Arquitectura de computadoras 2019
Parcial 2 - Tema B

95% 10 (diez)

Nombre: Mateo de Mayo

Ejercicio 1: Para un procesador de 32 bits que es capaz de direccionar 1G byte de memoria principal (30 bits de dirección), se requiere implementar una memoria caché con capacidad total de 4M bits para datos. La memoria caché utiliza correspondencia (mapeo) asociativo por conjuntos de 4 vías, con un tamaño de bloque de 8 palabras de 32 bits c/u.
Determinar:

a. ¿Cuántos bits hay en los diferentes campos del formato de dirección de memoria principal?

Memory Address			
Tag	Index	Word	Offset (si aplica)
13	12	3	2

b. ¿Cuántas líneas contiene la memoria caché?

Rta: $2^{14} = 16K$ líneas. (2^{12} líneas cada way / vía)

c. ¿Cuál es la capacidad total de memoria (expresada en bits) necesaria para implementar dicha caché? Considerar el cada línea está compuesta, además, del campo Tag y 1 bit de validación.

Rta: $2^{14} * 270b \approx 540KiB$ bits.

Ejercicio 2: Considere el siguiente fragmento de código de instrucciones LEGv8, el cual modela una lazo tipo ***do-while***, donde X6 contiene la dirección base del arreglo a A del tipo *uint64*.

```
        ADD X1, XZR, XZR
loop:   LSL X9, X1, #3
        ADD X9, X9, X6
        LDUR X10, [X9, #0]  //Load 64bits word
        LDUR X11, [X9, #8]  //Load 64bits word
        ADDI X1, X1, #2
        CMPI X1, #6
        B.NE loop
        ...
```

Suponga un procesador que emite direcciones de 64 bits y posee un sistema de memoria principal direccionable de a bytes. Además, posee solo una caché para DATOS de mapeo directo con un tamaño de 8 bloques de 4 palabras de 64 bits c/u. Suponga que la misma se encuentra vacía al inicio de la ejecución del fragmento. Suponiendo un valor inicial de X6 = 0x 00 81 00 00 00 00 F2 18, completar la siguiente tabla con la información de cada acceso a memoria de datos:

Dirección de acceso (HEX)	Tag (HEX) (56)	Index (HEX) (3)	Word (HEX) (2)	HIT o MISS
0x0081000000 00F218	0x0081000000 00F2	000	11	MISS
0x0081000000 00F220	"	001	00	MISS
" 228	"	001	01	HIT
" 230	"	001	10	HIT
" 238	"	001	11	HIT
" 240	"	010	00	MISS

binario

Arquitectura de computadoras 2019
Parcial 2 - Tema B

Nombre: Mateo de Mayo

Ejercicio 3:

Un procesador 2-issue de arquitectura LEGv8 posee las siguientes propiedades:

1. En cada *issue packet* una instrucción debe ser una operación de acceso a memoria y la otra de tipo aritmética/lógica o un salto.
2. El procesador tiene todos los caminos de forwarding posibles entre las etapas (incluyendo caminos a la etapa ID para la resolución de saltos).
3. El procesador predice los saltos perfectamente.
4. Dos instrucciones no pueden procesarse juntas en un paquete si una requiere el resultado de la otra.
5. El compilador asume toda la responsabilidad de eliminar los hazard, organizar el código e insertar instrucciones "nop" para que el código se ejecute sin necesidad de generación de stalls.

Para el siguiente fragmento de código LEGv8 (donde X2 = 0):

```
        1>  ADDI X11, XZR, #30
        2>  ADDI X0, XZR, #0x200
loop:   3>  LDUR X1, [X0,#0]
        4>  ADDI X0, X0, #8
        5>  ADD X2, X2, X1
        6>  LDUR X1, [X0,#0]
        7>  SUBI X11, X11, #1
        8>  ADD X2, X2, X1
        9>  STUR X2, [X0,#0]
        10> ADDI X0, X0, #16
        11> CBNZ X11, loop
        ...
```

a. Dibuje un diagrama de pipeline que muestre cómo se ejecuta el código LEGv8 dado en el procesador de 2-issue (sólo hasta completar una iteración del bucle). Sin modificar el orden de ejecución, organice el código para evitar la mayor cantidad posible de *stalls*. Deje indicados los caminos de *forwarding* utilizados. (Completar en la tabla dada al final del ejercicio).

b. Suponiendo que no es económicamente viable integrar los multiplexores de tres entradas que son necesarios para implementar todos los caminos de *forwarding*, analice el código dado y determine si es mejor reenviar solo desde el registro de pipeline EX / MEM (EX → EX) o solo desde el registro MEM / WB (MEM → EX).

