Parcial 2 - 2023 (Tema A)

Ejercicio 1:

Determinar cuáles de las siguientes instrucciones pueden ser ensambladas en LegV8. Justificar las respuestas negativas.

Instrucción	Si/No	Justificación
B 0	Si	Salta a la dirección de memoria 0.
LDUR X1, [XZR, X3]	No	Ya que el ldur espera como segundo operando un inmediato y “x3” es un registro.
MOVK X15, #0xCAFE, LSL#8	No	Ya que el máximo permitido del Imm es 2^15 - 1, y 0xCAFE se pasa.
ORRI XZR, X1, #-16	No	Ya que el valor inmediato en el ORRI es sin signo.
LSL X30, X30, #33	Si	Ya que el shamt puede ser hasta 2^6 - 1 = 63, y 33 es menor que 63

Ejercicio 2:

Usando el siguiente ejemplo, escriba un programa en LegV8 que divida X0 por X1 y salte dependiendo si el resultado es mayor o igual a 0.33. Se pueden usar todas las instrucciones de LegV8 excepto las de punto flotante. Debe resolverse en 5 instrucciones o menos

Lo que voy a hacer es hacer un poco de álgebra bonita:

$\frac{a}{b} \geq 1/3$

```
MOVZ X0, #1, LSL#0          // X0 = 1
MOVZ X1, #3, LSL#0          // X1 = 3
LSL X2, X0, #1              // X2 = X0 * 2^1 = 1 * 2 = 2
ADD X2, X2, X0              // X2 = X2 + X0 = 2 + 1 = 3 (multiplico por 3)
CMP X2, X1                  // Comparamos x2 = 3 con x1 = 3
B.GE end                    // Saltar si X0/X1 ≥ 0,33
```

Ejercicio 3:

Dada la siguiente sección de un programa en assembler LegV8, el registro X1 contiene la dirección base de un arreglo A, mientras que X0 contiene el tamaño de dicho arreglo.

Asuma que los registros y la memoria contienen los valores mostrados en la tabla al inicio de la ejecución de dicha sección.

```
   ADDI X10, XZR, #0          // x10 = 0 + 0 = 0
   SUBI X0, X0, #1            // x0 = x0 - 1 | 0x00000004
Loop: CMP X10, X0             // Compara x10 con x0,

B.GE LOOP_END               // Si x10≥x0 salta a LOOP_end
Proc: LSL X11, X10, #3        // x11 = x10 * 2^3 = 0 * 2^3 = 0
   ADD X11, X1, X11           // x11 = x1 + x11 = x1 + 0  = x1 | x1 = 10000100
   LDUR X12, [X11, #0]        // x12 cargo lo que hay en la dirección de memoria que apunta x11
   LDUR X13[X11, #8]          // x13 cargo lo que hay en la direc. de memoria + 8  q apunta x11
   CPM X13, X12               // comparo x13 con x12

B.GT NO_XCHG                // si x13 > x12 salto a NO_XCHG
   STUR X13 [X11, #0]         // almaceno lo que hay en x13 en la dirección apuntada por x11
   STUR X12, [X11, #0]        // almaceno lo que hay en x12 en la dirección apuntado por x11
```

```
NO_XCHG: ADDI X10, X10, #1  // x10 = x10 + 1
   B LOOP                   // salto a loop
LOOP_END ...
```

Direccion	Contenido (antes)	Contenido (despues)
0x10000100	122	-30
0x10000108	-30	70
0x10000110	70	10
0x10000118	10	122
0x10000120	45	45 (no se modifico)
0x10000128	200	200 (no se modifico)

Registro	Valor del registro
X0	0x00000004
X1	0x10000100

A) ¿Cómo queda el contenido de todas las posiciones de memoria mostradas en la tabla al finalizar la sección del programa? Responde completando las columnas en blanco de la tabla de memoria.

