

25/10/24

Apellido, Nombre:

Ejercicio 1

Considere la ejecución del siguiente segmento de código LEGv8 para $i > 0$, donde $i \rightarrow X3$, 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]
      ADD X0, x10, X0
      SUBI x9, x9, #8
      B loop
end:  STUR X0, [X9, #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 CACHE 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(*)

* 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: 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]

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		
RAW condicional		
WAW		

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

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.



