ESTRUCTURA DE COMPUTADORES

PRÁCTICA Nº 2

RIESGOS DE DATOS Y SALTOS RETARDADOS

EN WINMIPS64

Estas prácticas se basan en las desarrolladas por el Departamento

de Informática Aplica de la Universidad Politécnica de Madrid (http://www.dia.eui.upm.es)

INTRODUCCIÓN

Este documento contiene el enunciado de la práctica sobre algunos de los aspectos que influyen en el tiempo de ejecución de un programa en el simulador WinMIPS64.

OBJETIVO

El objetivo perseguido es que el alumno compruebe, en un cauce segmentado, el impacto que tiene la ejecución de las diferentes instrucciones que forman un programa. En esta práctica se mostrará la influencia de las técnicas de anticipación de datos y saltos retardados, así como la influencia de la optimización del código para lograr mejores resultados en los tiempos de ejecución de un programa.

DESARROLLO DE LA PRÁCTICA

Para desarrollar la práctica se partirá de un programa de ordenación de un vector por el método de la burbuja. El código del programa en lenguaje C es:

```
int main()
{
 Burbuja();
}

void Burbuja() {
  int i;
  int j;
  int n=10;
  int A[10]={1,3,5,7,9,2,4,6,8,0};
  int temp;

  for (i = 0; i < n-1; i++)
    for (j = n-1; j > i; j--)
      if (A[j] < A[j-1]) {
        temp = A[j];
        A[j] = A[j-1];
        A[j-1] = temp;
      }
} // Burbuja
```

Una posible traducción del programa a ensamblador MIPS64 es (fichero `P2.s`):

```
; Ordenacion creciente de un vector por el metodo de la
; Burbuja

.data
i:    .word32  0        ; indice en sentido creciente
j:    .word32  0        ; indice en sentido decreciente
n:    .word32  0        ; tamaño del vector
A:    .word32  1,3,5,7,9,2,4,6,8,0  ; array a ordenar
temp: .word32  0        ; var. para intercambio de elementos
```

```
.text
MAIN:
                        jal Burbuja                          Bloque 0
                        nop
                        halt
```

```
; Funcion Burbuja
Burbuja:                                                     Bloque 1
                        daddi R2,R0,10    ; inic. variable n
                        sw R2,n(R0)       ; guardar nuevo valor de n
```

```
                        lw R14,i(R0)      ;
                        daddi R15,R0,0    ; for ( i = 0;     Bloque 2
                        sw R15,i(R0)      ;
```

```
FOR1:                   lw R16,i(R0)      ;                  Bloque 3
                        lw R17,n(R0)      ;
                        daddi R18,R0,1    ; i < n-1;
                        dsub R19,R17,R18  ;
                        slt R20,R16,R19   ;
                        beqz R20,ENDFOR1  ;
                        nop               ;
```

```
                        lw R21,j(R0)      ;                  Bloque 4
                        lw R22,n(R0)      ;
                        daddi R23,R0,1    ; for ( j = n-1;
                        dsub R24,R22,R23  ;
                        sw R24,j(R0)      ;
```

```
FOR2:                   lw R25,j(R0)      ;                  Bloque 5
                        lw R26,i(R0)      ;
                        slt R27,R26,R25   ; j > i;
                        beqz R27,ENDFOR2  ;
                        nop               ;
```

```
IF1:                    lw R28, A(R0)     ;                  Bloque 6
                        lw R29, j(R0)     ;
                        daddi R1,R0,4     ; A [j]
                        dmul R2,R1,R29    ;
                        lw R30,A(R2)      ;
```

```
                        lw R3,A(R0)       ;                  Bloque 7
                        lw R4,j(R0)       ;
                        daddi R5,R0,1     ;
                        dsub R6,R4,R5     ; A [j-1]
                        daddi R8,R0,4     ;
                        dmul R9,R8,R6     ;
                        lw R7,A(R9)
```

```
                        slt R10,R30,R7    ;                  Bloque 8
                        beqz R10, IFELSE1 ; if (A[j] < A[j-1])
                        nop               ;
```

```
                        lw R11,temp(R0)   ;                  Bloque 9
                        lw R12,A(R0)      ;
                        lw R13,j(R0)      ;
                        daddi R15,R0,4    ;
                        dmul R16,R15,R13  ;  temp = A[j]
```

```
                        lw R14,A(R16)     ;
                        sw R14,temp(R0)   ;
```

