Parcial 2 - Arquitectura de computadoras 2024

25/10/24

**10 (Diez)**

Apellido, Nombre: Achával Berzero, Tomás

**Ejercicio 1**

Considere la ejecución del siguiente segmento de código LEGv8 para i > 0, donde i → X3, y X6 contiene la dirección base del arreglo A[ ] del tipo uint32_t.

```
        LSL X9, X3, #2
        ADD X9, X6, X9
loop:   SUB X11, X6, X9
        CBZ X11, end
        LDUR X10, [X9, #0]      A[6],A[4],A[2]
        ADD X0, x10, X0
        SUBI x9, x9, #8
        B loop
end:    STUR X0, [X9, #0]       A[0]
```

El procesador tiene una CACHE exclusiva para **DATOS** de correspondencia **DIRECTA** de 64Kbyte y 4 palabras ($w_p$) por línea, sobre un procesador de 32bits, CPI = 1, que resuelve todos los data y control hazard sin necesidad de stalls, tiene una memoria principal de 4G palabras ($w_m$) de 1 byte cada una. Considerar que el resto de los registros están inicializados en 0 y la CACHÉ vacía al inicio de la ejecución del segmento.

**a)** Completar cada casillero con el número de bits de cada campo del formato de dirección de memoria principal:

| Tag | Index | Word | Offset byte(*) |
|---|---|---|---|
| 16 | 12 | 2 | 2 |

* Llenar con 0 en el caso que no corresponda

**b)** Cuántos MISS de CACHE se produjeron en la ejecución del fragmento de código si i = 6 y la dirección base del arreglo A es 0x00F192B8?. Suponga que los accesos a memoria de escritura y lectura tienen **IDÉNTICO comportamiento** para la CACHE.

**Respuesta:** 3 **Miss**

**c)** Completar como queda la memoria CACHE al finalizar la ejecución del segmento para las condiciones del punto b). Llenar una fila por cada MISS producido. Se completó la primer línea sólo con fines ilustrativos.

| Index (hex) | Tag (hex) | V | Data $W_3$ | $W_2$ | $W_1$ | $W_0$ |
|---|---|---|---|---|---|---|
| 0xC0CA | 0xCAFE | 1 | A[11] | A[10] | A[9] | A[8] |
| 0x92D | 0x00F1 | 1 | A[9] | A[8] | A[7] | A[6] |
| 0x92C | 0x00F1 | 1 | A[5] | A[4] | A[3] | A[2] |
| 0x92B | 0x00F1 | 1 | A[1] | A[0] | ?? | ?? |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |

**Ejercicio 2**

En la figura se muestra un diagrama simplificado del procesador LEGv8 2-*issue*. Considerando la forma de funcionamiento de la implementación completa (con multiplexores, HDU, unidad de *forwarding*, los caminos de forwarding incluso a la etapa ID para la resolución de saltos, etc.) y asumiendo que los saltos son perfectamente predichos (en el ciclo de clock posterior al *fetch* de un salto se hace *fetch* del paquete correcto: PC+8 o PC+offset, según corresponda, no hay penalidad), responder:

a) Dado el siguiente fragmento de código en assembler LEGv8 (más abajo), analizar las dependencias de datos y completar con el **primer** caso que encuentre de cada una de las siguientes dependencias:

