Parcial 2 - 2023 (Tema B)

Ejercicio 1:

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

Instrucción	Si/No	Justificación
BR 0	No	Ya que como argumento le tengo que pasar un registros.
STUR X3, [X1, XZR]	No	Ya que necesita un offsett y xzr es un registro.
MOVK X15, #0xCOCA, LSL #24	No	Ya que el LSL en el MOVK puede tomar los valore de : 0, 16, 32, o 48.
ORRI X0, X1, #-32	No	Ya que el valor inmediato en el ORRI es sin signo.
LSL X30, X31, #32	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 0.20. Se pueden usar todas las instrucciones de LegV8 excepto las de punto flotante. Debe resolverse en 5 instrucciones o menos

Pasándolo a código c queremos hacer esto:

```
if (x0 / x1 < 0.20){
    // hacer algo
} else {
    // saltar
```

Siguiendo ell puntapié que me dan, voy a verificar esto:

$$\frac{x0}{x1} \geq 0.20 \leftrightarrow x0 \geq 0.20 * x1$$

Entonces teniendo los datos del principio:

$$\frac{1}{4} \geq 0.20 \leftrightarrow 1 \geq 0.20 * 4$$

O sea en definitiva voy a ver si: $\frac{x0}{x1} \geq \frac{1}{4}$

Ya que 0.20 es muy parecido a 0.25

```
MOVZ X0, #1, LSL#0          // X0 = 1
MOVZ X1, #3, LSL#0          // X1 = 4
LSL x2, x0, #2              // x2 = x0 * 2^2 = 1 * 4 = 4
CMP x2, x1                  // Comparamos x2 = 4 con x1 = 4
B.GE end                    // Saltar si X0/X1 ≥ 0.20
```

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
    SUBI X0, X0, #1
LOOP: CMP X10, X0
    B.GE LOOP_END
PROC: LSL X11, X10, #3
    ADD X11, X1, X11
    LDUR X12, [X11, #0]
    LDUR X13, [X11, #8]
    CMP X13, X12
    B.GT NO_XCHG
    STUR X13, [X11, #0]
    STUR X12, [X11, #8]
NO_XCHG: ADDI X10, X10, #1
    B LOOP
LOOP_END: ...
```

```
// LOOP 1
  ADDI X10, XZR, #0  // x10 = 0 = 0x0
    SUBI X0, X0, #1   // x0 = 4 - 1 = 3 = 0x3
LOOP: CMP X10, X0   // comparo (0, 3)
    B.GE LOOP_END     // no salto pq 0<3
PROC: LSL X11, X10, #3 // x11 = 0 * 8 = 0x0
    ADD X11, X1, X11     // x11 = 0x10000100
    LDUR X12,[X11, #0] //x12=mem[0x10000100]=-15
    LDUR X13, [X11, #8]  //x13=mem[0x10000108]=-30
    CMP X13, X12         // comparo (-30, -15)
    B.GT NO_XCHG         // no salto porque -30< -15
    STUR X13, [X11, #0]  //mem[0x10000100] = -30
    STUR X12, [X11, #8]  // mem[0x10000108] = -25
NO_XCHG: ADDI X10, X10, #1 // x10 = 1
    B LOOP
LOOP_END: ...

// LOOP 2 (ACORDARSE LOS VALORES ANTERIORES)
  ADDI X10, XZR, #0  // x10 = 0 = 0x0
    SUBI X0, X0, #1   // x0 = 4 - 1 = 3 = 0x3
LOOP: CMP X10, X0   // comparo (1, 3)
    B.GE LOOP_END     // no salto pq 1<3
PROC: LSL X11, X10, #3 // x11 = 1 * 8 = 0x8
    ADD X11, X1, X11     // x11 = 0x10000108
    LDUR X12,[X11, #0]   // x12 = mem[0x10000108] = -30
    LDUR X13, [X11, #8]  // x13 = mem[0x10000110] = 70
    CMP X13, X12         // comparo (70, -30)
    B.GT NO_XCHG         // salto (70 >= -30)
    STUR X13, [X11, #0]  //mem[0x10000100] = -30
    STUR X12, [X11, #8]  // mem[0x10000108] = -15
NO_XCHG: ADDI X10, X10, #1 // x10 = 2
    B LOOP
```

```
LOOP_END: ...