Bloque 10

```
                        lw R17,A(R0)      ;
                        lw R18,j(R0)      ;
                        daddi R20,R0,4    ; A[j]
                        dmul R21,R20,R18  ;
                        lw R19,A(R21)     ;
```

Bloque 11

```
                        lw R22,A(R0)      ;
                        lw R23,j(R0)      ;
                        daddi R24,R0,1    ; A [j-1]
                        dsub R25,R23,R24  ;
                        daddi R27,R0,4    ;
                        dmul R28,R27,R25  ;
```

Bloque 12

```
                        lw R26,A(R28)     ; A [j] = A[j-1]
                        sw R26,A(R21)     ;
```

Bloque 13

```
                        lw R29,A(R0)      ;
                        lw R30,j(R0)      ;
                        daddi R1,R0,1     ; A [j-1]
                        dsub R3,R30,R1    ;
                        daddi R5,R0,4     ;
                        dmul R6,R5,R3     ;
```

Bloque 14

```
                        lw R4,A(R6)       ;
                        lw R7,temp(R0)    ;
                        sw R7,A(R6)       ; A[j-1] = temp
                        j IFEND1          ;
                        nop
```

Bloque 15

```
IFELSE1:
IFEND1:                 lw R11,j(R0)      ;
                        daddi R11,R11,-1  ; j--
                        sw R11,j(R0)      ;
                        j FOR2            ;
                        nop
```

Bloque 16

```
ENDFOR2:
                        lw R13,i(R0)      ;
                        daddi R13,R13,1   ; i++
                        sw R13,i(R0)      ;
                        j FOR1
                        nop
```

Bloque 17

```
ENDFOR1:
FIN_Burbuja:            jr R31            ; Return to MAIN
                        nop
```

El programa consta de dos funciones MAIN y Burbuja. La primera corresponde con el programa principal conteniendo la llamada a la función (subrutina) Burbuja.

DESCRIPCIÓN DETALLADA DEL PROGRAMA

El programa está dividido en 18 bloques (de 0 a 17):

- Bloque 0 — Programa principal. Consta de una llamada a la función Burbuja. La instrucción `jal` guarda en el registro R31 la dirección de retorno.
- Bloque 1 — Inicialización de la variable `n` al valor 10.
- Bloque 2 — Inicialización de la variable `i` de la sentencia `for` más externa a 0. La instrucción `daddi R15,R0,0` asigna el valor 0 al registro R15 sumando dos operandos con valor 0 (el registro R0 siempre tiene el valor 0)
- Bloque 3 — Evalúa la condición del bucle `for` interno (`i < n-1`). Primero lee los valores de las variables `i` y `n`, y después realiza la resta, para finalizar con la comprobación. El salto, instrucción `beqz`, no se toma mientras la condición sea cierta.
- Bloque 4 — Inicialización de la variable `j` del for interno al valor `n-1`. Tras la lectura de memoria de las variables `j` y `n`, se evalúa la expresión `n-1`, finalizando con la escritura en memoria del valor de `j`.
- Bloque 5 — Evalúa la condición (`j>i`) del bucle interno. Mientras la condición sea cierta el salto no es tomado.
- Bloque 6 — Acceso al elemento `A[j]`. En primer lugar se leen la dirección de comienzo del vector `A` y el valor de la variable `j`. Los elementos del vector A tienen un tamaño 4 octetos (declaración `.word32`). La dirección de memoria que contiene un elemento del vector se calcula cómo la suma de la dirección base del vector más el producto del nº del elemento por su tamaño (en este caso 4 octetos)
- Bloque 7 — Acceso al elemento `A[j-1]`. El código es idéntico al del bloque anterior, pero al accederse al elemento `j-1` es necesario resta uno al valor de la variable `j`.
- Bloque 8 — Evalúa la condición `(A[j] < A[j-1])` de la sentencia if. El registro R7 contiene el valor A[j-1] y R30 el de A[30]. El salto se toma cuando la condición es falsa.
- Bloque 9 — Corresponde a la sentencia `temp = A[j]`. El acceso al elemento `A[j]` se realiza de forma análoga a los anteriores y la

asignación se realiza mediante una escritura en la dirección de memoria reservada a la variable `temp`.

