Escuela Universitaria de Informática
Ingeniería de Computadores
Curso 2022/2023

ARQUITECTURA DE COMPUTADORES

PRÁCTICA Nº 3

DESENROLLADO DE BUCLES

INTRODUCCIÓN

Este documento contiene el enunciado de la práctica de desenrollado de bucles sobre el simulador WinMIPS64.

OBJETIVO

El objetivo de la práctica es que el alumno aplique la técnica del desenrollado de bucles, conociendo sus ventajas y también sus limitaciones.

EL DESENROLLADO DE BUCLES

El desenrollado de bucles es una técnica que permite obtener un mayor rendimiento en el tiempo de ejecución de los programas. El mayor rendimiento se obtiene mediante el aprovechamiento del paralelismo que se puede obtener en los bucles de acceso a los elementos de un array.

Esta técnica, generalmente, la aplica el compilador realizando varias copias del código del bucle, además de utilizar algún registro adicional del procesador (renombrado de registros) para eliminar ciertos riesgos que puedan aparecer en el nuevo código.

El desenrollado no siempre es fácil aplicarlo debido a que en ciertas situaciones el número de veces que se ejecuta un bucle no es un valor estático sino que depende de una variable cuyo contenido no es conocido en tiempo de compilación. Otra dificultad añadida puede ser la no disponibilidad de registros para evitar ciertas dependencias de datos (RAW).

UN EJEMPLO

Para comprender mejor en qué consiste la técnica, partiremos del siguiente programa de ejemplo:

```
; ejecutar sin "delay slot"
.data
vector: .word32  1,2,3,4,5,6,7,8

.code
        daddi   r1,r0,0
        daddi   r2,r0,32        ; Tamaño del vector en bytes
loop:   lw      r10,vector(r1)
        daddi   r10,r10,4       ; Tamaño de cada elemento
        sw      r10,0(r1)
        daddi   r1,r1,4
        bne     r1,r2,loop
        halt
```

El propósito de este programa es incrementar en 4 cada uno de los elementos del vector.

```
FOR I := N TO 1 DO
  A[I] := A[I]+4;
END
```

En el programa en ensamblador existen tres dependencias de datos (RAW) que no permiten ninguna reordenación del código (por parte del compilador) para evitar las paradas que aparecerán en el cauce durante su ejecución.

En una primera aproximación se va a desenrollar el bucle en cuatro copias, quedando de la forma siguiente:

```
    daddi  r1,r0,0
    daddi  r2,r0,32           ; Tamaño del vector en bytes
LOOP:
    lw     r10,vector(r1)     ; Leer elemento vector
    daddi  r10,r10,4          ; Sumar 4 al elemento
    sw     r10,vector(r1)     ; Escribir nuevo valor

    lw     r11,vector+4(r1)   ; 2ª copia
    daddi  r11,r11,4          ;
    sw     r11,vector+4(r1)   ;

    lw     r12,vector+8(r1)   ; 3ª copia
    daddi  r12,r12,4          ;
    sw     r12,vector+8(r1)   ;

    lw     r13,vector+12(r1)  ; 4ª copia
    daddi  r13,r13,4          ;
    sw     r13,vector+12(r1)  ;

    daddi  r1,r1,16           ; Actualizar índice
    bne    r1,r2,LOOP         ; Fin de vector?
    halt
```

Para evitar dependencias de datos se han empleado para cada copia registros distintos al original del bucle. También se han modificado los desplazamientos en las instrucciones de `load` y `store` para permitir el acceso a los elementos anteriores al indicado por la variable índice del bucle (registro r1). Así mismo, la actualización de r1 se ha modificado, sustituyendo la constante 4 por 16 con el fin de reflejar el procesamiento de los cuatro elementos del array (cada elemento ocupa 4 octetos de memoria).

