico).

NOTAS:

- No considere los ciclos de llenado del pipeline
- Complete las respuestas i, ii, iii, iv en los espacios reservados a tal efecto
- Para justificar los ciclos de detención utilice las tablas adjuntas.

SOLUCION

El programa contiene 2 instrucciones de salto condicional (jumpZ y jumpNC) y ningún salto incondicional. La instrucción jumpZ (I4) se ejecuta 1000 veces de las cuales es efectivo en 142 y 858 veces es no efectivo. La instrucción jumpNC (I8) se ejecuta igualmente 1000 veces, con 999 saltos efectivos y un salto no efectivo.

i) Número total de instrucciones ejecutadas:

$$2 + 1000*2 + 858 + 1000*3 + 1 = 5861$$

(I1, I2 una vez; I3, I4 1000 veces; I5 858 veces; I6-I8 1000 veces; I9 una vez)

ii) Detenciones debido a la estrategia A (en ciclos)

$$\text{Saltos_efec} * \text{demEfectivo} + \text{Saltos_noef} * \text{demNOEfectivo}$$

$$(142+999) * 6 + (858+1) * 4 = 6846 + 3436 = 10282$$

iii) Detenciones debido a la estrategia B

$$\text{Saltos_Ad_Ef} * \text{Pred_NE_Ef} + \text{Saltos_Ad_NEf} * \text{Pred_NE_NE} + \text{Saltos_Atr_Ef} * \text{Pred_E_Ef} + \text{Saltos_Atr_NEf} * \text{Pred_E_NE}$$

$$858 * 0 + 142 * 6 + 999 * 3 + 1 * 3 = 3852$$

iv) Detenciones debido a la estrategia C

Para el primer salto (I4) siempre predecirá NO efectivo, ya que solo una de cada 7 veces será efectiva y no hará cambiar la predicción.

Para el segundo Salto (I8) acertará la predicción Efectiva 999 veces y solo fallará en la última.

El resultado numérico es igual al punto iii

TABLAS EXPLICATIVAS

Predicción Efectiva/salto NO Efectivo; Se Pierden 3 ciclos

	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12
jumpX	F1	F2	D1	RE ¹	A1	A2	A3 ²	M1	M2	WR		
N		F1	F2	D1	RE	A1	A2	A3	M1	M2	WR	
N+1			F1	F2	D1	RE	A1	A2	A3	M1	M2	WR
N+2				F1	F2	D1	RE	A1	A2	A3	M1	M2
N+3				-	-	-	-	F1	F2	D1	RE	A1
...
T					F1	F2	D1					-
T+1						F1	F2					-
T+2						-	F1	-	-	-	-	-

Predicción Efectiva/salto Efectivo; Se Pierden 3 ciclos

	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12
jumpX	F1	F2	D1	RE ¹	A1	A2	A3 ²	M1	M2	WR		
N		F1	F2	D1	RE	A1	A2	-	-	-	-	-
N+1			F1	F2	D1	RE	-	-	-	-	-	-
N+2				F1	F2	D1	-	-	-	-	-	-
N+3												
...
T					F1	F2	D1	RE	A1	A2	A3	M1
T+1						F1	F2	D1	RE	A1	A2	A3
T+2							F1	F2	D1	RE	A1	A2

Predicción NO Efectiva/salto NO efectivo; Se Pierden 0 ciclos

	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12
jumpX	F1	F2	D1	RE ¹	A1	A2	A3 ²	M1	M2	WR		
N		F1	F2	D1	RE	A1	A2	A3	M1	M2	WR	
N+1			F1	F2	D1	RE	A1	A2	A3	M1	M2	WR
N+2				F1	F2	D1	RE	A1	A2	A3	M1	M2
N+3					F1	F2	D1	RE	A1	A2	A3	M1
N+4						F1	F2	D1	RE	A1	A2	A3
N+5							F1	F2	D1	RE	A1	A2
...
T												
T+1												
T+2												

Predicción NO Efectiva/salto Efectivo; Se Pierden 6 ciclos:

	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12
jumpX	F1	F2	D1	RE ¹	A1	A2	A3 ²	M1	M2	WR		
N		F1	F2	D1	RE	A1	A2	A3	M1	M2	WR	

N+1			F1	F2	D1	RE	A1	-	-	-	-	-
N+2				F1	F2	D1	RE	-	-	-	-	-
N+3					F1	F2	D1	-	-	-	-	-
N+4						F1	F2	-	-	-	-	-
N+5							F1					
...
T								F1	F2	D1	RE	A1
T+1									F1	F2	D1	RE
T+2										F1	F2	D1

