Parcial 2 - Arquitectura de computadoras 2023

2 (dos)

Nombre: Bonelis Gastón Tomas

Ejercicio 1

Considere para un procesador de arquitectura LEGv8 (Address bus de 64 bits) una CACHE L1 exclusiva para INSTRUCCIONES de 2K bytes de datos, de correspondencia DIRECTA, de 2 palabras (cada palabra es 1 instrucción de 32 bits) por bloque. La siguiente lista corresponde a DIRECCIONES de acceso a la memoria de instrucciones (PC) para la ejecución de un lazo (solo se muestran 16 bits de los 64 bits para abreviar):

```
1> 0x00…00 00 02 F4      <loop: asm_instruction>
2> 0x00…00 00 02 F8
3> 0x00…00 00 1A FC
4> 0x00…00 00 1B 00
5> 0x00…00 00 1B 04      <cbz loop>
```

a) Calcular el formato de memoria principal para la CACHE dada y completar la siguiente tabla en formato HEXADECIMAL para 2 iteraciones del loop:

Acc.	Hit / Miss	Tag (55) bits	Index (8) bits	Word (1) bits
1>	Miss	0x1	0x7A	0x0
2>	hit	0x1	0x7C	0x0
3>	miss	0x1	0x7E	0x0
4>	hit	0xD	0x80	0x0
5>	hit	0xD	0x82	0x0
1>	hit	0x1	0x7A	0x0
2>	hit	0x1	0x7C	0x0
3>	hit	0x1	0x7E	0x0
4>	hit	0x1	0x80	0x0
5>	hit	0x1	0x82	0x0

b) Suponiendo que el procesador opera con el pipeline en régimen a 1GHz, el tiempo de acceso a CACHE es de 1 ciclo y el Miss penalty es de 50 ciclos, calcular el AMAT para las condiciones del punto a).

AMAT: 11 ns seg.

c) Ahora considere que puede cambiar el tamaño del bloque y la CACHE pasa a ser asociativa por conjunto de N-vías. Determine el tamaño del bloque y el número de vías ÓPTIMOS para minimizar la ocurrencia de MISS (se considera el número óptimo aquel arriba del cual no se generan beneficios adicionales).

Tamaño de bloque: 16 bytes.

Número de vías: 2 vías.

d) Calcular el AMAT de 2 iteraciones del lazo para las condiciones del punto c).

Tiempo de ejecución total: 6 ns seg.

AMAT

Ejercicio 2

Dado un procesador de arquitectura LEGv8 2-issue, que predice los saltos perfectamente (de modo que los hazard de control son manejados por hardware), con una modificación que permite ejecutar dos instrucciones aritméticas/lógicas juntas (es decir, una instrucción

en un paquete puede ser de cualquier tipo -*IssueA*-, pero la otra debe ser una instrucción aritmética o lógica -*IssueB*-). Para el siguiente fragmento de código LEGv8, donde X0=A (dirección base del arreglo A, de elementos de 1 Byte), X1=N (cantidad de elementos del arreglo), X2=1 y los demás registros se encuentran inicializados con "0":

```
1>          ADD X3, X0, X1
2>    loop:LDURB X9,[X0, #0]
3>          SUB X12, X9, X2
4>          LDURB X10,[X3, #0]
5>          STURB X12,[X3, #0]
6>          SUBI X3, X3, #1
7>          ADDI X0, X0, #1
8>          ADD X11, X10, X2
9>          STURB X11,[X0, #-1]
10>         SUBS X11, X0, X3
11>         B.GE end    //>= signado
12>         ADDI X2, X2, #2
13>         B loop
       end:
```

N° instrucción Issue A	N° instrucción Issue B
2>	12>
NOP	NOP
4>	3>
5>	6>
7>	8>
9>	10
11>	12>
13>	

No Pueden ir juntos (loop:)

+/− a) Analizar en el código las dependencias de **datos que NO sean condicionales** y luego completar la siguiente tabla:

Tipo	Registro	N° instrucciones
RAW	X9	2> 3>
RAW	X3	1> 4>
RAW	X3	1> 5>
RAW	X12	3> 5>
RAW	X3	6> 6>
RAW	X10	4> 9>
RAW	X11	8> 9>
~~WAW~~	~~X10~~	~~8> 10>~~
WAW	X3	1-6
WAW	X11	8-10