Las dependencias de datos entre cada copia se mantienen (instrucciones `lw`, `daddi` y `sw`), para evitarlas puede reorganizarse el código de la forma siguiente:

```
            daddi r1,r0,0
            daddi r2,r0,32
LOOP:       lw    r10,0(r1)
            lw    r11,vector+4(r1)
            lw    r12,vector+8(r1)
            lw    r13,vector+12(r1)
            daddi r10,r10,4
            daddi r11,r11,4
            daddi r12,r12,4
            daddi r13,r13,4
            sw    r10,vector(r1)
            sw    r11,vector+4(r1)
            sw    r12,vector+8(r1)
            sw    r13,vector+12(r1)
            daddi r1,r1,vector+16
            bne   r1,r0,LOOP

FIN:        halt
```

En este código la única dependencia de datos corresponde a las dos últimas instrucciones del bucle (registro r1).

Con el desenrollado que se ha realizado el bucle necesitará ejecutarse únicamente la cuarta parte de veces que el original.

DESARROLLO DE LA PRÁCTICA

Para realizar la práctica se partirá del programa siguiente en alto nivel:

```
main () {

 int i;
 int suma;
 int array[240];

 for (i = 0; i < 240; i++) {
   array[i] = 2;
 }

 suma = 0;
 for (i = 0; i < 240; i++) {
   suma += array[i]*array[i];
 }
}
```

La traducción a lenguaje ensamblador MIPS64 es la siguiente. El fichero con el código está disponible con el enunciado de la práctica (p3.s):

```
.data
i:      .word32  0
suma:   .word32  0
array:  .space 960
.code
main:
; Inicializar los elementos del array con el valor 2
        lw      r2,i(r0)
        daddi   r3,r0,240
        daddi   r6,r0,2
FOR_LOOP_1:
        slt     r4,r2,r3
        beqz    r4,END_FOR_LOOP1
        dsll    r5,r2,2
        sw      r6,array(r5)
        daddi   r2,r2,1
        j       FOR_LOOP_1
        nop
END_FOR_LOOP1:

; Recorre el bucle suma += array[i]^2
        daddi   r2,r0,0      ; i=0 en r2
        daddi   r3,r0,240    ; tamaño=240 en r3
        lw      r5,suma(r0)  ; suma en r5
FOR_LOOP_2:
        slt     r4,r2,r3
        beqz    r4,END_FOR_LOOP2

        dsll    r6,r2,2
        lw      r7,array(r6)
```

```
            dmul    r8,r7,r7            ; a[i]*a[i]
            dadd    r5,r5,r8

            daddi   r2,r2,1
            j       FOR_LOOP_2
            nop
END_FOR_LOOP2:
            sw      r5,suma(r0)
            halt
```

En la declaración de datos (`.data`) para la variable `array` se han reservado 960 octetos correspondientes a los 240 elementos de tipo `word32` (4 octetos).
Para la ejecución del programa se ha supuesto que el simulador tiene activos los mecanismos de *Data Forwarding* y *Delay Slot* (menú *Configure*). Obsérvese que la instrucción que aparece a continuación de una de salto se ejecuta siempre, por eso en algunas partes del programa la instrucción `NOP` sucede a una de salto.

El código a partir de la etiqueta `FOR_LOOP_2` es el encargado de realizar la suma de los cuadrados de cada uno de los elementos del `array`. La variable índice `i` se almacena en el registro r2 y `suma` en r5. Para acceder a cada uno de los elementos del `array` es necesario multiplicar el valor de la variable `i` por 4 ya que, como se ha citado anteriormente, cada elemento del vector ocupa 4 octetos, para llevar a cabo la multiplicación se utiliza la instrucción `dsll` con un desplazamiento de 2 bits.

En cada iteración del bucle se utilizan tres registros del procesador:

- **r6** Para realizar el cálculo de la dirección de comienzo de cada elemento del array.
- **r7** Para almacenar el contenido del elemento i-esimo del array.
- **r8** Para almacenar el cuadrado del elemento i-esimo del array.

