

## ARQUITECTURA E INGENIERIA DE COMPUTADORES. 1 - SEPTIEMBRE - 2005

**ATENCION:** Sólo se debe contestar a 4 de las 5 cuestiones. En el caso de contestar a las 5 sólo se considerarán las 4 primeras.

**C1.-** Se ha diseñado un detector de flanco de subida. El nombre de la entidad es RISE\_DETECT, sus entradas son D\_IN, CLK y RESET, y su única salida es R\_EDGE. El circuito detecta los flancos de subida en la señal D\_IN, muestreando esta señal en cada flanco de subida de CLK (evidentemente, la frecuencia de CLK debe ser mayor que la de D\_IN). Cuando se detecta el flanco, R\_EDGE permanece a '1' durante un ciclo de CLK. A la izquierda se muestra la declaración de la entidad RISE\_DETECT, y a la derecha se ofrecen las líneas de código que modelan su arquitectura, pero están desordenadas e incluso sobran algunas erróneas.

```
entity RISE_DETECT is
  port(
    D_IN : in STD_LOGIC;
    CLK : in STD_LOGIC;
    RESET : in STD_LOGIC;
    R_EDGE : out STD_LOGIC
  );
end RISE_DETECT;

architecture EXAMEN of RISE_DETECT is
begin
  begin
  elsif rising_edge (CLK) then
  end if;
end EXAMEN;
end process;
if RESET = '1' then
  new_sample := '0'; old_sample := '0';
  old_sample := new_sample; new_sample := D_IN;
process (RESET, CLK)
  R_EDGE <= new_sample and not old_sample;
  R_EDGE <= not new_sample and old_sample;
  R_EDGE <= new_sample and old_sample;
  R_EDGE <= not new_sample and not old_sample;
  variable old_sample, new_sample: STD_LOGIC := '0';
```

Escribe el código correcto de la arquitectura, reordenando estas líneas y eliminando las que sobran:

### SOLUCIÓN:

```
architecture EXAMEN of RISE_DETECT is
begin
  process (RESET, CLK)
    variable old_sample, new_sample: STD_LOGIC := '0';
  begin
    if RESET = '1' then
      new_sample := '0'; old_sample := '0';
    elsif rising_edge (CLK) then
      old_sample := new_sample; new_sample := D_IN;
    end if;
    R_EDGE <= new_sample and not old_sample;
  end process;
end EXAMEN;
```

## ARQUITECTURA E INGENIERIA DE COMPUTADORES. 1 - SEPTIEMBRE - 2005

**C2.-** Durante el desarrollo de un sistema que implementa un cierto algoritmo de procesamiento de vídeo, se detecta el problema de que el equipo no es capaz de trabajar en tiempo real: tarda 65 ms en procesar cada imagen, mientras que la cámara de vídeo manda 25 imágenes por segundo.

a) ¿Cuánto se debe acelerar el sistema para que llegue a trabajar en tiempo real?

Una primera solución puede ser usar un procesador más rápido, de tal manera que de los 250 MHz originales se llegue a 400 MHz. Si embargo, el sistema de memoria se mantendría igual. Hay que tener en cuenta que para este algoritmo se estima que un 30% del tiempo total se gasta en accesos a memoria. La segunda opción es mejorar el sistema de memoria, añadiendo una memoria caché que duplica el rendimiento de este sistema.

b) ¿Cuál es la aceleración que se consigue con cada una de estas dos opciones por separado? A la vista de los resultados, ¿Cuál es mejor? ¿Se consigue con alguna de ellas llegar a trabajar en tiempo real?

c) Finalmente, ¿Qué aceleración se consigue aplicando simultáneamente ambas mejoras? ¿Se consigue así trabajar en tiempo real?

Los apartados **b)** y **c)** deben resolverse **usando necesariamente la Ley de Amdahl.**

### SOLUCIÓN:

a)

25 imágenes por segundo corresponden con una imagen cada  $1/25 = 40$  ms

La aceleración necesaria para que el sistema funcione en tiempo real es:

$$A_{\text{necesaria}} = T_{\text{actual}} / T_{\text{necesario}} = 65 \text{ ms} / 40 \text{ ms} = 1,625$$

**Para que el sistema sea capaz de trabajar en tiempo real hay que mejorar sus prestaciones un 62,5%**

b)

**Mejora del procesador:**

$$F_m = 0,7 \quad A_m = 400 \text{ MHz} / 250 \text{ MHz} = 1,6 \quad A_g = 1 / [(1-F_m) + F_m / A_m] = 1 / [0,3 + 0,7 / 1,6] = 1,36$$

