**Arquitectura e Ingeniería de Computadores - 3r Grado Ing. Informática - ETSINF - UPV**

Ejercicios y soluciones de la Unidad Temática 2. "Computadores segmentados" (1)

# Segmentación básica

## La ruta de datos del MIPS

**Ejercicio 2.1.** Un procesador compatible binario con el MIPS64 posee su ciclo de instrucción segmentado en 5 fases:

**IF:** Búsqueda de la instrucción a ejecutar.

**ID:** Decodificación de la instrucción y lectura de los registros operandos.

**EX:** Operación en la U.A.L. Cálculo de la condición y escritura del PC en las instrucciones de salto.

**M:** Acceso a memoria en las instrucciones de Carga y Almacenamiento.

**WB:** Escritura del resultado sobre el registro destino.

La máquina resuelve los riesgos de control mediante *predict-not-taken*. También posee antememorias de instrucciones y datos separadas, así como dos puertos de lectura y uno de escritura en el banco de registros.

Sobre dicho procesador se está ejecutando el siguiente bucle, compuesto de $n$ iteraciones:

```
      ...
L :   ld $t1,X($t2)
      dadd $t1,$t1,$t3
      sd $t1,X($t2)
      daddi $t2,$t2,8
      daddi $t4,$t4,-1
      bnez $t4,L
```

1. Dibuja un diagrama en el que se indique, para cada instrucción y ciclo de reloj, qué fase de la instrucción se está completando. Considera sólo la primera iteración. Calcula los CPI y el tiempo de ejecución obtenidos en función del número de iteraciones $n$. Considera los siguientes casos:

   *a*) Los riesgos de datos se resuelven mediante la inserción de ciclos de parada.

   *b*) Los riesgos de datos se resuelven mediante la técnica del cortocircuito.

***Solución:***

1. Puesto que el PC se escribe en la tercera fase del ciclo de instrucción, se cancelan dos instrucciones:

```
                                pc
d:        bnez $t4,L    IF ID EX M  WB
d+4:      <sgte>           IF ID X
d+8:      <sgte+1>            IF X
d+12/L:                          IF ID EX M  WB
```

*a*) Código original con ciclos de parada.

```
                        1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17 18 19
begin: ld $t1,X($t2)    IF ID EX M  WB
       dadd $t1,$t1,$t3    IF id id ID EX M  WB
       sd $t1,X($t2)          if if IF id id ID EX M  WB
       daddi $t2,$t2,8              if if IF ID EX M  WB
       daddi $t4,$t4,-1                   IF ID EX M  WB
       bnez $t4,begin                        IF id id ID EX M  WB
       <sgte>                                   if if IF ID X
       <sgte+1>                                        IF X
begin: ld $t1,X($t2)                                      IF ID EX M  WB
```

Para una iteración, se ejecutan 6 instrucciones en 14 ciclos de reloj. Por tanto:

$$\textit{CPI} = \frac{14}{6} = 2{,}33 \text{ ciclos}$$

Y el tiempo de ejecución para $n$ iteraciones es $14 \cdot n$ ciclos:

$$T_{ej} = I \cdot \textit{CPI} \cdot T = 6n \cdot 2{,}33 \cdot T = 14 \cdot n \cdot T \text{ seg}$$

*b*) Código original con cortocircuitos.

```
                        1  2  3  4  5  6  7  8  9  10 11 12 13 14
begin: ld $t1,X($t2)    IF ID EX M  WB
       dadd $t1,$t1,$t3    IF id ID EX M  WB
       sd $t1,X($t2)          if IF ID EX M  WB
       dadd $t2,$t2,#8              IF ID EX M  WB
       dsub $t4,$t4,#1                 IF ID EX M  WB
       bnez $t4,begin                     IF ID EX M  WB
       <sgte>                                IF ID X
       <sgte+1>                                 IF X
begin: ld $t1,X($t2)                               IF ID EX M  WB
```

Para una iteración, se ejecutan 6 instrucciones en 9 ciclos de reloj. Por lo tanto:

$$\textit{CPI} = \frac{9}{6} = 1{,}5$$

Y el tiempo de ejecución para $n$ iteraciones es $9 \cdot n$ ciclos:

$$T_{ej} = I \times \textit{CPI} \times T = 6n \times 1{,}5 \times T = 9 \cdot n \cdot T \text{ seg}$$

□

**Ejercicio 2.2.** Se tiene el siguiente código en alto nivel:

```
typedef struct elem {
   int x[1..10];
   struct elem * siguiente;
} elem_t;

int cont;
elem_t * p;

...

do {
  cont = cont + 1;
  p = p->siguiente;
} while (p != NULL);
...
```

El tipo `int` y los punteros ocupan 64 bits. La constante `NULL` se representa con un 0. El compilador genera el bucle como se muestra seguidamente:

```
; cont se ubica en R2
; p se ubica en R1
...
eti: dadd r2, r2, #1
     ld r1, 80(r1)
     bnez r1, eti
```

Dicho código se pretende ejecutar sobre distintas versiones de un procesador segmentado en 5 etapas:

**IF:** Búsqueda de la instrucción a ejecutar.

**ID:** Decodificación de la instrucción y lectura de los registros operandos (2º semiciclo del reloj).

**EX:** Operación en la U.A.L.

**ME:** Acceso a memoria en las instrucciones de Carga y Almacenamiento.

**WB:** Escritura del resultado sobre el registro destino (1er semiciclo del reloj).

El procesador resuelve los riesgos de datos mediante la técnica del cortocircuito, los de control mediante *predict-not-taken* y funciona a 100 MHz. También emplea arquitectura *Harvard* (caches separadas para instrucciones y datos).

En cada uno de los supuestos siguientes, calcula la latencia de salto y los CPI, los MIPS y el tiempo de ejecución que alcanza el procesador suponiendo que el bucle se ejecuta 10.000 veces:

1. El cálculo de la dirección efectiva, condición de salto y la escritura del PC se realiza en ID.
2. El cálculo de la dirección efectiva y condición de salto se realiza en EX, y la escritura del PC se realiza en M.

***Solución:***

1. Escritura del PC en ID.

   La latencia de salto es uno, por lo que se cancela la instrucción siguiente al salto en cada iteración (excepto en la última). Además, se ejecutan tres instrucciones en cada iteración. El riesgo de datos entre *ld* y *bnez* requiere dos ciclos de parada, dando un total de seis ciclos por iteración. En consecuencia:

   $$\overline{\text{CPI}} = \frac{6}{3} = 2 \text{ ciclos}$$

   $$\text{MIPS} = \frac{I}{T_{ej} \cdot 10^6} = \frac{I}{I \cdot CPI \cdot T \cdot 10^6} = \frac{f(\text{en MHz})}{\text{CPI}} = \frac{100}{2} = 50 \text{ MIPS}$$

   Para procesar 10.000 iteraciones harán falta 60.000 ciclos de reloj, que, a 10 ns por ciclo, son 600.000 ns = 0.6 ms. Al mismo resultado llegamos tras aplicar la ecuación del tiempo de ejecución con $I = 3 \cdot 10^4$ instrucciones

   $$T(10000 \text{ iteraciones}) = (3 \cdot 10^4) \cdot 2 \cdot \frac{1}{100 \cdot 10^6} = 6 \cdot 10^{-4} \text{ s.}$$

2. Escritura del PC en M.

   La latencia de salto es ahora tres, por lo que se cancelan tres instrucciones en cada iteración mientras que se ejecutan tres instrucciones en cada iteración. El riesgo de de datos entre *ld* y *bnez* requiere ahora un único ciclo de parada. Por lo tanto, el bucle consume siete ciclos por iteración. Se obtienen los siguientes parámetros:

$$\overline{\text{CPI}} = \frac{7}{3} = 2{,}33 \text{ ciclos}$$

$$\text{MIPS} = \frac{100}{2{,}33} = 42{,}92 \text{ MIPS}$$

   Para procesar 10.000 iteraciones harán falta 70.000 ciclos de reloj, que, a 10 ns por ciclo, son 700.000 ns = 0.7 ms.

□

**Ejercicio 2.3.** Un computador, valorado en 2000€, lleva un MIPS/LC, idéntico al MIPS (5 etapas de segmentación: búsqueda de la instrucción (IF), decodificación y lectura de registros (ID), Ejecución (EX), Acceso a memoria (MEM) y Escritura de registros (WB)) pero con una sola memoria cache común para instrucciones y datos, con cortocircuitos, *predict-not-taken*, latencia de salto 1 y reloj a 80 MHz. Se tiene instalado un compilador de código abierto que compila el siguiente texto:

```
do {
  if (v[i] != 0) {
    temp = v[i];
    v[i] = w[i];
    w[i] = temp;
  }
  i = i-1;
} while (i != 0);
```

y genera el siguiente código:

```
etti1:   ld r2,v(r1)
         beqz r2,eti2
         ld r3,w(r1)
         sd r3,v(r1)
         sd r2,w(r1)
eti2:    dadd r1,r1,-8
         bnez r1,eti1
```

El bucle se aplica a vectores con un 80 % de componentes iguales a 0.

