# Arquitectura de Computadoras Práctica 3. Simulador MIPS

## Objetivos

Visualizar, a través del simulador WinMIPS, el funcionamiento de una arquitectura pipeline relativamente simple.

Identificar el impacto que tienen los saltos y saltos condicionales sobre la ejecución del procesador.

## Introducción

Con un editor convencional (por ejemplo, notepad), edite el siguiente programa y guárdelo como prueba1.s

; Arquitectura de computadoras

; Programa de demostración.

.data

i: .word32 0

j: .word32 0

.text

daddi R2,R0,0;

daddi r3, R0, 0; //

daddi r5,R0,10 ; //Comparacion r5=10

WHIL: slt R6, R2, R5

beqz R6, ENDW

daddi r3, r3, 5 //J+5

sw R3, j(r0) //Giardamos en Memoria J

daddi r2,r2,1 //I+1

sw r2,i(r0) //Guardamos en Memoria I

j WHIL //

ENDW: nop

halt

Este programa corresponde al programa en C:

int main() {

int i= 0; int j= 0;

while ( i < 10 ) {

j = j+5;

i = i +1;

}

}

Abra una ventana de comandos y compruebe que el programa es sintácticamente correcto mediante la instrucción:

C:> asm prueba1.s

Ejecute el simulador. Desde el menú file, cargue el programa prueba1.s y córralo paso a paso (F7)

## Primera parte

1. ¿Qué registros se utilizan para almacenar las variables i, j?

Se utilizan los registros R2 y R3 respectivamente.

1. ¿Para qué se utiliza la instrucción slt R6, R2, R5? ¿Qué ocurre si se intercambian los últimos dos registros de la instrucción?

Toma compara el registro 2 (R2) con el Resgistro 5 (R5). Cuando el valor en R2 es mayor a R5 el valor de R6 será 1. En la siguiente instrucción(beqz R6, ENDW) nos dice que cuando R6 vale 1 continuaremos con el orden de las instrucciones. Cuando R6 es igual a 0 entonces haremos un branch a la etiqueta ENDW. En ENDW terminaremos de iterar.

Si intercambiamos lugares R5 tiene actualmente el valor de 10 y R2 el valor de 0 por lo tanto R6 adquiere el valor de 0 y se termina el ciclo y el programa.

1. ¿Qué valores tienen i y j al final de la ejecución del programa?
2. Modifique el programa anterior para que las variables y el código se almacenen a partir de las direcciones 100 y 200 de sus respectivos segmentos de datos y código

## Segunda parte

Escriba un programa que almacene las primeras 10 potencias de 2 en un arreglo de 10 elementos. El tamaño de los elementos es de 32 bits.

1. Utilice un algoritmo para calcular las potencias de 2 mediante multiplicación
2. Ahora utilice un algoritmo basado en corrimientos a la izquierda
3. Compare los tiempos de ejecución para cada caso. ¿Cuál es la principal fuente de la diferencia en los tiempos de ejecución¡

## Tercera parte – Loop unrolling

Considere el siguiente programa:

LOOP: lw r10,0(r1); Leer un elemento de un vector

daddi r10,r10,4 ; Sumar 4 al elemento

sw r10,0(r1); Escribir el nuevo valor

daddi r1,r1,-4 ; Actualizar la var. índice

bne r1,r0,LOOP ; Fin de vector?

El cual corresponde al código:

FOR I := N DOWNTO 1 DO

A[I] := A[I]+4;

END

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 pipeline durante su ejecución.

En una primera aproximación se va a desenrollar el bucle en cuatro copias, quedando

de la forma siguiente:

LOOP: lw r10,0(r1); Leer elemento vector

daddi r10,r10,4 ; Sumar 4 al elemento

sw r10,0(r1); Escribir nuevo valor

lw r11,-4(r1); 2ª copia

daddi r11,r11,4 ;

sw r11,-4(r1);

lw r12,-8(r1); 3ª copia

daddi r12,r11,4 ;

sw r12,-8(r1);

lw r13,-12(r1); 4ª copia

daddi r13,r11,4 ;

sw r13,-12(r1);

daddi r1,r1,-16 ; Actualizar índice

bne r1,r0,LOOP ; Fin de vector?

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 arreglo (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:

LOOP: lw r10,0(r1);

lw r11,-4(r1);

daddi r10,r10,4 ;

daddi r11,r11,4 ;

lw r12,-8(r1);

lw r13,-12(r1);

daddi r12,r11,4;

daddi r13,r11,4;

sw r10,0(r1);

sw r11,-4(r1);

sw r12,-8(r1);

sw r13,-12(r1);

daddi r1,r1,-16;

bne r1,r0,LOOP ;

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.

1. Ejecute las tres versiones del código y verifique su ejecución.
2. Compare las estadísticas obtenidas en cada caso: Ciclos, instrucciones, CPI, riesgos RAW, Riesgos estructurales, Tamaño del código