- Bloques 10 y 11 Similares a los bloques 6 y 7.
- Bloque 12 Sentencia `A [j] = A[j-1]`. El registro R28 contiene el desplazamiento del elemento j dentro del vector. En la instrucción de lectura (`lw`) la dirección del elemento se forma sumando la dirección base del vector `A` y el valor del registro R28 (direccionamiento indexado con desplazamiento). El registro R21 contiene el desplazamiento del elemento `j-1` del vector. La asignación es simplemente la lectura de un elemento y su posterior escritura en otra posición del vector.
- Bloque 13 Idéntico al Bloque 11
- Bloque 14 Sentencia `A[j-1] = temp` El desplazamiento del elemento del vector A está en registro R6. La asignación termina con una escritura en la posición de memoria ocupada por el element j-1. La instrucción `j IFEND1` da por finalizada la sentencia if transfiriendo el control al Bloque 15
- Bloque 15 Actualización de la variable `j` del bucle `for` interno y salto a la condición del bucle `for` (Bloque 8)
- Bloque 16 Actualización de la variable `i` del bucle `for` externo y salto a la condición del bucle `for` externo (Bloque 3)
- Bloque 17 Fin de la función `Burbuja` mediante un salto a la dirección de retorno (función MAIN) almacenada en el registro R31.

PASO A PASO

El desarrollo de la práctica consta de una serie de pasos que se detallan a continuación.

Paso 1

Tras descargar el material de la práctica y arrancar el simulador WinMIPS64 se deberá configurar éste para que no aplique ni Anticipación (*Data forwarding*), ni salto retardado (*Delay Slot*). Estas dos características pueden quitarse (o ponerse) mediante las opciones ***Enable forwarding*** y ***Enable Delay Slot*** del menú ***Configure*** del simulador.

A continuación y mediante la opción ***Open*** del menú ***File*** se pasará a cargar el programa ensamblador del fichero `P2.s`, comprobando que código y variables se encuentran en memoria.

Una vez cargado se deberá ejecutar el programa en su totalidad (utilizar la opción ***Run to*** del menú ***Execute*** o bien la tecla de función ***F4***) obteniendo las siguientes informaciones de la ventana ***Statistics***:

- Ciclos empleados en la ejecución del programa
- Instrucciones ejecutadas
- CPI (*Cycles Per Instruction*)
- Paradas del cauce debidas a riesgos:
  - RAW
  - WAW
  - WAR
  - Estructurales
- Paradas del cauce debidas a:
  - Saltos tomados
  - Fallos en la predicción de salto

Paso 2

Comprobar en la memoria si el programa ha ordenado correctamente los números del vector. Indicar en qué direcciones de memoria están cada uno de los elementos, empezando por el elemento 0.

Paso 3

Recargar de nuevo el programa (opción ***Reload*** del menú ***File***).
Ejecutando paso a paso (opción ***Single Cycle*** del menú ***Execute*** o tecla de función ***F7***) detectar en qué instrucciones se producen las tres primeras paradas del cauce debidas a la presencia de riesgos RAW (antes del ciclo 20).

	Instrucción 1	Instrucción 2	Ciclos de parada
Riesgo 1			
Riesgo 2			
Riesgo 3			

De acuerdo con esta información, ¿el HW implementa anticipación en el banco de registros?

Paso 4

En este paso se va a comprobar el impacto del mecanismo de anticipación en el tiempo de ejecución del programa.