1. Calcula el CPI medio para tallas $n$ grandes.

2. Supón que el bucle original es una buena muestra de la carga usual de dicho computador. Para mejorar el rendimiento, considera dos posibles inversiones:

   - Cambiar el procesador por la versión MIPS/ST con memorias cache de instrucciones y datos separadas, con un coste de 200€.
   - Comprar un compilador comercial, valorado en 200€, capaz de optimizar el código anterior reduciendo en 3 ciclos de reloj cada iteración en que $v[i] \neq 0$ y en 2 ciclos cada iteración en que $v[i] = 0$. El número de instrucciones ejecutadas no se modifica.

Desde el punto de vista de la relación coste/prestaciones, y suponiendo que sólo podemos gastarnos 200€, ¿sería interesante alguna de las dos mejoras anteriores? En caso afirmativo, ¿cuál convendría aplicar?

***Solución:***

1. Cuando $v(r1) \neq 0$, en el bucle se ejecutan siete instrucciones. Para resolver los riesgos, se introducen 7 ciclos de parada:
   - Dos ciclos que ha de esperar la instrucción `beqz r2,eti2` para que la fase ID disponga del valor de r2 generado por la instrucción previa `ld r2,v(r1)`. Recuérdese que si la latencia de salto es uno, el PC se escribe en la segunda etapa (ID) del ciclo de instrucción, por lo que la condición debe calcularse también en dicha etapa.
   - Tres ciclos ocasionados por las restantes instrucciones de carga y almacenamiento al producirse un riesgo estructural por tener una única cache de instrucciones y datos.
   - Un ciclo que ha de esperar la instrucción `bnez r1,eti1` para que la fase ID disponga del valor de r1 modificado por la instrucción previa `daddi r1,r1,-8`.
   - Un ciclo debido a la cancelación (por *predict-not-taken*) de la instrucción siguiente tras el salto `bnez r1,eti1`.

   El total de ciclos requeridos es de 7+2+3+1+1=14.

   Si $v(r1) = 0$, se ejecutan cuatro instrucciones. Los ciclos de parada son los 3 correspondientes a los saltos `beqz` (2) y `bnez` (1) y los 2 por las cancelaciones de sus instrucciones siguientes, porque el resto de instrucciones de carga y almacenamiento no se ejecutan. El total de ciclos es, pues, de 4+3+2=9.

   Para calcular el CPI medio hay que tener en cuenta las proporciones de ceros en el vector $v$:

$$\overline{CPI} = \frac{\text{Número de ciclos}}{\text{Número de instrucciones}} = \frac{14 \cdot 0{,}2 + 9 \cdot 0{,}8}{7 \cdot 0{,}2 + 4 \cdot 0{,}8} = 2{,}17$$

2. Cálculo de las mejoras en tiempo medio de ejecución del bucle, medido en ciclos, en uno y otro caso:
   - nuevo procesador, en el que se eliminan los tres riesgos estructurales sólo cuando $v(r1) \neq 0$:
$$\Delta t = \frac{14 \cdot 0,2 + 9 \cdot 0,8}{11 \cdot 0,2 + 9 \cdot 0,8} = 1,06$$
   - nuevo compilador:
$$\Delta t = \frac{14 \cdot 0,2 + 9 \cdot 0,8}{11 \cdot 0,2 + 7 \cdot 0,8} = 1,28$$

   Para el mismo incremento de coste (10 %), queda claro que la mejora apropiada es la adquisición del nuevo compilador. Por otra parte, la mejora obtenida cambiando el procesador es inferior al incremento del coste, por lo que, desde el punto de vista de mantener una relación coste–prestaciones constante, no sería interesante.

□

**Ejercicio 2.4.** El ciclo de instrucción de un procesador *load/store* no segmentado se descompone en las siguientes fases (se indica entre paréntesis la duración de cada una):

- LI (10 ns): lectura de instrucción.
- DI (5 ns): decodificación de la instrucción y lectura de registros fuente.

- EXE (10 ns): cálculo de direcciones efectivas en instrucciones L/S, operación en instrucciones ALU, cálculo de condición y de valor del PC en instrucciones de salto.
- EPC (5 ns): escritura de PC en instrucciones de salto
- MEM (10 ns): acceso a memoria en instrucciones L/S
- ER (5 ns): escritura de registro destino en instrucciones de almacenamiento y de ALU.

El autómata que implementa el circuito de control cableado genera las fases en función del código de operación. El reloj funciona a 200 MHz, de manera que unas fases requieren dos ciclos y otras sólo uno. Todos los ciclos de instrucción comienzan por las fases LI y DI. Según el tipo de instrucción, las restantes fases del ciclo son (se indica entre paréntesis la frecuencia de cada tipo de instrucción):

- Instrucciones de carga (20 %): EXE, MEM y ER
- Instrucciones de almacenamiento (10 %): EXE y MEM
- Instrucciones ALU (50 %): EXE y ER
- Instrucciones de salto (20 %): EXE y EPC

Se pretende segmentar el procesador, utilizando registros de 2 ns de retardo y un reloj con desfase nulo. Desaparece la fase EPC y toda la lógica de salto pasa a la etapa DI, con lo que el retardo de esta etapa es ahora de 10 ns. El procesador queda con las 5 etapas LI, DI, EXE, MEM y ER. Se toman medidas reales y se observa que el 5 % de las instrucciones de carga generan un ciclo de parada por riesgo de datos, y que se cancelan el 10 % de las instrucciones posteriores al salto.

Se pide:

1. Los CPI del procesador no segmentado.
2. La frecuencia del reloj del procesador segmentado.
3. Los CPI del procesador segmentado.
4. La aceleración en el tiempo de ejecución obtenido por la segmentación.

***Solución:***

1. CPI del procesador no segmentado.
   Obtengamos primero el valor de CPI para cada tipo de instrucción:

| % | tipo | CPI |
|---|---|---|
| 20 | carga | 8 |
| 10 | almacenamiento | 7 |
| 50 | cálculo (ALU) | 6 |
| 20 | bifurcación | 6 |

   La media ponderada es:

$$\overline{CPI}_{NS} = 0{,}20 \cdot 8 + 0{,}10 \cdot 7 + 0{,}50 \cdot 6 + 0{,}2 \cdot 6 = 6{,}5 \text{ ciclos}$$

2. Frecuencia del reloj del procesador segmentado.

   El máximo retardo de etapa es de 10 ns, y el de registro de 2 ns. Por lo tanto, el periodo de reloj es $t_S = 10 + 2 = 12$ ns y la frecuencia:

$$f = \frac{1}{t_S} = \frac{1}{12 * 10^{-9}} = 83{,}3 * 10^6 \text{ Hz} = 83{,}3 \text{ MHz}$$

3. CPI del procesador segmentado.

   Cada instrucción buscada contribuye con un ciclo al tiempo de ejecución. También hay que contabilizar los ciclos perdidos por conflictos de datos en el 5 % de casos del 20 % de instrucciones de carga y en el 10 % de casos del 20 % de las instrucciones de salto.

| cantidad | tipo | CPI |
|---|---|---|
| 20 | carga | 1+$(0{,}05 \cdot 1)$=1,05 |
| 10 | almacenamiento | 1 |
| 50 | cálculo (ALU) | 1 |
| 20 | bifurcación | 1+$(0{,}10 \cdot 1)$=1,10 |
| 100 | total | |

   La media ponderada es:

$$\overline{CPI}_S = \frac{20 \cdot 1{,}05 + 10 \cdot 1 + 50 \cdot 1 + 20 \cdot 1{,}10}{100} = 1{,}03 \text{ ciclos}$$

   Otra forma de proceder es considerar que, al estar segmentado, el CPI será de 1 más el número medio de ciclos de parada que origina cada instrucción. En nuestro caso, solo el 5 % de las instrucciónes de carga y el 10 % de las bifurcaciones insertan un ciclo de parada. Por lo tanto:

$$\overline{CPI}_S = 1 + \frac{20 * 0{,}05 + 20 * 0{,}10}{100} = 1{,}03 \text{ ciclos}$$

4. Aceleración debida a la segmentación.

   Dividiendo las expresiones del tiempo de ejecución del procesador no segmentado y segmentado, obtenemos:

$$S = \frac{T_{NS}}{T_S} = \frac{I * CPI_{NS} * t_{NS}}{I * CPI_S * t_S} = \frac{I * 6{,}5 * 5}{I * 1{,}03 * 12} = 2{,}63$$

□

**Ejercicio 2.5.**

Se tiene un procesador segmentado en 5 etapas (IF: búsqueda de la instrucción; ID: decodificación y lectura de registros; EX: operación en la ud. aritmética; MEM: acceso a memoria y WB: escritura de registros). El procesador incorpora el juego de instrucciones del MIPS y posee cache de instrucciones y de datos separadas. La frecuencia de reloj es de 200 MHz. Los riesgos de datos y de control se resuelven mediante la técnicas del *forwarding* e insertando dos ciclos de parada cada vez que aparece una instrucción de salto, respectivamente.

