Procesadores Comerciales (3º Grado Informática)

ArqTecCom - Dpto. Informática e Ingeniería de Sistemas

CPS - UniZar

Práctica 2: Simulador de un procesador con tres caminos de ejecución

Los objetivos de esta práctica son: i) entender y modelar un procesador segmentado con varios caminos de ejecución y el scoreboard que lo controla, ii) conocer en detalle el papel del Reorder Buffer y modelarlo para nuestro procesador, iii) estudiar el comportamiento y cuantificar las prestaciones del procesador.

1. Descripción del simulador de partida

En esta práctica se estudia y posteriormente se modifica un simulador sencillo de un procesador segmentado con tres caminos de ejecución. El control se centraliza en la etapa de Decodificación y se gestiona mediante scoreboard. El simulador consta de varios ficheros en C que se encuentran en:

```
pilgor:/export/home/alumnos/a01/3caminos/
```

Debes copiar todos los ficheros puesto que algunos han sufrido pequeñas modificaciones respecto a la versión de la practica 1.

El fichero cpu.c contiene el modelo del procesador, que incluye los tres caminos de ejecución, el control de riesgos estructurales en las unidades funcionales (float), y el control de riesgos estructurales en la escritura del banco de registros. A continuación mostramos el bucle principal que simula un ciclo de ejecución:

```
while (quedan_inst){

    /* etapa Decode */
    /*********************************************************************/
    /* deteccion de riesgos y lectura en BR, implementa SCOREBOARD
        si no hay problemas la instruccion pasa a A */
    /* Tres caminos:
    Integer: nunca para, alu de un ciclo
    Memory: nunca para, @ y M de un ciclo cada una
            (se puede hacer multiciclo segmentada facilmente)
    Float: 5 ciclos no segmentada */

    wBR=wBR>>1;   /* ocupacion de puerto de escritura en BR */
    /* tras este despl. el bit de menor peso de wBR representa ciclo actual*/

    if (ciclos_parada_AF>0) { /* ocupacion UF FLOAT */
        ciclos_parada_AF--;
    }

    carga_D = 1;

    /* riesgos RAW: verifica si los reg. fuentes estan preparados */
    if ( 0  /*  deteccion de riesgo RAW, a rellenar  */  ){
              craw++;
              carga_D = 0;
    }
```

```
else
{
    /* RIESGO ESTRUCTURAL EN FLOAT */
    if (etapa_Din.co == FLOAT && ciclos_parada_AF>0){
        cfloat++;
        carga_D = 0;
    }
    else {
        /* RIESGO ESTRUCTURAL EN BR */
        if (etapa_Din.rd > 0 && etapa_Din.co != STORE){
            aux=1<<(latenciasWR[etapa_Din.co]+1);
            if(wBR & aux) {
                cwBR++;
                carga_D = 0;
            }
        }
    }
}

if(carga_D){
    /* lanzo la inst. por el camino correspondiente */
    if (etapa_Din.co == FLOAT) ciclos_parada_AF=5; /* latencia 5 */

    /* si escribe un reg. ocupo wBR y marco ciclo en que se produce rd*/
    if (etapa_Din.rd > 0 && etapa_Din.co != STORE) {
             wBR=wBR | aux;
    }
}


/* etapa Busqueda */
/*****************************************************************/
....
```

2. Modelado de la detección y parada en riesgos RAW y WAW

Se trata de añadir al scoreboard (etapa D) la detección de riesgos RAW/WAW y el bloqueo de esta etapa hasta que se solucionen dichos riesgos. Para ayudar en esta tarea, ofrecemos la definición de la estructura de datos necesaria:

```
unsigned long int disp_reg[max_reg];
/* guarda el ciclo en que se hizo la última actualización de cada reg o el
ciclo en el que se hará si hay alguna instrucción en vuelo que lo va a modi-
ficar */
/* si es mayor que tiempo, el registro está pendiente de escribir
donde tiempo es la variable que contiene el ciclo actual */
```

3. Modelado del Reorder Buffer (ROB)

En esta tarea vais a añadir el ROB al modelo de nuestro procesador. Para ayudar en esta tarea, ofrecemos la definición de la estructura de datos y las funciones necesarias para manipularla en el fichero rob.c:

```
#include "cabecera.h"
#define MAX_ROB 8
typedef struct ROBentry {
    unsigned long terminada;
    IREG instr; /* solo se usa para chivato */
    char rd;
} ROBentry_t;
ROBentry_t ROB[MAX_ROB];
int ROBfirst=0, ROBlast=0, ROBocu=0;

int ROBadd(IREG IR)
{
    int i;
    if (ROBocu == MAX_ROB) return -1;
    ROBocu++;
    ROB[ROBlast].rd = IR.rd;
    ROB[ROBlast].instr = IR;
    ROB[ROBlast].terminada = 0;
    i = ROBlast;
    ROBlast++;
    if (ROBlast == MAX_ROB) ROBlast = 0;
    return i;
}

void ROBejecuta(int entrada, unsigned long int ciclo)
{
    ROB[entrada].terminada = ciclo;
}

char ROBjubila(unsigned long int ciclo)
{
    if (ROBocu == 0 || ROB[ROBfirst].terminada >= ciclo) return 0;
    ROBocu--;
    ROBfirst++;
    if (ROBfirst == MAX_ROB) ROBfirst = 0;
    return 1;
}
```

4. Experimentación y análisis de resultados

Para el benchmark matrizc, medir tiempos de ejecución para los distintos modelos. Descomponer los ciclos de detención en función de quien provocó su pérdida: bloqueo por RAW después de load, por RAW después de float, por riesgo estructural en la UF de float, ...