Sin Predicción y Detención / salto NO Efectivo; Se Pierden 4 ciclos

	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12
jumpX	F1	F2	D1 ⁰	RE ¹	A1	A2	A3 ²	M1	M2	WR		
N		F1	F2	-	-	-	-	D1	RE	A1	A2	A3
N+1			F1	-	-	-	-	F2	D1	RE	A1	A2
N+2								F1	F2	D1	RE	A1
N+3									F1	F2	D1	RE
...
T												
T+1												
T+2												

Sin Predicción y Detención / salto Efectivo; Se Pierden 6 ciclos

	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12
jumpX	F1	F2	D1 ⁰	RE ¹	A1	A2	A3 ²	M1	M2	WR		
N		F1	F2	-	-	-	-	-	-	-	-	-
N+1			F1	-	-	-	-	-	-	-	-	-
N+2				-	-	-	-	-	-	-	-	-
N+3												
...
T								F1	F2	D1	RE	A1
T+1									F1	F2	D1	RE
T+2										F1	F2	D1

0. Decodifica la Instrucción. Determina que se trata de un salto

1. Obtiene la dirección de salto. En el siguiente ciclo puede cargar esa dirección

2. Resuelve la condición. Descarga el pipeline si no acertó la predicción

Si bien este programa no posee saltos incondicionales se presentan los ciclos de demora correspondientes

Ciclos de demora en saltos incondicionales; Se Pierden 3 ciclos

	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12
JUMP	F1	F2	D1	RE1	A1	A2	A3	M1	M2	WR		
N		F1	F2	-	-	-	-	-	-	-	-	-
N+1			F1	-	-	-	-	-	-	-	-	-
N+2				-	-	-	-	-	-	-	-	-
N+3												
...
T					F1	F2	D1	RE	A1	A2	A3	M1
T+1						F1	F2	D1	RE	A1	A2	A3
T+2							F1	F2	D1	RE	A1	A2

C5.- Se tiene un diseño VHDL que consta de estos procesos:

P1: process(A,B) begin if Sel='0' then Q<=not(A); else Q<=B and A; end if; end process;	P2: process(Q) begin Qand<=Q and Q; end process;
---	--

La señales empleadas están definidas así en la arquitectura:

```

signal A,B  : std_logic_vector(11 downto 0);
signal Q,Qand : std_logic_vector(11 downto 0);
signal Sel  : std_logic;

```

Desde otro proceso se dan valores a las señales A, B y Sel en diferentes momentos:

P3: process begin A<=x"010"; B<=x"100"; Sel<='1'; wait for 10 ns; -- instante t0 A<=x"BB1"; B<=x"BB2"; -- instante t1 wait for 100 ns; -- instante t2 Sel<='0'; wait for 15 ns; -- instante t3 B<=x"111"; wait for 5 ns; -- instante t4 A<=x"000"; wait for 1 ns; -- instante t5 end process;	
---	--

Se pide:

a) Indicar los valores (en hexadecimal) de las señales en los diferentes instantes t0, t1, t2, t3, t4, y t5 que están indicados en los comentarios del código:

	t0	t1	t2	t3	t4	t5
A	x"010"	x"010"	x"BB1"	x"BB1"	x"BB1"	x"000"
B	x"100"	x"100"	x"BB2"	x"BB2"	x"111"	x"111"
Sel	1	1	1	0	0	0
Q	x"000"	x"000"	x"BB0"	x"BB0"	x"44E"	x"FFF"
Qand	x"000"	x"000"	x"BB0"	x"BB0"	x"44E"	x"FFF"

b) Justificar lo ocurrido en cada instante:

Instante	Descripción
t0	Se dan valores iniciales a las señales A,B y Sel. Se ejecuta el proceso P1 que da el valor x"000" a Q y luego se ejecuta P2 donde se asigna x"000" a Qand.
t1	Se ha planificado el valor futuro que las señales A y B tendrán cuando se suspenda el proceso, pero su valor real todavía no habrá cambiado. Todas las señales tendrán el mismo valor que en t0.
t2	Se suspende el proceso P3 por lo que A y B pasan a tomar los valores x"BB1" y x"BB2" respectivamente. Por haber cambiado de valor la señal A, se ejecuta P1 dando el valor x"BB0" a Q. Por cambiar el valor de Q, se ejecuta P2, tomando la señal Qand el valor x"BB0"
t3	La señal Sel no está en la lista de sensibilidad del proceso P1. Esto implica que el proceso P1 no se lanza a pesar de la suspensión del proceso P3. El proceso P2 tampoco se activa y las señales Q y Qand siguen valiendo lo mismo que en t2. La señal Sel si cambia de valor a 0 por el wait del proceso P3.
t4	La señal B cambia a x"111". Se ejecuta P1 y Q pasa a valor x"44E". Se ejecuta P2 y Qand toma el valor x"44E".
t5	La señal A cambia a x"000". Se ejecuta P1 y Q pasa a valor x"FFF". Se ejecuta P2 y Qand toma el valor x"FFF"

NOTA: Si la descripción de lo que ocurre en cada instante no es correcta o no se ha realizado, la correspondiente columna de la tabla del apartado a será considerada incorrecta.