Los programas ejecutan, por término medio, un 18 % de saltos, un 39 % de cargas/almacenamientos y un 43 % de instrucciones aritméticas. Las cargas son doble frecuentes que los almacenamientos. Los accesos a *bytes* y *halfwords* suponen un 20 % de los accesos a memoria. La frecuencia de riesgos de datos entre una instrucción LOAD y otra posterior que consume el dato procedente de la memoria es la siguiente:

| Frecuencia: 25 % | Frecuencia: 15 % |
|---|---|
| `LOAD R1, ...`<br>Instrucción que lee `R1`<br>... | `LOAD R1, ...`<br>Instrucción que no lee `R1`<br>Instrucción que lee `R1` |

Con el objeto de mejorar las prestaciones, se plantea realizar las siguientes modificaciones:

- Eliminar las instrucciones de acceso a *bytes* y *halfwords* del juego de instrucciones. Como consecuencia, los programas que necesiten esta funcionalidad deberán utilizar otras instrucciones del procesador:

| | |
|---|---|
| `LB R1,x` | `LW R1,x'` |
| o | `SRL R1,R1,#pos` |
| `LH R1,x` | `AND R1,R1,#masc` |

| | |
|---|---|
| `SB R1,x` | `LW R2,x'` |
| o | `SLL R1,R1,#pos` |
| `SH R1,x` | 5 instrucciones ALU más |
| | `SW x',R1` |

- Aumentar la frecuencia de reloj, al simplificar el diseño del procesador.

Se pide:

1. Los CPI del procesador original.
2. El número de instrucciones del procesador modificado en relación al original.
3. Los CPI del procesador modificado.
4. La frecuencia de reloj que debería alcanzarse, como mínimo, para que sea interesante incorporar las modificaciones propuestas.

***Solución:***

1. CPI del procesador original.

   Al ser un procesador segmentado, los CPI serán 1 más los ciclos medios de parada por instrucción. En este caso, se insertan cuando aparecen saltos (18 %, 2 ciclos de parada) y cuando se producen riesgos de datos en los que interviene una instrucción de carga y otra instrucción inmediatamente a continuación que consume el dato cargado (25 % de las cargas, 1 ciclo de parada). Sabemos que hay un 39 % de carga/almacenamiento, y que las cargas son doble frecuente que los almacenamientos. Si $c$ y $a$ son los porcentajes de carga y almacenamiento, respectivamente, tenemos:

   $$\left.\begin{array}{rcl} c+a & = & 39 \\ c & = & 2a \end{array}\right\}$$

   Resolviendo, $c = 26\,\%$ y $s = 13\,\%$.

   Por lo tanto, los ciclos de parada son:

   $$18\,\% \cdot 2(\text{Saltos}) + 26\,\% \cdot 0{,}25 \cdot 1(\text{Cargas}) = 0{,}425$$

   y $CPI = 1{,}43$ ciclos.

2. Número de instrucciones en el procesador mejorado.

   El procesador mejorado ejecuta las mismas instrucciones originales, menos las que hemos sustituido, más las que equivalen a las sustituidas.

   Teniendo en cuenta que los accesos a *bytes y halfwords* suponen el 20 % de los accesos a memoria, la frecuencias de aparición de LB/LH y SB/SH es de:

   Frecuencia LB/LH = $0{,}26 \cdot 0{,}20$ = 0.052

   Frecuencia SB/SH = $0{,}13 \cdot 0{,}20$ = 0.026

   Cada instrucción LB/LH se sustituye por 3 nuevas instrucciones, y cada instrucción SB/SH se sustituye por 8 nuevas instrucciones. Por tanto, la nueva cuenta de instrucciones es:

   $I' = I - 0{,}052 \cdot 1 + 0{,}052 \cdot 3 - 0{,}026 \cdot 1 + 0{,}026 \cdot 8 = 1{,}286I$

3. CPI del procesador mejorado.
   Los ciclos de parada se modifican debido a que las sustituciones de LB/LH introducen siempre un ciclo de parada, independientemente de si antes lo producían o no. También hay que tener en cuenta que el numero total de instrucciones ha cambiado:
   Nº de ciclos de parada: $\frac{0,18}{1,29}*2(Saltos)+\frac{0,26}{1,29}*0,8*0,25*1(LW)+\frac{0,26}{1,29}*0,2*1(LB/LH) = 0,359$
   CPI'=1,36 ciclos.
4. Compararemos el tiempo de ejecución de la mejora propuesta con la configuración original.
   Original: $Tej = I \times CPI \times T = I \cdot 1{,}43 \cdot 5 = 7{,}15I$ ns
   Mejora: $Tej' = I' \times CPI' \times T' = 1{,}286I \cdot 1{,}359 \cdot T' = 1{,}75IT'$ ns
   Igualando y despejando:
   T' = 4.08 ns. Por lo tanto, la frecuencia debería aumentar hasta 245 MHz para fuera interesante la propuesta.

□

**Ejercicio 2.6.** Un procesador con arquitectura registro-memoria tiene el ciclo de instrucción segmentado en 6 etapas:

- IF: Búsqueda de la instrucción e incremento del PC.
- RF: Decodificación y lectura de registros (2º semiciclo).
- ALU1: Cálculo de la dirección efectiva en accesos a memoria y saltos.
- MEM: Acceso a memoria.
- ALU2: Operaciones aritméticas, evaluación de la condición de salto y escritura del nuevo PC, en su caso.
- WB: Escritura del registro destino ($1^{er}$ semiciclo).

Todas las instrucciones ejecutan las 6 etapas. Hay dos tipos de instrucciones aritméticas:

**Tipo R**: ALUop Rd,Rs,Rt

**Tipo M**: ALUop Rd,Rs,desp(Rt)

1. Para evitar riesgos estructurales, ¿cuál es el número mínimo de sumadores necesario en esta segmentación?
   Encuentra el número mínimo de puertos de lectura/escritura tanto para el banco de registros como para memoria, a fin de evitar riesgos estructurales.
2. Si la máquina resuelve los riesgos de control con *predict-not-taken*, ¿cuántas instrucciones se cancelan cuando el salto es efectivo?
3. ¿Tiene interés aplicar una estrategia *predict-taken* para los saltos condicionales hacia posiciones anteriores del código en este procesador? En caso afirmativo, ¿qué penalización en ciclos de reloj conllevan los saltos correctamente predichos?

***Solución:***

1. Recursos necesarios para evitar riesgos estructurales.
   - Número de sumadores. Hay tres etapas que pueden requerir un sumador:

IF: para incrementar el PC

ALU1: para calcular la dirección efectiva de alguna referencia a memoria

ALU2: para realizar alguna operación ALU

- Banco de registros: Se leen (2 registros) en la etapa RF, y se escribe (1 registro) en la etapa WB. Por tanto, necesitamos al menos **1 puerto de escritura y 2 puertos de lectura**.
  Por otra parte, puesto que el banco de registros es de de ciclo partido, una optimización sería tener **un único** puerto combinado de lectura/escritura más otro puerto sólo de lectura.
- Memoria: Se lee en la etapa IF (instrucción) y en la etapa MEM (dato), y se escribe en la etapa MEM (dato). Por tanto, se requieren dos puertos, uno de lectura y otro de lectura/escritura o bien una utilizar antememorias de instrucciones y datos separadas (arquitectura Harvard).

2. Instrucciones canceladas cuando el salto es efectivo.

   Puesto que el PC se escribe en la fase ALU2, es cuando la instrucción de salto está ejecutando la fase WB cuando se puede buscar la instrucción destino del salto. Por lo tanto, se buscan y comienzan a ejecutar hasta 4 instrucciones entre la del salto y la instrucción destino del mismo.

| Salto | IF | RF | A1 | ME | **A2** | WB | |
|---|---|---|---|---|---|---|---|
| instr. 1 | | IF | RF | A1 | ME | | |
| instr. 2 | | | IF | RF | A1 | | |
| instr. 3 | | | | IF | RF | | |
| instr. 4 | | | | | IF | | |
| Destino | | | | | | IF | RF |

3. Interés de una estrategia *predict-taken*.

   Sí, ya que:

   - Se sabe la dirección de salto antes que la condición.
   - Los saltos condicionales hacia atrás suelen ser efectivos en un alto porcentaje de los casos.

