

Arquitectura de computadoras 2022 - Parcial 2A

Nombre: _____

El sistema posee una CACHE exclusiva para INSTRUCCIONES de correspondencia ASOCIATIVA POR CONJUNTOS de 2 VIAS de 128 bytes y 2 palabras de 32 bits por línea, sobre un procesador de 64bits, CPI = 1, que resuelve todos los data y control hazard sin necesidad de stalls, tiene una memoria principal de 4G palabras de 1 byte cada una. Considerar que la CACHE contiene los datos mostrados a continuación al inicio de la ejecución del segmento:

Set	Via #0			Via #1		
	Tag	V	Data	Tag	V	Data
000	00ce520	1	8b1f03ff_8b1f03ff	00ce521	1	9410b5bc_b500004a
001	3fffffff	0	ffffffff_ffffffff	3fffffff	0	ffffffff_ffffffff
010	01f5580	1	8b1f03ff_17ffffffa	00ce581	0	00000000_00000000
011	3fffffff	1	301f030f_87f00ffa	3fffffff	1	9b1f03ff_570ffff8
100	3fffffff	0	ffffffff_ffffffff	3fffffff	0	ffffffff_ffffffff
101	1fffffff	0	00000000_00000000	3fffffff	0	ffffffff_ffffffff
110	3fffffff	0	ffffffff_ffffffff	01f5580	1	9b1f0300_870f65f8
111	00000f1	1	9410b5bc_b500004a	3fffffff	0	00000000_00000000

a) Determinar el estado de la CACHE al final de la ejecución del segmento para $N = 100$. Para esto completar la siguiente tabla SOLO con el contenido de las líneas de CACHE que hayan sufrido modificaciones. El tamaño de la tabla no representa la cantidad de líneas necesarias.

Set	Tag	V	Data	Tag	V	Data
1						
2						
3						
4						
5						
6						
7						
8						
9						
10						

b) Determine la cantidad de ciclos de clk necesarios para su ejecución en las mismas condiciones del punto a), considerando que cada MISS de caché tiene un tiempo de ejecución de 10 ciclos. Suponer que el pipeline ya se encuentra en régimen y que la CACHE de DATOS solo produce aciertos en los accesos, por lo que no se tiene penalidades por acceso a datos.

Expresión: _____ Respuesta: _____ ciclos de clk.

Considere una caché de 4Mbyte, ASOCIATIVA por conjuntos de 8 vías, dispuesta en un procesador de 64 bits con una capacidad de direccionamiento de 4Gbytes (cada byte es directamente direccionable en memoria). Se sabe que la memoria principal posee 32M bloques. Se pide:

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

Tag	Set	Word	Offset byte(*)

* Llenar con 0 en el caso que no corresponda

b) Suponga que cada LÍNEA de la caché contiene además un bit de validación (V) y 8 bits de contador de accesos. Cual es el tamaño completo de un CONJUNTO (expresado en bits) de la caché, considerando datos, tags y los bits de status antes mencionados?

Respuesta: _____ bits.

c) Cuál es el tamaño total de cada VÍA de la caché, expresada en bits?

Respuesta: _____ bits.

d) Si la memoria caché y la memoria principal tienen tiempos de acceso de 5ns y 400ns respectivamente, ¿qué hit rate (tasa de acierto) se necesitaría para obtener un tiempo promedio de acceso a memoria (AMAT) de 25ns?

Respuesta: _____

Ejercicio 2

Considere la ejecución del siguiente segmento de código LEGv8 para $N > 0$, donde $N \rightarrow X3$ y $X6$ contiene la dirección base del arreglo $A[]$ del tipo `uint64_t`.