Recargar de nuevo el programa y activar la opción **Enable Forwarding** del menú **Configure**. Ejecutar el programa en su totalidad (tecla F4) y anotar las informaciones proporcionadas por el simulador en la ventana de Statistics (mismos datos que los obtenidos en el Paso 1). Obtener con ellos el índice de mejora de los parámetros siguientes:

	Valor	Factor de mejora
Ciclos empleados en la ejecución del programa		
Instrucciones ejecutadas		
CPI (*Cycles Per Instruction*)		
Paradas del cauce debidas a riesgos RAW		

Paso 5

Recargar de nuevo el programa (opción Reload del menú File) y Enable Forwarding activado.

Ejecutar paso a paso hasta el ciclo 20, ¿qué diferencias encuentras con los resultados obtenidos en el Paso 3? ¿Por qué sigue existiendo el riego RAW entre las instrucciones slt R20,R16,R19 y beqz R20,ENDFOR1 (ciclos 13 y 14)?

Paso 6

Otro aspecto que influye en el tiempo de ejecución de los programas es la optimización. Al programa P2.s que se ha venido usando se le van a aplicar varias optimizaciones cómo almacenamiento de las variables en registros, con el consiguiente ahorro en accesos a memoria o sustituyendo ciertas instrucciones aritméticas por otras equivalentes con menor tiempo de ejecución.

Si se toma el Bloque 6 del programa se observa que pueden aplicarse dos tipos de optimización:

```
IF1:                    lw R28, A(R0)      ;
                        lw R29, j(R0)      ;
                        daddi R1,R0,4      ; A [j]
                        dmul R2,R1,R29     ;
                        lw R30,A(R2)       ;
```

1. Suponiendo que las variables, excepto el vector `A`, están almacenadas en registros, se podría eliminar un acceso (instrucción `lw`).

2. La instrucción de multiplicación `dmul` se utiliza para calcular la dirección de memoria en la que se encuentra almacenado el elemento `j` del vector A. Si observamos, en la instrucción siempre se multiplica el índice `j` (registro `R29`) por el valor 4 almacenado en el registro R1. Una multiplicación de un potencia de dos puede sustituirse por un desplazamiento de bits a la izquierda, en este caso, al multiplicar por 4 es necesario desplazar 2 ($\log_2 4$) bits a la izquierda. En la arquitectura MIP64 podemos utilizar la instrucción `dsll` para realizar este desplazamiento.

El bloque de código con las dos optimizaciones quedaría:

```
IF1:                    dsll R2,R9,2      ;
                        lw R30,A(R2)      ;
```

Suponiendo que la variable j está almacenada en el registro `R9`. También se ha eliminado la lectura (instrucción `lw R28,A(R0)`) por ser una operación innecesaria al no utilizarse el operando destino (`R8`) en ninguna otra instrucción.

Estas optimizaciones sobre el programa anterior se en encuentran en el fichero de nombre `P2-1.s`. Para realizar la optimización se han utilizado los siguientes registros:

Variable	Registro
i	R8
j	R9
n	R10
temp	R11

En primer lugar se han sustituido las variables por sus correspondientes registros, eliminando así los accesos de lectura y escritura. A continuación se ha sustituido la instrucción de multiplicación por la de desplazamiento. El programa admite una tercera optimización consistente en sustituir la instrucción de resta por una instrucción de suma, un ejemplo de este tercer tipo es:

```
daddi R24,R0,1    ; A [j-1]
dsub  R25,R23,R24
```

Estas dos instrucciones pueden sustituirse por: `daddi R25,R23,-1`

Cargar el fichero `P2-1.s` y ejecutarlo en el simulador con la opción **Enable Forwarding**.

Volver a calcular los índices de mejora del programa (`P2-1.s`) con respecto del programa `P2.s` con la opción de anticipación activa.

	Valor	Factor de mejora
Ciclos empleados en la ejecución del programa		
Instrucciones ejecutadas		
CPI (*Cycles Per Instruction*)		
Paradas del cauce debidas a riesgos RAW		

Paso 7

Si se observa la ejecución del programa `P2-1.s` se ve que las instrucciones `NOP` nunca llegan a ejecutarse por completo siendo abortadas tras la etapa de Fetch (F). Como podrás imaginar este comportamiento tiene que ver con la gestión de las instrucciones de salto por parte del procesador.