Rta: Mem → EX .

c. Indique el aumento de velocidad en la ejecución del código dado al pasar de un procesador de 1-issue a un procesador de 2-issue. Considere la totalidad de las iteraciones realizadas por el código.

Rta: n° de ciclos de reloj en 2-issue / n° de ciclos de reloj en 1-issue = 86,76 [%]

d. Cuántas instrucciones LDUR se ejecutarán por lazo si se aplica la técnica del loop-unrolling al código dado, con el fin de minimizar la cantidad de iteraciones del lazo? Asuma que X11 se inicializa en la instrucción <1> con un valor múltiplo de 6.

Rta: 12 veces.

Arquitectura de computadoras 2019
Parcial 2 - Tema B

Nombre: Mateo de Mayo

Nota: las alturas de los puntos de forwarding determinan de que instrucción vienen y a cual van

instr.\clk	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1) ADDI NOP	F	D	E	M	W											
2) ADDI NOP		F	D	E	M	W										
3) LDUR 4) ADDI			F	D	E	M	W									
NOP NOP				F	D	E	M	W								
5) ADD 6) LDUR					F	D	E	M	W							
7) SUBI NOP						F	D	E	M	W						
8) ADD NOP							F	D	E	M	W					
9) STUR 10) ADDI								F	D	E	M	W				
11) CBNZ NOP									F	D	E	M	W			

2 R
dep x0
dep x1
2 R
dep x2

Instruction	Instruction status		
	Issue	Execute	Write result
1> SUBI X2, X4, #16	☑	☑	☑
2> FSUBD D3, D2, D1	☑	☑	☐
3> LDURD D1, [X2, #0]	☑	☐	☐
4> LDURD D2, [X4, #8]	☑	☑	☐
5> FADDD D3, D6, D4	☑	☑	☐
6> FSUBD D4, D5, D4	☑	☐	☐
7> FADDD D5, D4, D3	☑	☐	☐
8> FDIVD D6, D2, D1	☑	☐	☐
9> FMULD D2, D3, D8	☑	☐	☐
10> FSUBD D6, D2, D6	☐	☐	☐
11> LDURD D2, [X2, #16]	☐	☐	☐
12> FMULD D7, D3, D7	☐	☐	☐

Harware
Issue = 4 instrucciones
Suma entera = 2 FU - 4 RS / 1 clock cycles
Suma flotante = 2 FU - 4 RS / 2 clock cycles
Load = 4 RS / 2 clock cycle
Store = 4 RS / 2 clock cycle
Multiplicación flotante = 2 FU - 4 RS / 4 clock cycles
Multiplicación entera = 2 FU - 4 RS / 1 clock cycles

Clk	Stage				
	IF	ID	IS	Ex	WB
0	1,2,3,4				
1	5,6,7,8	1,2,3,4			
2	9,10,11,12	5,6,7,8	1,2,3,4		
3	---	9,10,11,12	5,6,7,8,3	1,2,4	
4	---	~~10,11,12~~	9,6,7,8,3	2,4,5	1,

Full RS

Sob 2 FU

Name	Reservation stations						
	Busy	Op (Inst)	Vj	Vk	Qj	Qk	A
Store 1	☐						
Store 2	☐						
Store 3	☐						
Store 4	☐						
Load 1	☐	(3)	[x2]	#0	~~Mult FP1~~	0	#0
Load 2	☐	(4)	[x4]	#8	0	0	[x4]+#8
Load 3	☐						
Load 4	☐						
Mult FP 1	☐	(8)	-	-	Load2	Load3	
Mult FP 2	☐	(9)	-	[D8]	Add FP2	0	
Mult FP 3	☐						
Mult FP 4	☐						
Add FP 1	☑	(2)	[D2]	[D1]	0	0	
Add FP 2	☑	(5)	[D6]	[D4]	0	0	
Add FP 3	☐	(6)	[D5]	[D4]	0	0	
Add FP 4	☐	(7)	-	-	Add FP3	Add FP2	
Mult int 1	☐						
Mult int 2	☐						
Mult int 3	☐						
Mult int 4	☐						
Add int 1	☑	(1)	[x4]	#16	0	0	
dd int 2	☐						
dd int 3	☐						
dd int 4	☐						

Nota: La tacha de hace referencia a lo que "borró" el WB de <1>

	Register Status								
	D0	D1	D2	D3	D4	D5	D6	D7	D8
Qi		Load 1	Mult FP2	Add FP2	Add FP3	Add FP4	Mult FP1		
	X0	X1	X2	X3	X4	X5	X6	X7	X8
Qi			~~Add FP1~~						