Paso 1

Ejecutar el programa anterior en el simulador obteniendo las informaciones siguientes:

RESULTADOS	
Ciclos	6021
Instrucciones	3854
CPI	1.562
Riesgos RAW	2162
Riesgos Estructurales	240
Tamaño del código	96 bytes
Resultado de la suma	960

Paso 2

Desenrollar el bucle FOR_LOOP_2 en dos copias partiendo del programa original, guardando el nuevo código en un fichero de nombre `p3-2.s`

Para realizar la segunda copia del bucle se aconseja:

- Si para la primera copia de utilizaron los registros **r6** ,**r7** y **r8**, para las operaciones de la segunda copia se recomienda utilizar los registros **r9**, **r10** y **r11.**
- incrementar en uno la variable índice (`i`) antes de la segunda copia, evitando de esta forma tener que modificar la condición de finalización del bucle.
- leer y actualizar en memoria la variable de índice (`i`) y la suma acumulada (`suma`) una sola vez por iteración.

Ejecutar el nuevo programa e indicar los nuevos resultados.
¿A qué son debidos los riesgos estructurales que aparecen? ¿Pueden eliminarse?

RESULTADOS	
Ciclos	5421
Instrucciones	3375
CPI	1.606
Riesgos RAW	2042
Riesgos Estructurales	240
Tamaño del código	116 bytes
Resultado de la suma	960

Paso 3

Reorganizar el código (fichero de nombre `p3.2opt.s`) del Paso 2 para eliminar, en la medida de lo posible, las dependencias de tipo RAW del código. Ejecutar de nuevo el programa e indicar los resultados en la tabla.

RESULTADOS	
Ciclos	4101
Instrucciones	3135
CPI	1.308
Riesgos RAW	722
Riesgos Estructurales	240
Tamaño del código	112 bytes
Resultado de la suma	960

Paso 4

Desenrollar el bucle FOR_LOOP_2 en cuatro copias (fichero `p3-4.s`). Utilizar los registros **r12**, **r13** y **r14** para la tercera copia. Y **r15**, **r16** y **r17** para la cuarta copia.
Rellenar la tabla de resultados tras la ejecución.

RESULTADOS	
Ciclos	5121
Instrucciones	3135
CPI	1.634
Riesgos RAW	1982
Riesgos Estructurales	240
Tamaño del código	156 bytes
Resultado de la suma	960

Paso 5

Reorganizar el código del paso 4 eliminando el mayor número de dependencias RAW. Guardar el resultado en el fichero `p3-4opt.s` y rellenar la tabla de resultados.

RESULTADOS	
Ciclos	3501
Instrucciones	2955
CPI	1.185
Riesgos RAW	302
Riesgos Estructurales	240
Tamaño del código	144 bytes
Resultado de la suma	960

Paso 6

Desenrollar el bucle FOR_LOOP_2 en seis copias (guardando el código en el fichero `p3-6.s`).

RESULTADOS	
Ciclos	5021
Instrucciones	3055
CPI	1.644
Riesgos RAW	1962
Riesgos Estructurales	240
Tamaño del código	196 bytes
Resultado de la suma	960

Paso 7

Reorganizar el código del paso 6 tratando de eliminar las dependencias RAW. Guardar el resultado en el fichero `p3-6opt.s`

RESULTADOS	
Ciclos	3261
Instrucciones	2855
CPI	1142
Riesgos RAW	282
Riesgos Estructurales	120
Tamaño del código	176 bytes
Resultado de la suma	960

Paso 8

Indicar la aceleración obtenida para cada versión del programa, con respecto al original (`p3.s`).

Versión de programa	Aceleración (*SpeedUp*)
P3-2.s	1.110680686
P3-2opt.s	1.468178493
P3-4.s	1.175746924
P3-4opt.s	1.719794344
P3-6.s	1.199163513
P3-6opt.s	1.846366145