   La penalización cuando la condición no se cumple es de dos ciclos de reloj:

| Salto | IF | RF | **A1** | ME | A2 | WB | |
|---|---|---|---|---|---|---|---|
| instr. 1 | | IF | RF | | | | |
| instr. 2 | | | IF | | | | |
| Destino | | | | IF | RF | A1 | ME |

□

**Ejercicio 2.7.**

Los siguientes diagramas instrucciones–tiempo corresponden a la ejecución de ciertos fragmentos de código en varios procesadores. Indica, para cada uno de los casos, qué técnica se utiliza para resolver los riesgos de datos (inserción de ciclos de parada o cortocircuito) y los riesgos de control (inserción de ciclos de parada, *predict-not-taken* o salto retardado), así como la fase en la que se escribe el PC.

1.
```
L      LW r2,a(r1)    IF ID EX M  WB
L+4    ADD r3,r2,r3      IF ID ID EX M  WB
L+8    ADD r3,r4,r3         IF IF ID EX M  WB
L+12   SUB r1,r1,#4               IF ID EX M  WB
L+16   BNEZ r1, L                    IF ID EX M  WB
L+20   SW z(r0), r3                      IF ID
L+24   ADD r3,r0,r0                        IF
L      LW r2,a(r1)                            IF ID EX M  WB
```

2.
```
L     LW r2,a(r1)    IF ID EX M  WB
L+4   ADD r3,r2,r3      IF ID ID EX M  WB
L+8   ADD r3,r4,r3         IF IF ID EX M  WB
L+12  SUB r1,r1,#4            IF ID EX M  WB
L+16  BNEZ r1, L                 IF ID EX M  WB
L+20  SW z(r0), r3                  IF IF IF
L     LW r2,a(r1)                           IF ID EX M  WB
```

3.
```
L     LW r2,a(r1)    IF ID EX M  WB
L+4   SUB r1,r1,#4      IF ID EX M  WB
L+8   ADD r3,r2,r3         IF ID ID EX M  WB
L+12  BNEZ r1, L              IF IF ID EX M  WB
L+16  ADD r3,r4,r3               IF ID ID EX M  WB
L     LW r2,a(r1)                   IF IF ID EX M  WB
```

***Solución:***

1. Riesgos de datos: cortocircuito; riesgos de control: *predict-not-taken*, escritura del PC en EX.
2. Riesgos de datos: cortocircuito; riesgos de control: inserción de ciclos de parada, escritura del PC en M.
3. Riesgos de datos: inserción de ciclos de parada; riesgos de control: salto retardado, escritura del PC en ID.

□

## Operadores multiciclo y gestión estática de instrucciones

## Operadores multiciclo

**Ejercicio 2.8.** Se dispone de un procesador MIPS con los siguientes operadores multiciclo:

- Sumador/Restador segmentado lineal. Lat= 2, IR= 1
- Multiplicador segmentado lineal. Lat= 3, IR= 1
- Divisor convencional. Lat= 4, IR= $\frac{1}{4}$

Los riesgos estructurales y de datos se detectan en la fase ID, insertando tantos ciclos de parada como sean necesarios. También se utilizan cortocircuitos.

Mostrar el diagrama de ejecución del siguiente fragmento de código, representando las etapas que atraviesan cada una de las instrucciones, utilizando la siguiente notación: `IF` fase de búsqueda, `ID` fase de decodificación, `EX` fase de ejecución monociclo, `A1,A2` fases de ejecución del sumador/restador, `M1,M2,M3` fases de ejecución del multiplicador, `D1,D2,D3,D4` fases de ejecución del divisor, `ME` fase de acceso a memoria y `WB` fase de escritura en registros. Las instrucciones multiciclo no realizan la fase `ME`.

```
l.d    f1, 0(r1)
div.d  f4, f0, f1
add.d  f2, f3, f4
l.d    f4, 8(r1)
mul.d  f3, f4, f2
l.d    f5, 16(r1)
```

***Solución:***

```
                  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16
l.d f1, 0(r1)     IF ID EX M  WB
div.D f4,f0,f1       IF ID ID D1 D2 D3 D4 WB
add.d f2,f3,f4          IF IF ID ID ID ID A1 A2 WB
l.d f4,8(r1)                  IF IF IF IF ID EX M  WB
mul.d f3,f4,f5                           IF ID ID M1 M2 M3 WB
l.d f5,16(r1)                               IF IF ID ID EX M  WB
```

□

**Ejercicio 2.9.**

Un procesador ejecuta el siguiente bucle que calcula $\vec{z} = A\vec{x} + B\vec{y}$:

```
loop:
      l.d f0,x(r10)
      l.d f1,y(r11)
      mul.d f4,f2,f0;  F2 contiene A.
      mul.d f5,f3,f1;  F3 contiene B.
      add.d f6,f4,f5
      daddi r14,r14,-1
      daddi r10,r10,8
      daddi r11,r11,8
      s.d f6,z(r12)
      daddi r12,r12,8
      bnez r14,loop
      <sgte>
```

El procesador cuenta con dos bancos de registros para almacenar datos enteros y de coma flotante. Además, para la ejecución de las operaciones en coma flotante, dispone de los siguientes operadores multiciclo:

- Un multiplicador segmentado con $T_{ev} = 5$ e $IR = 1$, con etapas denominadas M1, M2, etc.
- Un sumador no segmentado con $T_{ev} = 3$ e $IR = 1/3$, con etapas denominadas A1, A2, etc.

Las restantes instrucciones se ejecutan utilizando el pipeline clásico de 5 etapas (IF,ID,EX,ME,WB). Los riesgos de datos se resuelven con cortocircuitos e insertando ciclos de parada cuando es necesario. Los saltos condicionales utilizan la técnica *predict-not-taken*. La condición y el destino del salto se calculan durante la etapa ID del salto. Si el salto es efectivo, el PC se actualiza con la nueva dirección destino al final de esta etapa.

Se pide:

1. El diagrama instrucciones-tiempo de la primera iteración del bucle y la primera instrucción de la segunda iteración.
2. Asumiendo que todas las iteraciones son iguales, calcule el CPI medio del bucle para $n$ iteraciones.
3. Para acelerar la ejecución del bucle, se plantean dos opciones: a) sustituir el sumador no segmentado por otro segmentado con $Tev = 3$ e $IR = 1$, o bien b) sustituir el multiplicador segmentado por otro no segmentado con $Tev = 2$ e $IR = 1/2$. ¿Cuál de ambas opciones es la más adecuada para reducir el tiempo de ejecución del bucle? Razone la respuesta.

***Solución:***

1. Diagrama instrucciones-tiempo de la primera iteración del bucle:

```
                 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15 16 17 18 19 20 21
l.d F0,x(r10)    IF ID EX ME WB
l.d F1,y(r11)       IF ID EX ME WB
mult.d F4,F2,F0        IF ID M1 M2 M3 M4 M5 WB
mult.d F5,F3,F1           IF ID M1 M2 M3 M4 M5 WB
add.d F6,F4,F5               IF ID ID ID ID ID A1 A2 A3 WB
daddi r14,r14,-1                IF IF IF IF IF ID EX ME WB
daddi r10,r10,8                                IF ID EX ME WB
daddi r11,r11,8                                   IF ID EX ME WB
s.d F6,z(r12)                                        IF ID EX ME
daddi r12,r12,8                                         IF ID EX ME WB
bnez r14,loop                                              IF ID EX ME WB
<sgte>                                                        IF X
l.d F0,x(r10)                                                    IF ID EX ME WB
```

2. CPI medio para $n$ iteraciones:

   La segunda iteración comienza en el ciclo 17, luego una iteración dura 16 ciclos. Teniendo en cuenta que cada iteración ejecuta 11 instrucciones, el CPI medio resulta $CPI = 16/11 = 1{,}45$

3. Mejora del tiempo de ejecución:

   La opción a) no tendrá ningún efecto en el tiempo de ejecución puesto que el sumador no introduce ningún riesgo estructural en el bucle y la latencia del sumador alternativo es la misma.

   Opción b):

```
                 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15 16 17 18 19 20 21
l.d F0,x(r10)    IF ID EX ME WB
l.d F1,y(r11)       IF ID EX ME WB
mult.d F4,F2,F0        IF ID M1 M2 WB
mult.d F5,F3,F1           IF ID ID M1 M2 WB
add.d F6,F4,F5               IF IF ID ID A1 A2 A3 WB
daddi r14,r14,-1                IF IF ID EX ME WB
daddi r10,r10,8                    IF ID EX ME WB
daddi r11,r11,8                       IF ID EX ME WB
s.d F6,z(r12)                         IF ID EX ME
daddi r12,r12,8                          IF ID EX ME WB
bnez r14,loop                               IF ID EX ME WB
<sgte>                                         IF X
l.d F0,x(r10)                                     IF ID EX ME WB
```

   La opción b) mejora el tiempo de ejecución puesto que el multiplicador alternativo presenta menor latencia y permite que la suma se lanze al operador en el ciclo 9, lo que reduce los ciclos de parada en 2.