Tipo	Registro	N° instrucciones
	X0	7-9
	11	7-10
	X3	6-10

++/− b) Sin alterar el orden de las instrucciones, mostrar en la tabla junto al código cómo organizaría los *issue packets* para ejecutar el programa en la menor cantidad posible de ciclos de clock (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.

c) Calcular la cantidad de ciclos de reloj que toma la ejecución del código dado en un procesador LEGv8 de 1-issue con forwarding-stall y en el procesador LEGv8 2-issue descrito. Suponer que en ambos procesadores los saltos son perfectamente predichos y los hazard de control son manejados por hardware (no hay flush de instrucciones). Suponer que N=6 y considerar la totalidad de las iteraciones realizadas. Luego completar:

i. Tiempo de ejecución en 1-issue= ~~115~~ ciclos 42

ii. Tiempo de ejecución en 2-issue= ~~73~~ ciclos 28

iii. Ganancia en velocidad = 1.5+5

Ejercicio 3

Considerando un microprocesador out-of-order execution implementado mediante el algoritmo de Tomasulo, el código en assembler en el ejercicio 2, muestre el contenido de las tablas de estado de las Reservation stations y registros para el 5to clock de ejecución (inclusive). El procesador cuenta con el siguiente hardware:

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 = 2 RS / 3 clk

Ejercicio 4

Se cuenta con un predictor global con un GR de 8 bits, considerando el siguiente segmento de código en LEGv8:

```
.org 0x00
0x1> L: add x1, xzr, #3
0x2>    udiv x2, x0, x1 //Se verifica
0x3>    mul x3, x1, x0  //si x0 es
0x4>    sub x3, x3, x2  //múltiplo de 3
0x5>    cbnz x3, s1
        …
0x6>s1: add x1, xzr, #5
0x7>    udiv x2, x0, x1  //Se verifica
0x8>    mul x3, x1, x0   //si x0 es
0x9>    sub x3, x3, x2   //múltiplo de 5
0xA>    cbnz x3, s2
        …
0xB>s2: add x0, x0, #1
0xC>    cmp x0, 1000
0xD>    b.ne L
```

PHT predictor global

Dirección	Contenido
0x69	11

Parcial 2 - Arquitectura de computadoras 2023

a) Completar la tabla en hexadecimal en las posiciones de la PHT del predictor global que se modifican en dos iteraciones del salto de la posición 0xD, considerando que está completamente inicializada en cero y que X0=12 y el GR=0xA1 antes de comenzar a ejecutarse primer instrucción del código.

b) Considerando que inicialmente X0=0 y que existe un patrón repetitivo (cada dos iteraciones del código aparece un número par, y cada 4 aparece un número múltiplo de 4). ¿Qué cantidad de bits debería tener como **mínimo** el GR para almacenar el patrón del salto de la posición 0x5? ¿Y para el salto de la posición 0xA?

0x5> 9

0xA> 15

Ejercicio 5

Completar el siguiente kernel de openCL que multiplique un vector de N elementos por una constante alpha.

```
__kernel void parallelVMult(
 const int N, const float alpha,
 __global float *a_g)
{
  int i = 0
  float tmp = 0.0f;
  ____________________________
  ____________________________
  ____________________________
}
```

El kernel debe corresponderse con el siguiente código de host:

```
plataform_list = cl.get_platforms()
devices = plataform_list[0].get_devices(device_type =
cl.device_type.GPU)
context = cl.Context(devices=devices)
queue = cl.CommandQueue(context)
a_np = np.arange(N).astype(np.float32)
a_g = cl.Buffer(context, cl.mem_flags.READ_WRITE |
cl.mem_flags.COPY_HOST_PTR, hostbuf = a_np)
program = cl.Program(context, KernelSource).build()
kernel = program.parallelVMult
kernel.set_scalar_arg_dtypes([np.int32, np.float32, None])
kernel.set_args(N, alpha, a_g)
globalRange = a_np.shape
localRange = a_np.shape
ev = cl.enqueue_nd_range_kernel(queue, kernel, globalRange, localRange)
cl.enqueue_copy(queue, a_np, a_g)
```