ESTA MAL PORQUE PUSE EL SEGUNDO STUR CON 0 Y HABIA UN 8 Y ME QUIERO CORTAR LOS TESTICULOS PORQUE ME DI CUENTA CUANDO TERMINE (LA TABLA ESTA BIEN CON LOS RESULTADOS)

Para hacer esto debemos seguir el flujo del codigo considerando los valores iniciales indicados.

```
 ADDI X10, XZR, #0          // x10 =  0
 SUBI X0, X0, #1            // x0 = x0 - 1 = 4 - 1 = 3
Loop: CMP X10, X0           // compara:
0, 3 | 1,3 |  2, 3 | 3, 3

B.GE LOOP_END               // 0<3 (no salta) | 1<3 | 2<3 | 3≥ 3 (salto)
Proc: LSL X11, X10, #3      //
x11 =  0 |  x11 = 8 | x11 = 16
   ADD X11, X1, X11         //
x11 = 0x10000100 | x11 =  0x10000108 | x11  = 0x10000110
   LDUR X12, [X11, #0]      //
x12 = Mem[0x10000100] = 122 | x12 = Mem[0x10000108] = -30 | x12 = Mem[0x10000110] = 70
   LDUR X13, [X11, #8]      //
x13 = Mem[0x10000108] = -30 |  x13 = M[0x10000110]=70 | x13 = M[0x10000118] = 10
   CPM X13, X12             //
comp: -30, 122 |  70, -30 | 10, 70

B.GT NO_XCHG                // -30 ≤ 122 ( no salto) |  70>-30 (salto) | 10 < 70 (no salto)
   STUR X13 [X11, #0]       //
Mem[0x10000100] = x13 = -30 | Mem[0x10000110] = x13 = 70
   STUR X12, [X11, #0]      //
Mem[0x10000100] = x12 = -30 | Mem[0x10000110] = x12 = 70
NO_XCHG: ADDI X10, X10, #1  //
x10 = 1 | x10 = 2 | x10 = 3
   B LOOP                   // salto a loop
LOOP_END ...
```

B) La instrucción contenida en la línea con el label PROC se ejecuta 3 veces. (esto si esta bien en el analisis jajja)

Ejercicio 4:

Considere el segmento de memoria que se muestra en la primera columna de la forma dirección: contenido que contiene codificado un programa en ISA LegV8. Parte del programa desensamblado se presenta en la 2da columna.

label	Direccion: contendio	Desensamblado
	0x10000100: 0x910013E0	ADDI x0, XZR, #4
loop	0x10000104: 0xB89103E9	LDURSW x9, [XZR, #-240]
	0x10000108: 91001529	ADDI X9, X9, #5
	0x1000010C: B81103E9	STURW X9, [XZR, 0x110]
	0x10000110: 13FFFFFD	B loop
	0x10000114: 8B1F03E0	ADD X0, XZR, XZR
	0x10000118: D3600400	LSL x0, x0, #1

a) Completar las instrucciones restantes.

Para completar las instrucciones hay que desensamblarlas:

0xB89103E9 = 1011 1000 1001 0001 0000 0011 1110 1001 (pasamos de hexa a binario)

los 11 primeros bits nos dicen el rango del opcode para saber de que instruccion estamos hablanado ⇒ 1011 1000 100 = 0101 1100 0100 = 0x5C4

Viendo la green card sabemos que 0x5C4 corresponde al opcode de LDURSW

Ahora complemetemos los datos:

opcode	**DT_ADDRESS**	**OP**	**Rn**	**Rt**
11 bits	9 bits	2	5	5
10111000100	100010000	00	11111	01001

DT_ADDRESS = 100010000 = -240 (ya que es signado en complemento a 2)

Rn = 11111 = 31

Rt = 01001 = 9

Entonces completando la instruccion tenemos: LDURSW x9, [XZR, #-240]

Ahora desensamblemos la otra:

0xD3600400 = 1101 0011 0110 0000 0000 0100 0000 0000

Busquemos el opcode: 1101 0011 011 = 0110 1001 1011 = 0x69B

Viendo en la green card sabemos que se trata de LSL, la cual es de tipo “R”:

opcode	Rm	shamt	**Rn**	**Rd**
11 bits	5 bits	6 bits	5	5
11010011011	00000	0000 01	00 000	0 0000

Rm = 00000 = x0

Shamt = 000001= #1

Rn = x0

Rd = x0

Completando la instruccion tenemos: LSL x0, x0, #1

b) ¿Cuántas veces se ejecuta la instrucción con label: loop? 1 vez

c) ¿Cuál es el valor de X0 luego de la ejecución del segmento?