**Mejora de la memoria:**

$$F_m = 0,3 \quad A_m = 2 \quad A_g = 1 / [(1-F_m) + F_m / A_m] = 1 / [0,7 + 0,3 / 2] = 1,18$$

**La mejora del procesador es  $1,36/1,18 = 1,15$  (15%) mejor que la mejora de la memoria, pero ninguna de ellas alcanzan la aceleración de 1,625 necesaria para que el sistema trabaje en tiempo real.**

c)

**Aplicar simultáneamente las dos mejoras:**

Sobre la mejora del sistema de memoria, aplicamos la mejora del procesador. Puesto que ahora el tiempo gastado en accesos a la memoria se ha reducido a la mitad, la fracción mejorada será:

$$F_m = 0,7 / [0,7 + (0,3 / 2)] = 0,7 / 0,85 = 0,82$$

La aceleración de la mejora es la misma que la calculada en el punto anterior, por lo que el resultado será:

$$A_m = 400 \text{ MHz} / 250 \text{ MHz} = 1,6 \quad A_g = 1 / [(1-F_m) + F_m / A_m] = 1 / [0,18 + 0,82 / 1,6] = 1,45$$

**La aceleración total alcanzada será el resultado de multiplicar este valor por el que ya se había obtenido para la mejora del sistema de memoria. Así,  $A_{\text{total}} = A_{\text{mem}} \cdot A_{\text{cpu}} = 1,18 \cdot 1,45 = 1,70$**

**Con esta aceleración si se consigue que el sistema trabaje en tiempo real.**

## ARQUITECTURA E INGENIERIA DE COMPUTADORES. 1 - SEPTIEMBRE - 2005

**C3.-** Se pide identificar las características de un sistema de memoria virtual, a partir de la secuencia de eventos que ocurren al ejecutar la instrucción de lectura de memoria  $\text{LOAD R1, \#70584AH}$  (que significa traer al registro R1 el contenido de la posición de memoria dada por la constante  $70584A_{16}$ ). Ordenada desde el último evento hasta el primero, la secuencia es:

- 1.- En R1 se escribe  $40AA_{16}$ .
- 2.- La memoria DRAM es habilitada para leer el contenido de la posición  $654A_{16}$ .
- 3.- La memoria DRAM es habilitada para leer el contenido de la posición  $50B0_{16}$ .
- 4.- La memoria DRAM es habilitada y se lee el contenido de la posición  $30C0_{16}$ .
- 5.- Se accede a la tabla de primer nivel en la MMU (directorio) y se lee el valor  $3000_{16}$ .

**Nota:** Los descriptores de página tienen el mismo tamaño que una palabra: las posiciones más significativas contienen el marco de página, y las menos significativas se rellenan con bits a '0'. Cualquier tabla en DRAM ocupa el tamaño de una página. Se pide, **justificando necesaria y brevemente** la respuesta:

- a) El tamaño de la memoria virtual y real.
- b) La estructura del sistema de traducción de tabla de páginas y el número total de estas.
- c) El contenido de la posición de memoria  $30C0_{16}$ .
- d) La entrada a la que se accede de la tabla de descriptores de la MMU.

### SOLUCION:

- a) La dirección que envía la CPU es de 24 bits por tanto **la memoria virtual es de 16 MB**. La dirección de memoria es de 16 bits por tanto **la memoria real es de 64kB**.
- b) Si se accede en tres fases a la dirección real, es porque la estructura de la tabla de páginas es de 3 niveles con un primer nivel en la MMU y los dos últimos en DRAM. Todas las tablas en DRAM ocupan el tamaño de una página. Según los valores de las direcciones virtual y real, se conserva el valor 4A, lo cual implica un offset de 8 bits y un TP de 256 bytes. Si el descriptor tiene tamaño palabra (2 bytes) las tablas de los niveles DRAM deben tener 127 entradas, por lo tanto se necesitan 7 bits para indexarlas, quedando 2 para acceder a la tabla de primer nivel. **Por tanto, en el primer nivel hay una tabla en la MMU con 4 entradas, un segundo nivel de tablas con 4 páginas de 128 entradas y un tercer nivel con (4x128) 512 tablas de 128 entradas. Por tanto el número total de tablas es de  $1+4+512 = 517$  tablas.**
- c) La dirección de memoria  $30C0_{16}$  corresponde al acceso a la tabla de segundo nivel, por lo tanto debe leer el descriptor que contiene el marco de página correspondiente a la tabla del tercer nivel.  $50B0_{16}$  es la dirección de acceso a dicha tabla, que se obtiene sumando a la base calculada el índice dado por los bits correspondientes de la dirección virtual multiplicado por el tamaño en bytes del descriptor (2 bytes) en este caso  $1011000_2 \times 2 = 10110000_2 = B0_{16}$ . Por lo tanto la dirección base de la tabla de tercer nivel es  $50B0_{16} - B0_{16} = 5000_{16}$  y el marco de página,  $50_{16}$ . Como el descriptor contiene el marco de página en las posiciones más significativas y el resto de bits están a cero, y los descriptores tienen 2 bytes, **el valor almacenado en la posición  $30C0_{16}$  es  $5000_{16}$ .**
- d) La entrada a la que se accede en la tabla de descriptores es la indexada con 01, que corresponde a los dos primeros bits de la dirección virtual.