□

## Gestión estática de instrucciones

**Ejercicio 2.10.** Se dispone de un procesador compatible con el juego de instrucciones del MIPS. En dicho procesador se ejecuta la siguiente secuencia de código.

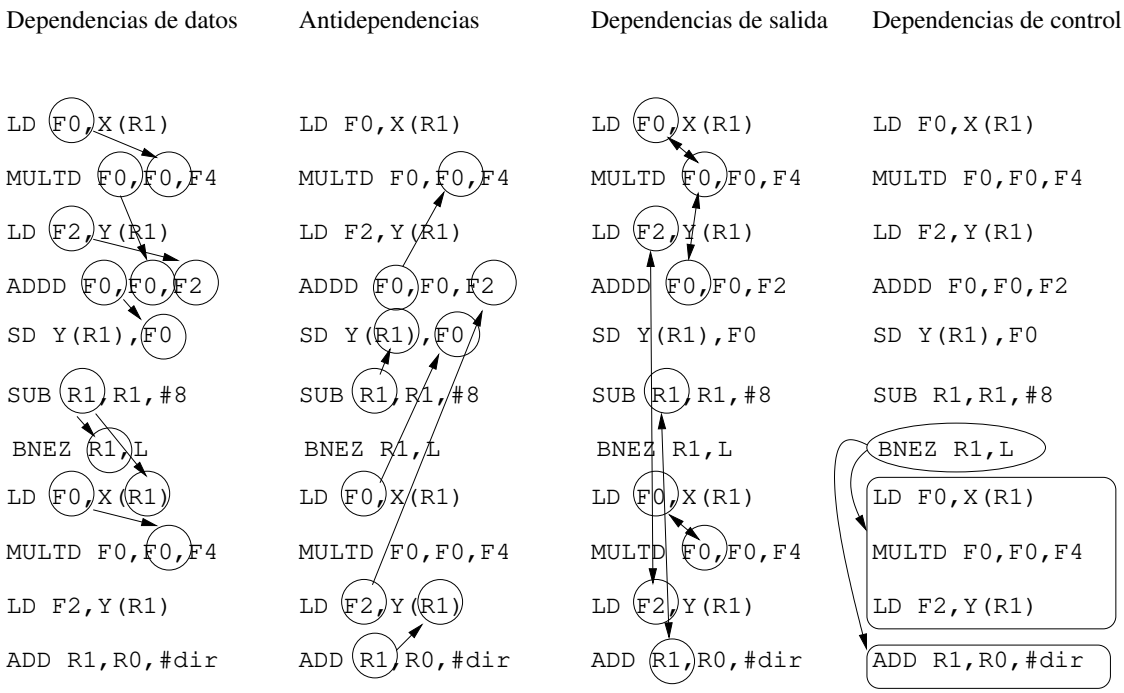
```
i1      L.D F0,X(R1)
i2      MULT.D F0,F0,F4
i3      L.D F2,Y(R1)
i4      ADD.D F0,F0,F2
i5      S.D F0,Y(R1)
i6      DSUB R1,R1,#8
i7      BNEZ R1,L
i8      L.D F0,X(R1)
```

```
i9        MULT.D F0,F0,F4
i10       L.D F2,Y(R1)
i11 L:    DADD R1,R0,#dir
```

Identifica al menos dos dependencias de cada tipo en el fragmento de código anterior.

***Solución:***

La siguiente figura muestra algunas de las dependencias existentes en la secuencia de código propuesta.




□

**Ejercicio 2.11.** Sea el siguiente código en ensamblador del MIPS:

```
loop: l.d f0, 0(r1)
      mul.d f0, f0, f10
      add.d f0, f0, f11
      s.d f0, 0(r1)
      dadd r1, r1, #8
      bne r1, r3, loop
```

Dicho código se pretende ejecutar sobre un MIPS en el que las instrucciones enteras atraviesan las siguientes etapas: IF (búsqueda de la instrucción), ID (decodificación de la instrucción, lectura de registros fuente y detección de riesgos), EX (ejecución), M (acceso a memoria) y WB (writeback), mientras que las de coma flotante son: IF, ID, En (ejecución en el operador multiciclo correspondiente) y WB. Los riesgos de datos se resuelven mediante cortocircuitos, insertando en ID los ciclos de parada necesarios y los de control mediante *predict-not taken*, escribiendo el PC en la fase ID.

El procesador funciona a 4 GHz y el CPI de las instrucciones aritméticas enteras es 1.

Las características de las unidades funcionales multiciclo son:

| Tipo de operador | Número | Latencia | Tipo |
|---|---|---|---|
| Multiplicación | 1 | 3 ciclos | Segmentada |
| Suma/resta | 1 | 3 ciclos | Segmentada |

1. Identifica una dependencia de datos, una antidependencia, una dependencia de salida y una dependencia de control en el código original.
2. Dibuja el diagrama instrucciones–tiempo de la primera iteración del bucle. Calcula el tiempo de ejecución para $n$ iteraciones.
3. Muestra el código obtenido tras modificarlo aplicando la técnica del *loop-unrolling*. Sin realizar nuevamente el diagrama instrucciones–tiempo, calcula el nuevo tiempo de ejecución y la aceleración (si la hubiese) con respecto al código original.

***Solución:***

1. Dependencias de datos identificadas
   - Dependencia de datos. Entre L.D **F0**, 0(R1) y MULT.D F0, **F0**, F10.
   - Antidependencia. MULT.D F0, **F0**, F10 y ADD.D **F0**, F0, F11
   - Dependencia de salida. MULT.D **F0**, F0, F10 y ADD.D **F0**, F0, F11
   - Dependencia de control. BNE R1, R3, loop y cualquier otra instrucción
2. Diagrama instrucciones–tiempo de la primera iteración del bucle para el procesador MIPS in-order.

```
                        1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17
loop: L.D F0, 0(R1)     IF ID EX M  WB
      MULT.D F0, F0, F10   IF ID ID M1 M2 M3 WB
      ADD.D F0, F0, F11       IF IF ID ID ID A1 A2 A3 WB
      S.D F0, 0(R1)                IF IF IF ID ID EX M
      DADD R1, R1, #8                       IF IF ID EX M  WB
      BNE R1, R3, loop                         IF ID ID EX M  WB
      <desc>                                      IF IF -- -- --
loop: L.D F0, 0(R1)                                     IF ID EX M  WB
```

   Notad que si la latencia de salto es de una instrucción, la dependencia de datos DADD/BNE (via R1) provoca un ciclo de parada para poder aplicar el cortocircuito M→ID en el ciclo 12 y el fallo en la predicción provoca la cancelación de una instrucción siguiente (que aparece indicada como `desc` en el diagrama).

   Cada iteración consume 12 ciclos de reloj, por lo que el tiempo de ejecución es $T(n) = 12n$
3. Código tras aplicar loop-unrolling.

   Como se insertan dos ciclos de parada para resolver la dependencia de datos entre MULT.D y ADD.D, hay que desenrollar 3 veces. El bucle resultante quedaría:

```
loop:   L.D F0, 0(R1)
        L.D F1, 8(R1)
        L.D F2, 16(R1)
        MULT.D F0, F0, F10
        MULT.D F1, F1, F10
        MULT.D F2, F2, F10
        ADD.D F0, F0, F11
        ADD.D F1, F1, F11
        ADD.D F2, F2, F11
        S.D F0, 0(R1)
        S.D F1, 8(R1)
        S.D F2, 16(R1)
        DADD R1, R1, #24
        BNE R1, R3, loop
```

Ahora ya no es necesario insertar ciclos de parada entre las instrucciones multiciclo. Se mantienen los dos ciclos de parada por el riesgo de datos DADD – BNE y por el riesgo de control. Como hay 14 instrucciones por iteración, cada iteración consume 16 ciclos. Teniendo en cuenta que se realizan un tercio de las iteraciones originales, el tiempo de ejecución será:

$$T(n) = \frac{16n}{3}$$

y la mejora sobre el código original es

$$S = \frac{12n}{16n/3} = 2{,}25 \text{ veces}$$

□

## Predicción dinámica de saltos

**Ejercicio 2.12.** Un procesador dispone de un predictor dinámico de saltos del tipo BTB (*Branch Target Buffer*) que obtiene su predicción en la fase de búsqueda de la instrucción. La dirección y condición de salto se calcula en la 3ª fase del ciclo de instruccion. La probabilidad de que un salto se encuentre en la tabla es del 80 % y de que se acierte en la prediccion es del 90 %. Los saltos son efectivos en el 60 % de los casos. Se pide:

1. Mostrar las instrucciones que se buscarían después de la de salto y sus fases para las opciones siguientes:

   - No hay una entrada en la tabla de predicción y el salto **no salta**.
   - No hay una entrada en la tabla de predicción y el salto **salta**.
   - El predictor predice que **no salta** y finalmente el salto **no salta**.
   - El predictor predice que **no salta** y finalmente el salto **sí salta**.
   - El predictor predice que **sí salta** y finalmente el salto **no salta**.
   - El predictor predice que **sí salta** y finalmente el salto **sí salta**.