Ejercicio 5:

Considerar que el procesador está ejecutando la instrucción de LegV8: 0xF801816A y el contenido de los registros es:
X10 = 0x20, X11 = 0x23, X12 = 0x54. X13 = 0x00, PC = 0x104

a) Desensamblar la instrucción.

Desensamblemos la instrucción: 0xF801816A = 1111 1000 0000 0001 1000 0001 0110 1010

Los 11 primeros bits son el opcode que nos dicen de que instrucción se trata: **1111 1000 000**

0111 1100 0000 = 0x7C0 = **STUR**

Como la instrucción es STUR sabemos que es de tipo: "D" entonces:

opcode	DT_ADDRESS	op	Rn	Rt
11 bits	9 bits	2 bits	5 bits	5 bits
11111000000	000011000	00	01011	01010
STUR	24	0	11	10

Entonces juntando esos datos tenemos: **STUR x10, [x11, #24]**

b) ¿Qué operación realiza la ALU?

: La ALU realiza una operación de suma para calcular la dirección efectiva de memoria sumando el contenido de x1 y el desplazamiento inmediato 24.

c)
Completar el estado de las señales de control

Reg2Loc	ALUSrc	MemtoReg	Regwrite	MemRead	MemWrite	Branch
1	1	X	0	0	1	0

Completar:

Senal	E/S	N-Bits	Valor	ID (para explicacion abajo)
Register, Read data 1	S	64	0x23	1
Rgister, Read register 2	E	5	0xA	2
Data Memory, address	E	64	0x3b	3
Register, write register	E	5	0xA	4
Entrada del pc	E	64	0x108	5
Add, ALUResult	S	64	0x164	6

1. En la instruccion STUR leemos la data del registro 1, ya que viendo en el datapath, es en los bits del 9-5, qeu corresponden al Rn, en nuestro caso el Rn es x11, por eso el valor es 0x23, x11 por defecto es de 64bits, asi qeu el N-Bits es 64, y es de salida ya que proporciona datos desde un registro interno del procesador hacia otros componentes o etapas del datapath donde esos datos son necesarios para realizar operaciones específicas.
2. En este caso, sabemos que el Reg2Loc es 1, asi que el mux del datapath va a dejar pasar la senal de la isntruction del 4-0, que en el stur corresponde al Rt, que es x10, por eso el valor es 0xA, como el Read

Register 2 es una señal de entrada que indica qué registro se está leyendo como segundo operando (requiere 5 bits para especificar el número de registro, ya que hay 31 registros qeu se puede usar y 2^5 = 32) para la operación actual.

3. El data memory address, va a leer la direccio nde memoria, por lo tanto es de entrada, ademas esa direccion va a ser de 64bits, y en el caso del stur es la parte del DT_ADdress que en nuestra instruccion es 0x3b, ya que corresponde a la dirección calculada sumando el contenido de X11 (0x23) y el offset (#24). Por lo tanto, 0x23 + 0x18 = 0x3b.
4. En el gasto del Register, write register, es el registro en el que se escribirá el dato cargado desde la memoria, que es X10 en este caso, por eso la cantidad de bits es 64, y el valor es 0xA que corresponde a x10.
5. Esto representa la dirección actual del program counter (PC) como inicialmente eera 0x104, y realizamos la isntruccoin le sumamos los 4 = 0x108, y bueno la palabra ya te dice que ese de entrada.