## ARQUITECTURA E INGENIERIA DE COMPUTADORES. 1 - SEPTIEMBRE - 2005

**C4.-** Se tiene un sistema RISC segmentado con 5 etapas, en donde el salto se detecta en la segunda y la condición se averigua en la tercera. Se ejecuta la siguiente secuencia de saltos condicionales E-E-E-NE-E-E-NE-NE-NE-E-E-NE, donde E significa salto efectivo y NE no efectivo. Se pide:

- Los ciclos que se pierden por riesgos de control en esta secuencia, suponiendo que el sistema se detiene hasta conocer la condición.
- La mejora porcentual en la pérdida de ciclos por riesgos de control si se dispone de un BTB con predicción histórica. La predicción cambia tras dos fallos consecutivos y por defecto es efectiva.

### SOLUCION:

**a) Detiene el sistema:** Pierde 1 ciclo si no efectivo (5/12) y dos si efectivo (7/12).

	Bcc	N	--	N+1/T					
Decodifica		Bcc	--	N	N+1/T				
Condición			Bcc	--	N	N+1/T			
				Bcc	--	N	N+1/T		
					Bcc	--	N	N+1/T	

Penalización en ciclos:  $5 \cdot 1 + 7 \cdot 2 = 19$  ciclos.

**b1) Predicción BTB Efectiva:** Pierde 0 ciclos si acierta y 2 ciclos si falla

	Bcc	T	T+1	N/T+2					
Decodifica		Bcc	T	T+1	N/T+2				
Condición			Bcc	T	T+1	N/T+2			
				Bcc	T	T+1	N/T+2		
					Bcc	T	T+1	N/T+2	

**b2) Predicción BTB No Efectiva:** Pierde 0 ciclos si acierta y 2 ciclos si falla

	Bcc	N	N+1	N+2/T					
Decodifica		Bcc	N	N+1	N+2/T				
Condición			Bcc	N	N+1	N+2/T			
				Bcc	N	N+1	N+2/T		
					Bcc	N	N+1	N+2/T	

Secuencia	E	E	E	NE	E	E	NE	NE	NE	E	E	NE
Predicción	E	E	E	E	E	E	E	E	NE	NE	NE	E
A/F	A	A	A	F	A	A	F	F	A	F	F	F

Penalización en ciclos:  $6 \cdot 0 + 6 \cdot 2 = 12$  ciclos.

**Mejora  $19/12 = 1,583 \Rightarrow$  Con la mejora se pierden menos ciclos con un porcentaje del 58,3%**

## ARQUITECTURA E INGENIERIA DE COMPUTADORES. 1 - SEPTIEMBRE - 2005

**C5.-** El siguiente fragmento de código, donde el formato de las instrucciones es INST  $R_{Destino}, R_{Fuente1}, R_{Fuente2}$

```
LOAD R1, 0(R10)
MULF R4, R1, R20
ADDF R6, R4, R21
LOAD R2, 4(R10)
MULF R5, R2, R22
ADDF R7, R5, R6
DIVF R8, R7, #5
ADD R10, R10, #8
STORE R8, 0(R10)
```

Se quiere ejecutar en un procesador con múltiples unidades funcionales y que emplea el algoritmo de Tomasulo para planificar la emisión de las instrucciones. Éstas pasan por las tres etapas típicas de estos procesadores: emisión, ejecución y escritura. Emisión y escritura tardan 1 ciclo, mientras que la ejecución necesita 2 ciclos para la unidad de LOAD/STORE, 3 ciclos para la suma/multiplicación en coma flotante (ADDF y MULF), 5 ciclos para la división en coma flotante (DIVF) y 1 ciclo para el resto de operaciones (unidad de enteros). Sólo hay disponibles estos 4 tipos de unidades funcionales.