   Las instrucciones se representarán como: **I.Salto**, instrucción de salto, **PC+i** (i= 1, 2, ...), instrucciones posteriores a la instrucción de salto, **Dest**, instrucción destino del salto y **Dest+i** (i= 1, 2, ...), instrucciones posteriores a la instrucción destino del salto. Las fases del ciclo de instrucción como **F1**, **F2**, etc...

2. Calcula el CPI medio de las instrucciones de salto.

3. Supóngase que se puede modificar el diseño del BTB de dos formas. La primera consiste en aumentar el número de entradas de la tabla, de forma que aloje el 90 % de los saltos ejecutados. La segunda es utilizar un predictor de dos bits de manera que se aumente la precisión de la predicción hasta el 95 %. ¿Cual de las dos es la que permite reducir los CPI de las instrucciones de salto?

***Solución:***

1. Mostrar las instrucciones que se buscarían después de la de salto y sus fases para las opciones siguientes:

- No hay una entrada en la tabla de predicción y el salto **no salta**.

| | p | | d,c | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| I. Salto | F1 | F2 | F3 | F4 | F5 | F6 | | | |
| PC + 1 | | F1 | F2 | F3 | F4 | F5 | F6 | | |
| PC + 2 | | | F1 | F2 | F3 | F4 | F5 | F6 | |
| PC + 3 | | | | F1 | F2 | F3 | F4 | F5 | F6 |

→ 0 ciclos de penalización.

- No hay una entrada en la tabla de predicción y el salto **salta**.

| | p | | d,c | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| I. Salto | F1 | F2 | F3 | F4 | F5 | F6 | | | |
| PC + 1 | | F1 | F2 | X | | | | | |
| PC + 2 | | | F1 | X | | | | | |
| Dest | | | | F1 | F2 | F3 | F4 | F5 | F6 |

→ 2 ciclos de penalización.

- El predictor predice que **no salta** y finalmente el salto **no salta**.

| | p | | d,c | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| I. Salto | F1 | F2 | F3 | F4 | F5 | F6 | | | |
| PC + 1 | | F1 | F2 | F3 | F4 | F5 | F6 | | |
| PC + 2 | | | F1 | F2 | F3 | F4 | F5 | F6 | |
| PC + 3 | | | | F1 | F2 | F3 | F4 | F5 | F6 |

→ 0 ciclos de penalización.

- El predictor predice que **no salta** y finalmente el salto **sí salta**.

| | p | | d,c | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| I. Salto | F1 | F2 | F3 | F4 | F5 | F6 | | | |
| PC + 1 | | F1 | F2 | X | | | | | |
| PC + 2 | | | F1 | X | | | | | |
| Dest | | | | F1 | F2 | F3 | F4 | F5 | F6 |

→ 2 ciclos de penalización.

- El predictor predice que **sí salta** y finalmente el salto **no salta**.

| | p | | d,c | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| I. Salto | F1 | F2 | F3 | F4 | F5 | F6 | | | |
| Dest | | F1 | F2 | X | | | | | |
| Dest + 1 | | | F1 | X | | | | | |
| PC + 1 | | | | F1 | F2 | F3 | F4 | F5 | F6 |

→ 2 ciclos de penalización.

- El predictor predice que **sí salta** y finalmente el salto **sí salta**.

| | p | | d,c | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| I. Salto | F1 | F2 | F3 | F4 | F5 | F6 | | | |
| Dest | | F1 | F2 | F3 | F4 | F5 | F6 | | |
| Dest + 1 | | | F1 | F2 | F3 | F4 | F5 | F6 | |
| Dest + 2 | | | | F1 | F2 | F3 | F4 | F5 | F6 |

→ 0 ciclos de penalización.

2. Calcula el CPI medio de las instrucciones de salto.

   La tabla siguiente muestra los casos posibles, así como la probabilidad de que se produzcan y su CPI (1 de la instrucción de salto más los posibles ciclos de parada:

| Caso | Probabilidad | CPI |
|---|---|---|
| 1 | 0.2*0.4 | 1 |
| 2 | 0.2*0.6 | 3 |
| 3 | 0.8*0.4*0.9 | 1 |
| 4 | 0.8*0.6*0.1 | 3 |
| 5 | 0.8*0.4*0.1 | 3 |
| 6 | 0.8*0.6*0.9 | 1 |
| $\overline{\text{CPI}}$ | | 1.4 |

Casos:

1. No hay una entrada en la tabla de predicción y el salto **no salta**.
2. No hay una entrada en la tabla de predicción y el salto **salta**.
3. El predictor predice que **no salta** y finalmente el salto **no salta**.
4. El predictor predice que **no salta** y finalmente el salto **sí salta**.
5. El predictor predice que **sí salta** y finalmente el salto **no salta**.
6. El predictor predice que **sí salta** y finalmente el salto **sí salta**.

El CPI medio se calcula obteniendo la media aritmética ponderada.

3. Supóngase que se puede modificar el diseño del BTB de dos formas. La primera consiste en aumentar el número de entradas de la tabla, de forma que aloje el 90 % de los saltos ejecutados. La segunda es utilizar un predictor de dos bits de manera que se aumente la precisión de la predicción hasta el 95 %. ¿Cual de las dos es la que permite reducir los CPI de las instrucciones de salto?

   Las tablas siguientes muestran los nuevo valores de la probabilidad de cada caso para cada una de las opciones:

   - Se aumenta el tamaño de la tabla de predicción.

| Caso | Probabilidad | CPI |
|---|---|---|
| 1 | 0.1*0.4 | 1 |
| 2 | 0.1*0.6 | 3 |
| 3 | 0.9*0.4*0.9 | 1 |
| 4 | 0.9*0.6*0.1 | 3 |
| 5 | 0.9*0.4*0.1 | 3 |
| 6 | 0.9*0.6*0.9 | 1 |
| $\overline{\text{CPI}}$ | | 1.3 |

   - Se aumenta la precisión en la predicción.

| Caso | Probabilidad | CPI |
|---|---|---|
| 1 | 0.2*0.4 | 1 |
| 2 | 0.2*0.6 | 3 |
| 3 | 0.8*0.4*0.95 | 1 |
| 4 | 0.8*0.6*0.05 | 3 |
| 5 | 0.8*0.4*0.05 | 3 |
| 6 | 0.8*0.6*0.95 | 1 |
| $\overline{\text{CPI}}$ | | 1.32 |

   La primera opción es ligeramente mejor que la segunda.

□

**Ejercicio 2.13.** Se dispone de un procesador con un juego de instrucciones similar al MIPS con una unidad de ejecución segmentada con las siguientes etapas:

- **IF** Búsqueda de la instrucción.
- **ID** Decodificación de la instrucción y lectura de registros.
- **ALU** Cálculo de la dirección de destino del salto y de la dirección de acceso a memoria.
- **MEM** Acceso a memoria.

- **EX1** Primera fase de ejecución y cálculo de la condición de salto.
- **EX2** Segunda fase de ejecución.
- **WB** Escritura en registros.

Se desea evaluar dos esquemas de predicción de saltos para su implementación en el procesador. Los predictores que se desean evaluar son: un *Branch Prediction Buffer* y un *Branch Target Buffer*, los cuales ofrecen su predicción al final de la etapa ID. Ambos mecanismos están implementados con 4 entradas en el *buffer* y utilizan un predictor de 2 bits, cuyo funcionamiento se ilustra en la figura:


Para la evaluación de los mecanismos de predicción se utiliza un programa de prueba, del que se muestra un fragmento a continuación:

```
Dirección     Instrucciones
...
0x03          add r1, r0, r0
      lfor:
...
0x05          beqz r1, lendif
...
      lendif:
...
      ldo:
0x09          sub r8, r8, r2
0x0A          slt r3, r2, #2
0x0B          seq r4, r1, r0
0x0C          and r5, r3, r4
0x0D          beqz r5, lbreak
```

```
Dirección     Instrucciones

...
0x10          add r2, r2, #1
0x11          slt r4, r2, #3
0x12          bnez r4, ldo
      lbreak:
...
0x15          add r1, r1, #1
0x16          slt r6, r1, #2
0x17          bnez r6, lfor
0x18          sw z(r0), r8
```

correspondencia directa -> hay algo

correspondencia asociativa

Inicialmente el *Branch Prediction Buffer* contiene todas las entradas en estado “D”, y el *Branch Target Buffer* tiene todas las entradas vacías. Cuando se añade una nueva entrada en el *Branch Target Buffer*, su estado será “A” si el salto ha sido efectivo y “D” si el salto no ha sido efectivo.