P1.- Se dispone de un procesador VLIW con 5 unidades de ejecución. Una de load/store que necesita 4 ciclos de ejecución, una unidad de enteros (ALU) y saltos de 2 ciclos y tres unidades de punto flotante que requiere de 3 ciclos. Todas las unidades de ejecución son segmentadas (en cada ciclo de reloj puede empezar una nueva ejecución). Los ciclos se refieren a la latencia total del pipeline (segmentación). Dentro del pipeline no existe adelantamiento de datos.

Se utiliza este procesador VLIW para computar el programa que multiplica y suma elemento a elemento los 500 valores de dos vectores almacenados en memoria (arreglos A y B). Además calcula la suma del vector A y B por separado. Los elementos del arreglo (A[i], B[i], C[i] y D[i]) son en coma flotante de tamaño Word (4 bytes).

El código C y ensamblador (al margen de las inicializaciones) correspondiente es:

Código C

```
for (i=500; i>0; i--)
{ C[i] = A[i]*B[i];
  D[i] = A[i]+B[i];
  Sum1 += A[i];
  Sum2 += B[i];
};
```

Ensamblador

```
; suponer los reg ya inicializados
Loop: Load R2, 1000(R1) ; R2 <- mem(1000+R1)
      Load R4, 2000(R1) ; R4 <- mem(2000+R1)
      Mulf R5, R2, R4    ; R5 <- A[i]*B[i]
      Addf R6, R2, R4    ; R6 <- A[i]+B[i]
      Addf R7, R7, R2    ; R7 suma los A[i]
      Addf R8, R8, R4    ; R8 suma los B[i]
      Store 3000(R1), R5 ; mem(3000+R1) <- R5
      Store 4000(R1), R6 ; mem(4000+R1) <- R6
      Sub R1, R1, 4      ; R1 <- R1 - 4
      Bnez R1, loop     ; salto si R1 > 0
      Stop
```

a) ¿Cuántos ciclos necesita para ejecutarse este código si no se aplica ninguna optimización?. Utilice la tabla adjunta que representa las 5 unidades de ejecución:

Ciclo	LD/ST	ALU enteros y saltos	Coma Flot 1	Coma Flot 2	Coma Flot 3
1	Load R2, 1000(R1)	Nop	Nop	Nop	Nop
2	Load R4, 2000(R1)	Nop	Nop	Nop	Nop
3	Nop	Nop	Nop	Nop	Nop
4	Nop	Nop	Nop	Nop	Nop
5	Nop	Nop	Nop	Nop	Addf R7, R7, R2
6	Nop	Nop	Mulf R5, R2, R4	Addf R6, R2, R4	Addf R8, R8, R4
7	Nop	Nop	Nop	Nop	Nop
8	Nop	Nop	Nop	Nop	Nop
9	Store 3000(R1), R5	Sub R1, R1, 4	Nop	Nop	Nop
10	Store 4000(R1), R6	Nop	Nop	Nop	Nop
11	Nop	Bnez R1, loop	Nop	Nop	Nop
12	Nop	Nop	Nop	Nop	Nop
13	Stop	Stop	Stop	Stop	Stop
14					

Tiempo total de ejecución:

Tiempo: $500 \cdot 12 + 1 = 6001$ ciclos

b) Utilizando desenrollamiento de bucles, actualización de referencias y renombramiento de registros, se quiere mejorar el tiempo de ejecución. El desenrollamiento completo del programa ocuparía demasiada memoria. Se propone desenrollar, replicando el bucle central cinco veces. Suponga que no tiene límite en la cantidad de registros y nómbrelos de la forma Rxx (xx números). Utilice la tabla adjunta