Se pide cargar de nuevo el fichero P2-1.s[1] activando las funcionalidades de salto retardado y data forwarding, mediante el menú **Configure**.

Explica porqué en las ventana **Statistics** la información **Branch Taken Stalls** es cero al habilitar los saltos retardados.

Explica, también, qué instrucciones son las causantes de los riesgos RAW que aparecen en el programa.

Paso 8

Al activar los saltos retardados se observa que las instrucciones (`NOP` en nuestro caso) que van a continuación de los saltos se ejecutan siempre, se tome o no el salto. Una de las tareas que tienen asignadas los compiladores es colocar una instrucción "útil" después de una de salto o poner NOP si no encuentra una candidata.

Revisa el código del fichero `P2-1.s` y, si puedes, sustituye las instrucciones `NOP` por otras instrucciones presentes en el fichero, de tal forma que no cambies la semántica del programa. Guarda el nuevo programa en el fichero `P2-2.s`

Para cada instrucción de salto del programa indica con qué instrucción has rellenado el hueco de retardo.

---

[1] *Si no has terminado el apartado anterior pide al profesor encargado de prácticas que te proporcione una versión del fichero P2-1.s*

Por último, indica para este nuevo programa las informaciones obtenidas en la ventana **Statistics** del simulador (con las opciones data *forwarding* y *delay slot* activadas).

HOJA DE RESPUESTAS	
NOMBRE Y APELLIDOS	

Paso 1

RESULTADOS de P2.s	
Ciclos empleados en la ejecución del programa	
Instrucciones ejecutadas	
CPI (Cycles Per Instruction)	
Paradas del Cauce debidas a Riesgos	
RAW	
WAW	
WAR	
Estructurales	
Paradas del cauce debidas a:	
Saltos Tomados	
Fallos en la predicción de salto	

Paso 2

Indicar en qué direcciones de memoria están cada uno de los elementos, empezando por el elemento 0.

Paso 3

	Instrucción 1	Instrucción 2	Ciclos de parada
Riesgo 1			
Riesgo 2			
Riesgo 3			

¿El HW implementa anticipación en el banco de registros?

Paso 4

Resultados de P2.s	Valor	Factor de mejora
Ciclos empleados en la ejecución del programa		
Instrucciones ejecutadas		
CPI (*Cycles Per Instruction*)		
Paradas del cauce debidas a riesgos RAW		

Paso 5

¿Qué diferencias encuentras con los resultados obtenidos en el Paso 3?

¿Por qué sigue existiendo el riesgo RAW entre las instrucciones `slt R20,R16,R19` *y* `beqz R20,ENDFOR1` *(ciclos 13 y 14)?*

Explica a qué es debido el incremento de los riesgos estructurales. Muestra (copia la pantalla) la situación del cauce para el primer riesgo estructural que encuentres ¿cuál es la causa concreta del riesgo estructural?

Paso 6

	Valor	Factor de mejora
Ciclos empleados en la ejecución del programa		
Instrucciones ejecutadas		
CPI (*Cycles Per Instruction*)		
Paradas del cauce debidas a riesgos RAW		

¿Por qué han desaparecido los riesgos estructurales presentes en el programa `P2.s`*?*

Paso 7

Explica porqué en las ventana ***Statistics*** *la información* ***Branch Taken Stalls*** *es cero al habilitar los saltos retardados.*

Explica, también, qué instrucciones son las causantes de los riesgos RAW que aparecen en el programa.

Paso 8

Para cada instrucción de salto del programa indica con qué instrucción has rellenado el hueco de retardo.

RESULTADOS de P2-2.s (Con forwarding y delay slot activados)	
Ciclos empleados en la ejecución del programa	
Instrucciones ejecutadas	
CPI (Cycles Per Instruction)	
Paradas del Cauce debidas a Riesgos	
RAW	
WAW	
WAR	
Estructurales	
Paradas del cauce debidas a:	
Saltos Tomados	
Fallos en la predicción de salto	