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

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.

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 dominio público 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:

```
eti1:   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?

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.

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, ...` Instrucción que lee `R1` ...	`LOAD R1, ...` Instrucción que no lee `R1` 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`
o
`LH R1,x`
	`LW R1,x'`
	`SRL R1,R1,#pos`
	`AND R1,R1,#masc`

`SB R1,x`
o
`SH R1,x`
	`LW R2,x'`
	`SLL R1,R1,#pos`
	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.

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?

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
```

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, 4(R1)
MULT.D F3, F4, F2
L.D    F5, 8(R1)
```

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)
      mult.d F4,F2,F0;  F2 contiene A.
      mult.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.

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.

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

```
loop:   L.D F0, 0(R1)
        MULT.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.

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...

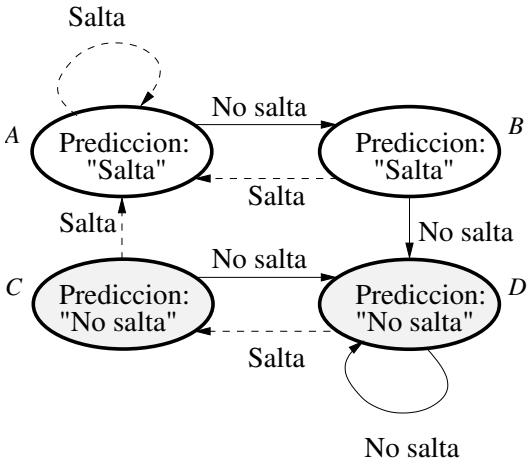
2. Calcula el CPI medio de las instrucciones de salto.

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?

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              Dirección     Instrucciones
...
0x03          add r1, r0, r0             ...
      lfor:                              0x10          add r2, r2, #1
...                                      0x11          slt r4, r2, #3
0x05          beqz r1, lendif            0x12          bnez r4, ldo
...                                            lbreak:
```

```
      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
```

```
...
0x15          add r1, r1, #1
0x16          slt r6, r1, #2
0x17          bnez r6, lfor
0x18          sw z(r0), r8
```

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.

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