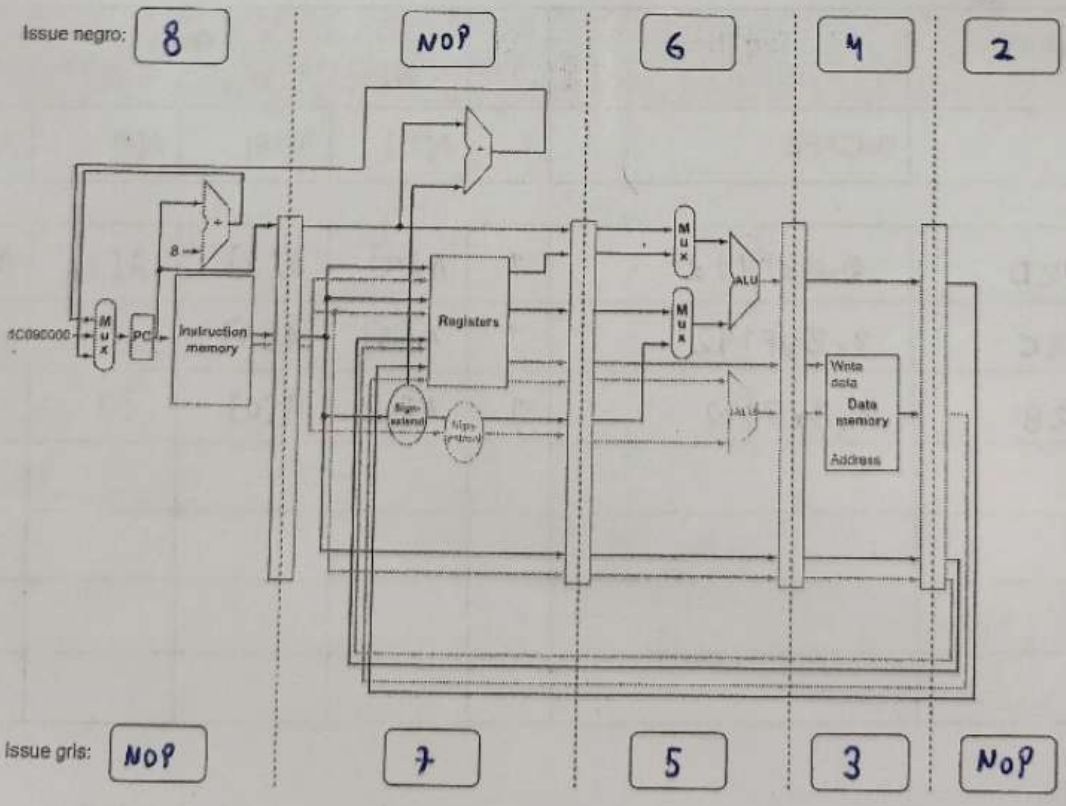
| Tipo de dependencia | Números de instrucciones | Operando en conflicto |
|---|---|---|
| RAW | 3,7 | x2 |
| RAW condicional | 1,6 | x4 |
| WAW | 5,8 | x3 |

b) Mostrar en la tabla cómo organizaría los *issue packets* para ejecutar el programa en la menor cantidad posible de ciclos de clock **sin alterar el orden de las instrucciones** (cada instrucción sólo puede agruparse con la inmediata anterior, la inmediata posterior o una nop). El compilador asume toda la responsabilidad de insertar instrucciones nop para que el código se ejecute sin necesidad de generación de *stalls*. Usar los números correspondientes para referirse a las instrucciones del código y "nop" para las instrucciones agregadas.

```
1>        ORRI X4, XZR, #0x8
2> loop: CBZ X0,end
3>        LDUR X2, [X1, #0]
4>        SUBI X0, X0, #1
5>        LDUR X3, [X1, #8]
6>        ADD X1, X1, X4
7>        STUR X2, [X1, #0]
8>        ADD X3, X3, X1
9>        CBZ XZR, loop
10> end:
```

| Issue negro | Issue gris |
|---|---|
| 1 | NOP |
| 2 | NOP |
| 4 | 3 |
| 6 | 5 |
| NOP | 7 |
| 8 | NOP |
| 9 | NOP |
| | |
| | |

c) Completar en los recuadros vacíos del diagrama del procesador qué instrucción se está ejecutando en cada etapa del *pipeline*, tanto en el *issue* negro como en el gris, en el ciclo de clock número **6**, considerando que antes de comenzar la ejecución del programa X0=2.