- a) Suponiendo que el procesador sólo dispone de una unidad funcional de cada tipo, ¿Cuántos ciclos de reloj tardaría en ejecutarse este código? **Responder en la siguiente tabla**, donde en cada celda debes poner los ciclos de reloj durante los que se realiza la operación.

Instrucción	Emisión	Ejecución	Escritura
LOAD R1, 0(R10)	1	2-3	4
MULF R4, R1, R20	2	5-7	8
ADDF R6, R4, R21	9	10-12	13
LOAD R2, 4(R10)	10	11-12	<del>13</del> 14
MULF R5, R2, R22	14	15-17	18
ADDF R7, R5, R6	19	20-22	23
DIVF R8, R7, #5	20	24-28	29
ADD R10, R10, #8	21	22	<del>23</del> 24
STORE R8, 0(R10)	22	30-31	(32)

**NÚMERO TOTAL DE CICLOS DE RELOJ:**

31

32 si se considera que el STORE pasa por la etapa de escritura en el CDB, aunque no escriba nada

- b) ¿Cuál es el tiempo mínimo que se necesitaría para ejecutar este código, suponiendo que hubiera tantas unidades funcionales como se quisiera? ¿Cuál es el número mínimo de unidades funcionales de cada tipo con el que se consigue este tiempo mínimo de ejecución? **Responder en la siguiente tabla.**

Instrucción	Emisión	Ejecución	Escritura
LOAD R1, 0(R10)	1	2-3	4
MULF R4, R1, R20	2	5-7	8
ADDF R6, R4, R21	3	9-11	12
LOAD R2, 4(R10)	4	5-6	7
MULF R5, R2, R22	5	8-10	11
ADDF R7, R5, R6	6	13-15	16
DIVF R8, R7, #5	7	17-21	22
ADD R10, R10, #8	8	9	10
STORE R8, 0(R10)	9	23-24	(25)

**NÚMERO TOTAL DE CICLOS DE RELOJ:**

24

25 si se considera que el STORE pasa por la etapa de escritura en el CDB, aunque no escriba nada

**UNIDADES LOAD/STORE: 2**

**MULF/ADDF: 4**

**DIVF: 1**

**ENTEROS: 1**

## ARQUITECTURA E INGENIERIA DE COMPUTADORES. 1 - SEPTIEMBRE - 2005

**P1.-** Se tiene un sencillo sistema RISC donde el bus de datos y de direcciones son ambos de 16 bits. El sistema dispone de una unidad caché no unificada. Ambas cachés tienen una capacidad de 16 KB, tienen una estructura asociativa de 4 vías, con una política LRU para el reemplazamiento, guardan 8 palabras por bloque y existen bits de suciedad o modificación donde corresponda. Se sabe que el acceso a memoria DRAM es de 100 ns y el acceso a caché es de 10 ns. Inicialmente ambas caches están vacías.

**Justificando brevemente** las respuestas, se pide:

- a) Los campos en los que se divide la dirección para el acceso a las caches.
- b) La tasa de aciertos en la caché de instrucciones.
- c) La tasa de aciertos en la caché de datos
- d) El tiempo total utilizado para el acceso a memoria en la ejecución del código mostrado.

**Nota:** El código suma 100 a un vector de 50 elementos de 2 bytes, ubicado a partir de la posición  $4400_{16}$  y lo reescribe modificado en la misma posición.

Dirección	Instrucción	
00EC	MOVE 0,R1	; $0 \Rightarrow R1$
00EE	MOVE 50,R2	; $50 \Rightarrow R2$
00F0	L1: SUB R2,1,R2	; $R2 - 1 \Rightarrow R2$
00F2	Bneg Fin	; Salto condicional. Si N = 1, salta a Fin
00F4	LOAD #4400,R1,R4	; $\text{Mem}[4400+(R1)] \Rightarrow R4$
00F6	ADD R1,2,R1	; $R1 + 2 \Rightarrow R1$
00F8	CALL Sub	; Llamada a la rutina Sub
00FA	STORE R6, #4400,R1	; $R6 \Rightarrow \text{Mem}[4400+(R1)]$
00FC	JUMP L1	; Salta incondicional a la etiqueta L1
00FE	Fin: STOP	; Fin del programa principal
.....	.....	
0400	Sub: ADD R4,100,R6	; $R4 + 100 \Rightarrow R6$
0402	RETURN	; Vuelve de la rutina