Ciclo	LD/ST	ALU enteros y saltos	Coma Flot 1	Coma Flot 2	Coma Flot 3
1	Load R2, 1000(R1)	Nop	Nop	Nop	Nop
2	Load R4, 2000(R1)	Nop	Nop	Nop	Nop
3	Load R12, 00FC(R1)	Nop	Nop	Nop	Nop
4	Load R14, 10FC(R1)	Nop	Nop	Nop	Nop
5	Load R22, 00F8(R1)	Nop	Nop	Nop	Addf R7, R7, R2
6	Load R24, 10F8(R1)	Nop	Mulf R5, R2, R4	Addf R6, R2, R4	Addf R8, R8, R4
7	Load R32, 00F4(R1)	Nop	Nop	Nop	Nop
8	Load R34, 10F4(R1)	Nop	Mulf R15,R14,R12	Addf R16,R14,R12	Addf R7, R7, R12
9	Load R42, 00F0(R1)	Nop	Nop	Nop	Addf R8, R8, R14
10	Load R44, 10F0(R1)	Nop	Mulf R25,R24,R22	Addf R26,R24,R22	Nop
11	Store 3000(R1), R5	Nop	Nop	Nop	Addf R17,R22,R32
12	Store 4000(R1), R6	Nop	Mulf R35,R34,R32	Addf R36,R34,R32	Addf R18,R24,R34
13	Store 20FC(R1), R15	Nop	Nop	Nop	Addf R7,R7,R42
14	Store 30FC(R1), R16	Nop	Mulf R45,R44,R42	Addf R36,R34,R32	Addf R8,R8,R44
16	Store 20F8(R1), R25	Nop	Nop	Nop	Nop
17	Store 30F8(R1), R26	Nop	Nop	Nop	Addf R7, R7, R17
18	Store 20F4(R1), R35	Nop	Nop	Nop	Addf R8, R8, R18
19	Store 30F4(R1), R36	Nop	Nop	Nop	Nop
20	Store 20F0(R1), R45	Sub R1, R1, 14	Nop	Nop	Nop
21	Store 30F0(R1), R46	Nop	Nop	Nop	Nop
22	Nop	Bnez R1, loop	Nop	Nop	Nop
23	Nop	Nop	Nop	Nop	Nop
24	Stop	Stop	Stop	Stop	Stop
25					

Tiempo total de ejecución b:

Tiempo: $23 \cdot 100 + 1 = 2301$ ciclos

P2.- Sea un sistema caché con dos niveles donde las características de ambos niveles son:

Nivel L1 dual:

Caché de instrucciones, $H = 96\%$ y bloques de 8 palabras.

Caché de datos, $H = 94\%$ y bloques de 4 palabras. Estrategia de postescritura

Nivel L2 unificado: $H_{LOCAL} = 95\%$ y bloques de 8 palabras. Estrategia de postescritura.

Los tiempos de acceso son 10 nsec para ambas cachés del nivel L1, 50 nsec para la del nivel L2 y 150 nsec para la memoria.

Se sabe que la conexión entre la caché del nivel L1 y L2 es de 128 bits que el de la caché L2 con la memoria es de 256 bits, que el tamaño de palabra es de 32 bits y que la transferencia entre niveles siempre es por bloques.

Se sabe que el promedio de instrucciones que acceden a memoria de datos es del 20%, y que el porcentaje de bloques modificados es, en las caches correspondientes, del 45%.

Con los datos anteriores se pide el tiempo de acceso promedio por instrucción de este sistema de memoria.

SOLUCION:

$$1/t_{acc} = \%A_{men}^I \{t_{L1} + (1-H_{L1}^I) * t_{L2 \rightarrow L1}^I\} + \%A_{men}^D \{t_{L1} + (1-H_{L1}^D) * W_M * t_{D^{L1 \rightarrow L2}} + (1-H_{L1}^D) * t_{D^{L2 \rightarrow L1}}\}$$

Si siempre se transfieren bloques, en caso de fallo del nivel L2, el ancho del bus es suficiente para que en un único acceso a memoria se complete el bloque requerido

$$t_{B^{L2}} = 1 * t_{men} = 150 \text{ nsec}$$

Si siempre se transfieren bloques y el bus entre L1 y L2 es de 128 bits (4 palabras)...

...la penalización en caso de fallo en L1 de instrucciones, se necesita acceder dos veces al nivel L2

$$t_{I^{L2 \rightarrow L1}} = 2 * t_{L2} + (1+W_M) (1-H_{L2}) 1 * t_{men} = 2 * 50 + 1,45 * 0,05 * 150 = 110,875 \text{ nsec}$$

...la penalización en caso de fallo en L1 de datos....

...cuando se debe escribir en L2 por estar modificado el bloque a sustituir, nunca se falla en L2, ya que L1 esta incluida en L2.

$$t_{D^{L1 \rightarrow L2}} = 1 * t_{L2} = 1 * 50 = 50 \text{ nsec}$$

...cuando se transfiere un bloque de L2 a L1, con un único acceso es suficiente

$$t_{D^{L2 \rightarrow L1}} = 1 * t_{L2} + (1+W_M) (1-H_{L2}) 1 * t_{men} = 1 * 50 + 1,45 * 0,05 * 150 = 60,875 \text{ nsec}$$

...en caso de fallo en L1 de datos,

$$\begin{aligned} 1/t_{acc} &= 1 * \{10 + 0,04 * 110,875\} + 0,20 * \{10 + 0,06 * 0,45 * 50 + 0,06 * 60,875\} = \\ &= 1 * 14,44 + 0,2 * \{10 + 1,35 + 3,65\} = 17,44 \text{ nsec.} \end{aligned}$$