## Ejercicio 3

Considerando un microprocesador out-of-order execution implementado mediante el algoritmo de Tomasulo que cuenta con las características del hardware descritas en la tabla y que ejecuta el siguiente código en assembler, muestre el contenido de las tablas de estado de las Reservation stations y registros (en la próxima página) para el 4to clock de ejecución (inclusive y considerando que se hace fetch en el clk 0):

```
1> L1: SUB X0, X0, #8
2>     LDUR X10, [X0, #0]
3>       LDUR X11, [X1, #0]
4>     ADD X1, X0, #8
5>     ADD X12, X10, X12
6>     MULT X12, X10, X12
7>     STUR X12, [X0, #0]
8>     STUR X12, [X1, #0]
9>     CBNZ X0, L1
10>     ORR X0, X1, X2
```

| Hardware |
|---|
| *El salto se predice correctamente* |
| Issue = 4 instrucciones |
| Load = 4 RS / 1 clk |
| Store = 4 RS / 1 clk |
| ALU entera = 4 RS / 1 clk |
| ALU punto flotante = 4 RS / 2 clk |
| Multiplicación ~~punto flotante~~ ENTERA = 2 RS / 3 clk |

## Ejercicio 4

Considerar que un procesador que cuenta con un predictor global de 8 bits ejecuta el siguiente código en C y compilado como se muestra a la derecha:

```
A[3]={3,7,4};

for (i = 0; i<3; i++) { ~~//X0<-i~~

    if (A[i] < 5) { //X1<-A[i]; X2<-5;
    ...
    }

}
```

| Dirección | | |
|---|---|---|
| 0x14 | L: | ... |
| ... | | ... |
| ... | | cmp X1, X2 |
| 0x34 | | b.lt e_if // TAKEN si A[i]<5 |
| ... | | ... |
| 0x44 | | cbnz X0, L // ASUMO NT |

LUEGO DE "PROCESAR" A[2]

Completar la siguiente tabla con todas las posiciones de la PHT que se modifican, considerar que todas las posiciones de memoria están inicializadas en 0 y el GR=0x34

| Dirección (16) | Contenido (6) |
|---|---|
| 0011 0120 | 01 |
| 0110 1001 | 01 |
| 1101 0011 | 00 |
| 1010 0110 | 01 |
| 0100 1101 | 01 |
| 1001 1011 | 00 |
| | |
| | |
| | |

IF A[0] < 5
FIN A[0], CONTINÚO LOOP
IF A[1] < 5
FIN A[1], CONTINÚO LOOP
IF A[2] < 5
FIN A[2], FIN LOOP

(el profe no leyó esto)

* RES INST 7 = [x0] ACTUALIZADO POR INSTRUCCIÓN 7

| Name | Reservation stations | | | | | | |
|---|---|---|---|---|---|---|---|
| | Busy | Op | Vj | Vk | Qj | Qk | A |
| LOAD 1 | | LOAR | ~~RES INST 7~~ X0 | - | 0 | - | ~~RES INST 7~~ + 0 |
| LOAD 2 | | LOAR | [x1] | - | 0 | - | [x1] + 0 |
| STORE 1 | | STOR | ~~RES INST 1~~ X0 | - | 0 | MULT 1 | 0 |
| STORE 2 | | STOR | - | - | ALU 2 | MULT 1 | 0 |
| ALU 1 | | | | | | | |
| ALU 2 | ✓ | ADD | RES INST 7 | #8 | 0 | 0 | |
| ALU 3 | | ADD | - | [x12] | LOAD 1 | 0 | |
| ALU 4 | | | | | | | |
| MULT 1 | | MULT | - | - | LOAD 1 | ALU 3 | |
| BRANCH | ✓ | CBNZ | RES INST. 1 | -8 | 0 | 0 | |
| | | | | | | | |
| | | | | | | | |
| | | | | | | | |
| | | | | | | | |

[x0 + #0]

| Register Status | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | X0 | X1 | X2 | X3 | X4 | X5 | X6 | X7 | X8 | X9 | X10 | X11 | X12 |
| Qi | | ALU2 | | | | | | | | | LOAD 1 | LOAD 2 | MULT 1 |