### SOLUCION:

- a) Analizando los datos del enunciado se obtiene que:

$$2^{14} \text{ bytes} / 2^2 \text{ vías} = 2^{12} \text{ bytes/vía.}$$

$$2^{12} \text{ bytes/vía} / 2^4 \text{ bytes/bloque} = 2^8 \text{ bloques/vía} = 2^8 \text{ entradas.}$$

Los campos en los que se divide la dirección son:

Etiqueta	Indice	bytes/bloque
4	8	4

- b) El número de accesos a la memoria de instrucciones es:

$$2 \text{ (instr. iniciales)} + 50 \times 7 \text{ (bucle principal)} + 50 \times 2 \text{ (bucle rutina)} + 3 \text{ (instr. finales)} = 455 \text{ accesos}$$

Hay 3 fallos que ocurren en las direcciones 00EC, 00F0 y 0400, que llenan la primera vez los índices en hexadecimal 0E, 0F y 40.

En cada fallo, se carga un bloque completo que luego suponen los 452 aciertos.

**Hay por tanto 3 fallos en 455 llamadas es decir la tasa de aciertos es: 99,34%**

- c) El número de accesos a la memoria de datos es:

$$50 \text{ (lecturas)} + 50 \text{ (escrituras)} = 100 \text{ accesos.}$$

Hay 7 fallos que ocurren en las direcciones 4400, 4410, 4420, 4430, 4440, 4450 y 4460, que llenan la primera vez los índices en hexadecimal 40, 41, 42, 43, 44, 45 y 46.

En cada fallo, se carga un bloque completo que luego suponen 43 aciertos en las lecturas y 50 aciertos en las escrituras.

La presencia de los bits de modificación indican que se trata de una cache de post-escritura. Durante la ejecución del programa, no se precisa ningún reemplazamiento, por tanto se considera que  $w_M = 0$ .

**Hay por tanto 7 fallos en 100 llamadas es decir la tasa de aciertos es: 93%**

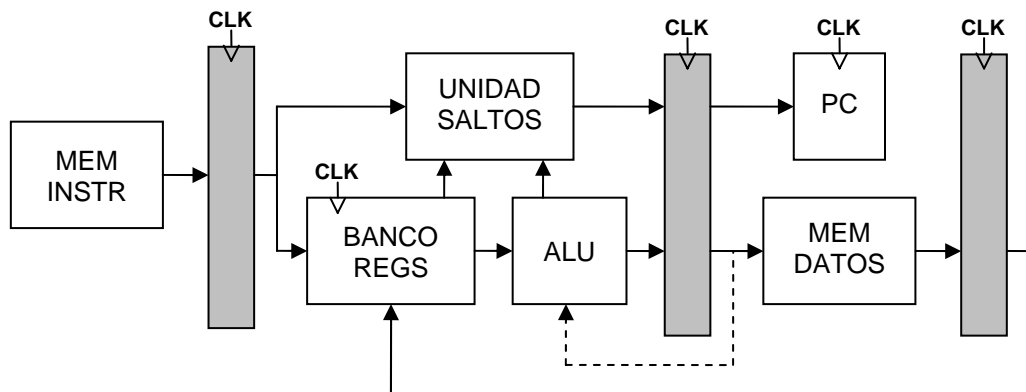
- d) La ecuación que calcula el tiempo total viene dada por:

$$T^{\text{INSTRUCCIONES}} = N^{\circ}\text{Acc}^{\text{INS}} \times \{ t_C + (1-H) t_B \} = 455 \{ 0,01 + (0,0066) \times (8 \times 0,1) \} = 6,95 \text{ msec}$$

$$T^{\text{DATOS}} = N^{\circ}\text{Acc}^{\text{DAT}} \times \{ t_C + (1-H) t_B \} = 100 \{ 0,1 + (0,07) \times (8 \times 0,1) \} = 6,60 \text{ msec}$$

$$T^{\text{TOTAL}} = T^{\text{INSTRUCCIONES}} + T^{\text{DATOS}} = 6,95 + 6,60 = 13,55 \text{ msec}$$

**P2.-** La siguiente figura muestra un diagrama de bloques muy simplificado de un procesador RISC segmentado con cauce único de emisión de instrucciones y ejecución en orden. Los elementos sombreados son los registros de segmentación, y la línea punteada es un camino para adelantamiento de datos.