ADDRESS	<LABEL>	OPCODE	ASSEMBLY
	...		
<code>0x03394834</code>	<code><COUNT_Z></code> :	<code>0x8b1f03e0</code>	<code>add x0, xzr, xzr</code>
		<code>0x8b0303e9</code>	<code>add x9, xzr, x3</code>
<code>0x03394838</code>	<code><LOOP_FOR></code> :	<code>0xb40000e9</code>	<code>cbz x9, NEXT</code>
		<code>0xf84000ca</code>	<code>ldur x10, [x6]</code>
		<code>0xb500004a</code>	<code>cbnz x10, NOT_Z</code>
		<code>0x9410b5bc</code>	<code>bl ZERO</code>
<code>0x03394848</code>	<code><NOT_Z></code> :	<code>0xd1000529</code>	<code>subi x9, x9, #0x1</code>
		<code>0x910020c6</code>	<code>addi x6, x6, #0x8</code>
		<code>0x17ffffffa</code>	<code>b LOOP_FOR</code>
<code>0x03394854</code>	<code><NEXT></code> :	<code>0x8b1f03ff</code>	<code>add xzr, xzr, xzr</code>
	...		
<code>0x037c1f38</code>	<code><ZERO></code> :	<code>0x91000400</code>	<code>addi x0, x0, #0x1</code>
		<code>0xd61f03c0</code>	<code>br x30</code>

Nombre: _____

Ejercicio 3

Dado un procesador de arquitectura LEGB 2-issue, que predice los saltos perfectamente, de modo que los hazard de control son manejados por hardware, con una modificación que permite que en cada issue packet una instrucción pueda ser cualquier tipo y la otra deba ser una instrucción aritmética. Para el siguiente fragmento de código LEGB, donde todos los registros se encuentran inicializados con '0':

```

1> ADDI X0, XZR, #32
LOOP: 2> ADDI X1, X10, #8x200
3> SUB X2, X10, X0
4> CBZ X2, LOOP_END
5> ADD X11, X1, XZR
6> ADDI X10, X10, #8
7> LDUR X12, [X11, #0]
8> LDUR X13, [X11, #8]
9> ADD X14, X12, X12
10> ORR X12, X13, X13
11> STUR X13, [X11, #0]
12> STUR X12, [X11, #8]
13> B LOOP
    
```

Instrucción de cualquier tipo	Instrucción aritmética
1	1
2	2
3	2
4	2
5	2
6	2
7	2
8	2
9	2
10	2
11	2
12	2
13	2

LOOP_END: ...

a) Sin alterar el orden de las instrucciones, mostrar en la tabla de arriba 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). Usar los números correspondientes para referirse a las instrucciones del código.

b) Mostrar el orden de ejecución del código del punto "a" en el procesador 2-issue (sólo hasta completar una iteración del bucle "LOOP"). Indicar los caminos de forwarding utilizados.

c) El siguiente código resultó de aplicar las técnicas de loop unrolling y register renaming al fragmento de código dado. Sin alterar los resultados obtenidos tras la ejecución del mismo y asumiendo que siempre en la instrucción n°1 se asigna el número 32 al registro X0, eliminar (tachar) la mayor cantidad de instrucciones posible y completar los espacios vacíos en las restantes.

```

ADDI X0, XZR, #32
LOOP: ADDI X1, X10, 8x200
SUB X2, X10, X0
CBZ X2, LOOP_END
ADD X11, X1, XZR
ADDI X10, X10, #8
LDUR X12, [X11, #0]
LDUR X13, [X11, #8]
ADD X14, X12, X12
ORR X12, X13, X13
STUR X13, [X11, #0]
STUR X12, [X11, #8]
ADDI X1, X10, #8x200
    
```

```

SUB X2, X10, X0
CBZ X2, LOOP_END
ADD X11, X1, XZR
ADDI X10, X10, #8
LDUR X12, [X11, #8]
LDUR X13, [X11, #16]
ADD X14, X12, X12
ORR X12, X13, X13
STUR X13, [X11, #8]
STUR X12, [X11, #16]
B LOOP
LOOP_END: ...
    
```