Las estadísticas finales de la ejecución del programa de prueba son las siguientes: el 15 % de las instrucciones son saltos condicionales, el 60 % de los saltos condicionales son efectivos (saltan), el *Branch Prediction Buffer* acierta en la predicción el 75 % de los casos, y el *Branch Target Buffer* acierta en la predicción el 90 %, incluyendo los casos en los que no hay una entrada en la tabla (los cuales se predicen como “no salta”).

Se solicita:

1. Realizar una traza de la ejecución del fragmento de código hasta que se complete la instrucción "`sw z(r0), r8`". Se deberá mostrar el contenido de las entradas de ambos predictores después de cada uno de los saltos ("`beqz r1, lendif`", "`beqz r5, lbreak`", "`bnez r4, ldo`" y "`bnez r6, lfor`"). Se deberán utilizar las etiquetas para anotar las direcciones de destino.

Instrucción de salto `beqz r1, lendif`. (r1 = 0)

**BPB**

| Índice | Estado |
|---|---|
| 00 | |
| 01 | |
| 10 | |
| 11 | |

**BTB**

| Índice | Dir. destino | Estado |
|---|---|---|
| | | |
| | | |
| | | |
| | | |

Instrucción de salto `beqz r5, lbreak`. (r5 = 1)

**BPB**

| Índice | Estado |
|---|---|
| 00 | |
| 01 | |
| 10 | |
| 11 | |

**BTB**

| Índice | Dir. destino | Estado |
|---|---|---|
| | | |
| | | |
| | | |
| | | |

Instrucción de salto `bnez r4, ldo`. (r4 = 1)

**BPB**

| Índice | Estado |
|---|---|
| 00 | |
| 01 | |
| 10 | |
| 11 | |

**BTB**

| Índice | Dir. destino | Estado |
|---|---|---|
| | | |
| | | |
| | | |
| | | |

Instrucción de salto `beqz r5, lbreak`. (r5 = 1)

**BPB**

| Índice | Estado |
|---|---|
| 00 | |
| 01 | |
| 10 | |
| 11 | |

**BTB**

| Índice | Dir. destino | Estado |
|---|---|---|
| | | |
| | | |
| | | |
| | | |

Instrucción de salto `bnez r4, ldo`. (r4 = 1)

**BPB**

| Índice | Estado |
|---|---|
| 00 | |
| 01 | |
| 10 | |
| 11 | |

**BTB**

| Índice | Dir. destino | Estado |
|---|---|---|
| | | |
| | | |
| | | |
| | | |

Instrucción de salto `beqz r5, lbreak`. (r5 = 0)

**BPB**

| Índice | Estado |
|---|---|
| 00 | |
| 01 | |
| 10 | |
| 11 | |

**BTB**

| Índice | Dir. destino | Estado |
|---|---|---|
| | | |
| | | |
| | | |
| | | |

Instrucción de salto `bnez r6, lfor`. (r6 = 1)

**BPB**

| Índice | Estado |
|---|---|
| 00 | |
| 01 | |
| 10 | |
| 11 | |

**BTB**

| Índice | Dir. destino | Estado |
|---|---|---|
| | | |
| | | |
| | | |
| | | |

Instrucción de salto `beqz r1, lendif`. (r1 = 1)

**BPB**

| Índice | Estado |
|---|---|
| 00 | |
| 01 | |
| 10 | |
| 11 | |

**BTB**

| Índice | Dir. destino | Estado |
|---|---|---|
| | | |
| | | |
| | | |
| | | |

Instrucción de salto `beqz r5, lbreak`. (r5 = 0)

**BPB**

| Índice | Estado |
|---|---|
| 00 | |
| 01 | |
| 10 | |
| 11 | |

**BTB**

| Índice | Dir. destino | Estado |
|---|---|---|
| | | |
| | | |
| | | |
| | | |

Instrucción de salto `bnez r6, lfor`. (r6 = 0)

**BPB**

| Índice | Estado |
|---|---|
| 00 | |
| 01 | |
| 10 | |
| 11 | |

**BTB**

| Índice | Dir. destino | Estado |
|---|---|---|
| | | |
| | | |
| | | |
| | | |

2. Analizar el comportamiento de la BTB cuando se equivoca en la predicción, indicando qué instrucciones son las que se ejecutan en los ciclos siguientes al salto. Se deberá indicar el número ciclos de ejecución perdidos. Las instrucciones canceladas se representarán con una **X** en el ciclo correspondiente. Las instrucciones siguientes al salto se representarán como **pc+1**, **pc+2**, ... y las instrucciones de destino del salto como **dest**, **dest+1**, etc.

3. Calcular el número medio de ciclos por instrucción (CPI) para el programa de prueba utilizando la BTB. Suponer que las instrucciones que no son saltos se ejecutan con CPI=1.

***Solución:***

1. Traza de la ejecución.

Instrucción de salto `0x05 (00000101) beqz r1, lendif`. (r1 = 0) **Salta**

**BPB**

| Índice | Estado |
|---|---|
| 00 | D |
| 01 | C |
| 10 | D |
| 11 | D |

**BTB**

| Índice | Dir. destino | Estado |
|---|---|---|
| 0x05 | lendif | A |
| | | |
| | | |
| | | |

Instrucción de salto `0x0D (00001101) beqz r5, lbreak`. (r5 = 1) **No salta**

se copia del ejer anterior y se modifica (C por D)

**BPB**

| Índice | Estado |
|---|---|
| 00 | D |
| 01 | D |
| 10 | D |
| 11 | D |

**BTB**

| Índice | Dir. destino | Estado |
|---|---|---|
| 0x05 | lendif | A |
| 0x0D | lbreak | D |
| | | |
| | | |

Se va añadiendo después del BPB

Instrucción de salto `0x12 (00010010) bnez r4, ldo`. (r4 = 1) **Salta**

**BPB**

| Índice | Estado |
|---|---|
| 00 | D |
| 01 | D |
| 10 | C |
| 11 | D |

**BTB**

| Índice | Dir. destino | Estado |
|---|---|---|
| 0x05 | lendif | A |
| 0x0D | lbreak | D |
| 0x12 | ldo | A |
| | | |

Instrucción de salto `0x0D (00001101) beqz r5, lbreak`. (r5 = 1) **No salta**

**BPB**

| Índice | Estado |
|---|---|
| 00 | D |
| 01 | D |
| 10 | C |
| 11 | D |

**BTB**

| Índice | Dir. destino | Estado |
|---|---|---|
| 0x05 | lendif | A |
| 0x0D | lbreak | D |
| 0x12 | ldo | A |
| | | |

Instrucción de salto `0x12 (00010010) bnez r4, ldo`. (r4 = 1) **Salta**

**BPB**

| Índice | Estado |
|---|---|
| 00 | D |
| 01 | D |
| 10 | A |
| 11 | D |

**BTB**

| Índice | Dir. destino | Estado |
|---|---|---|
| 0x05 | lendif | A |
| 0x0D | lbreak | D |
| 0x12 | ldo | A |
| | | |

Instrucción de salto `0x0D (00001101) beqz r5, lbreak`. (r5 = 0) **Salta**

**BPB**

| Índice | Estado |
|---|---|
| 00 | D |
| 01 | C |
| 10 | A |
| 11 | D |

**BTB**

| Índice | Dir. destino | Estado |
|---|---|---|
| 0x05 | lendif | A |
| 0x0D | lbreak | C |
| 0x12 | ldo | A |
| | | |

Instrucción de salto `0x17 (00010111) bnez r6, lfor`. (r6 = 1) **Salta**

**BPB**

| Índice | Estado |
|---|---|
| 00 | D |
| 01 | C |
| 10 | A |
| 11 | C |

**BTB**

| Índice | Dir. destino | Estado |
|---|---|---|
| 0x05 | lendif | A |
| 0x0D | lbreak | C |
| 0x12 | ldo | A |
| 0x17 | lfor | A |

Instrucción de salto `0x05 (00000101) beqz r1, lendif`. (r1 = 1) **No salta**

**BPB**

| Índice | Estado |
|---|---|
| 00 | D |
| 01 | D |
| 10 | A |
| 11 | C |

**BTB**

| Índice | Dir. destino | Estado |
|---|---|---|
| 0x05 | lendif | B |
| 0x0D | lbreak | C |
| 0x12 | ldo | A |
| 0x17 | lfor | A |