Todos los registros capturan los datos con el flanco de subida de la señal de reloj. La política con respecto a los saltos es parar el *pipeline* en cuanto se detecta uno, y mantenerlo así hasta que la ejecución de la instrucción de salto llega a la etapa en que se actualiza el PC. Si el salto es efectivo, se elimina del *pipeline* la instrucción (o instrucciones) que pudieran haber sido capturadas antes de la parada, y si no, se continúa con su ejecución.

- a) ¿La arquitectura del procesador es Harvard o Von Neumann? ¿Cuántas etapas *pipeline* tiene el procesador? Indicar brevemente la función de cada etapa, que se nombrarán de la forma E1, E2, etc...  
**Responder a continuación, justificando las respuestas**

**La arquitectura del procesador es Harvard, porque tiene memorias separadas para instrucciones y datos.**

**El procesador tiene 4 etapas pipeline, porque tiene 3 registros de segmentación:**

**E1: Captura de instrucción**

**E2: Decodificación, captura de operandos desde el banco de registros, ejecución en ALU, comprobación de la condición en saltos**

**E3: Acceso a memoria de datos y actualización del PC.**

**E4: Escritura de los resultados en el banco de registros.**

El siguiente fragmento de código se ejecuta en este procesador:

```

I1    LD R10, 4(R5)
I2    ADD R6, R3, R10
I3    AND R7, R3, R4
I4    SUB R8, R7, R6
I5    BEQ salto
I6    XOR R2, R10, R11
I7    ADD R5, R1, R2
I8    AND R7, R10, R12
I9    OR R8, R6, R9
I10   SUB R10, R2, R11
    
```

El formato de las instrucciones es INSTR  $R_{Destino}, R_{Fuente1}, R_{Fuente2}$  y la condición en el salto es falsa (el salto no se efectúa). Calcula el tiempo de ejecución y completa el cronograma del pipeline (empleando las plantillas que se adjuntan) para estos tres casos:

# ARQUITECTURA E INGENIERIA DE COMPUTADORES. 1 - SEPTIEMBRE - 2005

b) Sin considerar adelantamiento de datos

Ciclo de reloj →	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
LD R10, 4(R5)	E1	E2	E3	E4																		
ADD R6, R3, R10		E1	E2	E2	E2	E3	E4															
AND R7, R3, R4			E1	E1	E1	E2	E3	E4														
SUB R8, R7, R6						E1	E2	E2	E2	E3	E4											
BEQ salto							E1	E1	E1	E2	E3	E4										
XOR R2, R10, R11										E1	E1	E2	E3	E4								
ADD R5, R1, R2												E1	E2	E2	E2	E3	E4					
AND R7, R10, R12													E1	E1	E1	E2	E3	E4				
OR R8, R6, R9																E1	E2	E3	E4			
SUB R10, R2, R11																	E1	E2	E3	E4		

c) Usando el camino para el adelantamiento de datos

Ciclo de reloj →	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
LD R10, 4(R5)	E1	E2	E3	E4																		
ADD R6, R3, R10		E1	E2	E2	E2	E3	E4															
AND R7, R3, R4			E1	E1	E1	E2	E3	E4														
SUB R8, R7, R6						E1	E2	E2	E2	E3	E4											
BEQ salto							E1	E1	E1	E2	E3	E4										
XOR R2, R10, R11										E1	E1	E2	E3	E4								
ADD R5, R1, R2												E1	E2	E3	E4							
AND R7, R10, R12													E1	E2	E3	E4						
OR R8, R6, R9														E1	E2	E3	E4					
SUB R10, R2, R11															E1	E2	E3	E4				



# ARQUITECTURA E INGENIERIA DE COMPUTADORES. 1 - SEPTIEMBRE - 2005

- d) Manteniendo el adelantamiento de datos y cambiando la política de saltos, usando predicción estática no efectiva

Ciclo de reloj →	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
LD R10, 4(R5)	E1	E2	E3	E4																		
ADD R6, R3, R10		E1	E2	E2	E2	E3	E4															
AND R7, R3, R4			E1	E1	E1	E2	E3	E4														
SUB R8, R7, R6						E1	E2	E2	E2	E3	E4											
BEQ salto							E1	E1	E1	E2	E3	E4										
XOR R2, R10, R11										E1	E2	E3	E4									
ADD R5, R1, R2											E1	E2	E3	E4								
AND R7, R10, R12												E1	E2	E3	E4							
OR R8, R6, R9													E1	E2	E3	E4						
SUB R10, R2, R11														E1	E2	E3	E4					