Instrucción de salto `0x0D (00001101) beqz r5, lbreak`. (r5 = 0) **Salta**

**BPB**

| Índice | Estado |
|---|---|
| 00 | D |
| 01 | C |
| 10 | A |
| 11 | C |

**BTB**

| Índice | Dir. destino | Estado |
|---|---|---|
| 0x05 | lendif | B |
| 0x0D | lbreak | A |
| 0x12 | ldo | A |
| 0x17 | lfor | A |

Instrucción de salto `0x17 (00010111) bnez r6, lfor`. (r6 = 0) **No salta**

**BPB**

| Índice | Estado |
|---|---|
| 00 | D |
| 01 | C |
| 10 | A |
| 11 | D |

**BTB**

| Índice | Dir. destino | Estado |
|---|---|---|
| 0x05 | lendif | B |
| 0x0D | lbreak | A |
| 0x12 | ldo | A |
| 0x17 | lfor | B |

2. Comportamiento de los predictores en caso de fallo en la predicción.

   - Para el BTB:
     Predice que **no** salta y **sí** salta.

| | | p | d | | c | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| I. Salto | IF | ID | ALU | ME | EX1 | EX2 | WB | | | | | |
| PC+1 | | IF | ID | ALU | ME | X | | | | | | |
| PC+2 | | | IF | ID | ALU | X | | | | | | |
| PC+3 | | | | IF | ID | X | | | | | | |
| PC+4 | | | | | IF | X | | | | | | |
| DEST | | | | | | IF | ID | ALU | ME | EX1 | EX2 | WB |

     Se producen **cuatro** ciclos de parada.
     Predice que **sí** salta y **no** salta.

| | | p | d | | c | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| I. Salto | IF | ID | ALU | ME | EX1 | EX2 | WB | | | | | |
| PC+1 | | IF | X | | | | | | | | | |
| DEST | | | IF | ID | ALU | X | | | | | | |
| DEST+1 | | | | IF | ID | X | | | | | | |
| DEST+2 | | | | | IF | X | | | | | | |
| PC+1 | | | | | | IF | ID | ALU | ME | EX1 | EX2 | WB |

     Se producen **cuatro** ciclos de parada.

3. CPI medio.

   Faltaría obtener la penalización en caso de acierto en la predicción. En caso de que el salto no sea efectivo, no hay penalización. En caso contrario, el BTB busca la instrucción correcta tras acceder al predictor (fase ID), introduciendo sólo un ciclo.

   - Para el BTB:

| Predicción | Condición | Probabilidad | Penalización |
|---|---|---|---|
| No | No | 0.9*0.4 | 0 |
| No | Si | 0.1*0.6 | 4 |
| Si | No | 0.1*0.4 | 4 |
| Si | Si | 0.9*0.6 | 1 |

     El número medio de ciclos de parada de penalización es: 0.94 ciclos, y el CPI medio de: $1 + 0{,}15 \cdot 0{,}94 = 1{,}14$ ciclos.

□

**Ejercicio 2.14.**

Considere el código de la función `ones`, que devuelve (en `$v0`) el número de bits a 1 contenidos en el argumento (`$a0`). El procedimiento es obtener el LSB del argumento (con `andi $t1,$a0,1`), incrementar la cuenta si LSB==1 (con `daddi $v0,$v0,1`) y desplazar el argumento una posición hacia la derecha (con `dsrl $a0,$a0,1`). Repitiendo 64 veces estas operaciones, se hace el trabajo especificado.

```
ones:  li $t0,64         # Número de iteraciones
       li $v0,0          # Cuenta inicial = 0
loop:  andi $t1,$a0,1    # $t1 = LSB de $a0
       beqz $t1,next     # Si (LSB!=0 )
       daddi $v0,$v0,1   #    $v0++ /* Cuenta LSB=1 */
```

```
next:  dsrl $a0,$a0,1   # Desplaza $a0 a la derecha 1 bit
       daddi $t0,$t0,-1 # Contador de iteraciones restantes
       bgtz $t0,loop    # Siguiente iteración
       jr $ra           # Retorno de función
```

Note que el código contiene tres instrucciones de salto:

`beqz $t1,next` es efectiva cada vez que LSB==0

`bgtz $t0,loop` cierra el bucle

`jr $ra` es incondicional y vuelve al punto desde donde se llamó a la función.

El procesador donde se ejecuta está segmentado en las 5 etapas habituales, escribe el PC en la etapa *EX* (es decir, la latencia de salto es de dos ciclos) y aplica predicción de salto. Note que no se produce ningún ciclo de parada por conflictos estructurales ni de datos.

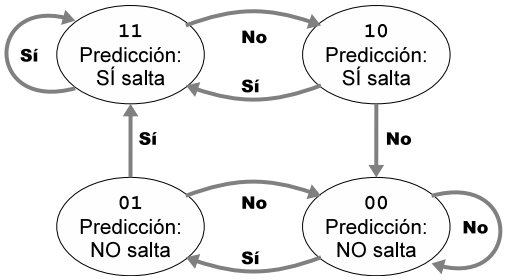
Calcule cuántas instrucciones ejecuta el procesador, cuántos ciclos de parada inserta y cuál es el tiempo de ejecución (en ciclos de reloj) de la función si:

1. El procesador aplica *predict-not taken* y `$a0`$=-1$ (0xFFF...FFF).
2. El procesador cuenta con un BTB sólo para los saltos condicionales, con un predictor de 1 bit, que permite predecir la dirección de salto en la etapa *IF* y aplica *predict-not taken* al retorno de función `jr $ra`. El programa llama a `ones` por primera vez con `$a0`$=-1$ (0xFFF...FFF)
3. El BTB del apartado 2 se ha perfeccionado con un predictor de 2 bits con histéresis. El programa llama a `ones` por primera vez con `$a0`=0x8080808080808080

***Solución:***

Llamar a `ones` provoca la ejecución de $3+5\times 64+n = 323+n$ instrucciones donde $n$ es el número de unos contenidos en el argumento.

1. *Predict-not taken*, `$a0`$=-1$
   El salto `beqz` nunca es efectivo. Por lo tanto,
   $I = 323+64 = 387$ instrucciones ejecutadas
   $P =$ 64 saltos efectivos $\times$ 2 paradas de penalización $= 128$ ciclos de parada
   $T = 387+128 = 515$ ciclos totales
2. BTB con predictor de 1 bit, `$a0`$=-1$
   Para `beqz` la predicción será siempre *not taken* (la primera iteración porque no hay historia, las demás porque el salto no será efectivo). No se produce ningún ciclo de parada.
   Para `bgtz` la primera predicción será *not taken* (no hay historia), pero las predicciones posteriores siempre serán *taken*. La predicción fallará la primera iteración y la última.
   El salto `jr $ra`, al quedar fuera del BTB, siempre sufrirá penalización
   $I = 323+64 = 387$ instrucciones ejecutadas
   $P =$ 3 predicciones no acertadas $\times$ 2 paradas de penalización $= 6$ ciclos de parada
   $T = 387+6 = 393$ ciclos totales
3. BTB con predictor de 2 bits, `$a0`=0x8080... El autómata del predictor de dos bits es este:




Con esta ruta de datos con BTB, si una instrucción de salto no tiene historia la predicción solo puede ser *not taken*. Cuando se ejecuta por primera vez, si el salto resulta no efectivo no tendrá penalización y la instrucción entra en el BTB con el estado 00; mientras que si resulta efectivo se aplica la penalización y la instrucción entra en el BTB con el estado 11. Seguidamente, y mientras el BTB mantiene el historico de la instrucción, el estado almacenado seguirá el autómata.

Con `beqz`, la predicción fallará en $b_0$ porque no tiene historia y entrará con el estado 11; en $b_1$ acertará y se quedará en el estado 11. De este estado saldrá solo en las 8 iteraciones en que $b_i = 1$ ($i = 3, 7, 11, \ldots$) para pasar al estado 10, pero siempre vuelve de nuevo al estado 11 en la iteración siguiente, donde $b_i = 0$. Por tanto, una vez insertada la instrucción de salto en el BTB, la predicción será siempre *taken* y fallara en los 8 casos donde $b_i = 1$. Total: 9 penalizaciones

Con `bgtz`, la predicción fallará en $b_0$ por falta de historia, despues acertará siempre gracias a la regularidad del comportamiento y volverá a fallar a la salida del bucle. Total: 2 penalizaciones

Con la penalización de `jr`, habrá un total de 12.

$I =$ $323 + 8 = 331$ instrucciones ejecutadas

$P =$ $12$ predicciones no acertadas $\times$ $2$ paradas de penalización $= 24$ ciclos de parada

$T =$ $331 + 24 = 355$ ciclos totales

□