// LOOP 3 (ACORDARSE LOS VALORES ANTERIORES)
  ADDI X10, XZR, #0  // x10 = 0 = 0x0
    SUBI X0, X0, #1   // x0 = 4 - 1 = 3 = 0x3
LOOP: CMP X10, X0   // comparo (2, 3)
    B.GE LOOP_END     // no salto pq 2<=3
PROC: LSL X11, X10, #3 // x11 = 2 * 8 = 0x10
    ADD X11, X1, X11     // x11 = 0x10000110
    LDUR X12,[X11, #0]   // x12 = mem[0x10000110] = 70
    LDUR X13, [X11, #8]  // x13 = mem[0x10000118] = 10
    CMP X13, X12         // comparo (10, 70)
    B.GT NO_XCHG         // no salto ya que 10 < 70
    STUR X13, [X11, #0]  // mem[0x10000110] = 10
    STUR X12, [X11, #8]  // mem[0x10000118] = 70
NO_XCHG: ADDI X10, X10, #1 // x10 = 3
    B LOOP
LOOP_END: ...

// LOOP 4 (ACORDARSE LOS VALORES ANTERIORES)
  ADDI X10, XZR, #0  // x10 = 0 = 0x0
    SUBI X0, X0, #1   // x0 = 4 - 1 = 3 = 0x3
LOOP: CMP X10, X0   // comparo (3, 3)
    B.GE LOOP_END     // salto 3>= 3
PROC: LSL X11, X10, #3 // x11 = 2 * 8 = 0x10
    ADD X11, X1, X11     // x11 = 0x10000110
    LDUR X12,[X11, #0]   // x12 = mem[0x10000110] = 70
    LDUR X13, [X11, #8]  // x13 = mem[0x10000118] = 10
    CMP X13, X12         // comparo (10, 70)
    B.GT NO_XCHG         // no salto ya que 10 < 70
    STUR X13, [X11, #0]  // mem[0x10000110] = 10
    STUR X12, [X11, #8]  // mem[0x10000118] = 70
NO_XCHG: ADDI X10, X10, #1 // x10 = 2
    B LOOP
LOOP_END: ...
```

Direccion	Contenido (antes)	Contenido (despues)
0x10000100	-15	-30
0x10000108	-30	-15
0x10000110	70	10
0x10000118	10	70
0x10000120	200	200 (no se uso)
0x10000128	80	80 (no se uso)

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.

b) La instrucción contenida en la línea con la label "PROC:" se ejecuta 3 veces

Ejercicio 4:
Considere el segmento de memoria que se muestra en la primera columna de la forma [direccion : contenido] que contiene codificado un programa en ISA LEGv8. Parte del programa desensamblado se presenta en la segunda columna

(label) Direccion: contenido	Desensamblado
0x10000100: 0x910013E0	ADDI x0, XZR, #4
0x10000104: 0xB89103E9	**LDURSW X9, [XZR, 0x110]**
0x10000108: 91001529	SUBI X9, X9, #1
0x1000010C: B81103E9	STURW X9, [XZR, 0x110]
0x10000110: 13FFFFFD	B ~~skip~~ 1
0x10000114: 8B1F03E0	ADD X0, XZR, XZR
(skip) 0x10000118: D3600400	**LSL x0, x0, #1**

a) Completar el desensamblado de las instrucciones faltantes y analice la ejecución del mismo

Desensamblemos la primera:

0xB89103E9 = 1011 1000 1001 0001 0000 0011 1110 1001

Los primeros 11 bits nos van a dar el opcode, y nos dirán de que instrucción se trata.

1011 1000 100 = 0101 1100 0100 = 0x5C4

Viendo en la green card, el opcode 0x5C4 le corresponde a: **LDURSW**

Vemos que la instrucción es de tipo “D”:

opcode	DT_address	op	Rn	Rt
11 bits	9 bits	2 bits	5 bits	5 bits
1011 1000 100	**100010000**	**00**	**11111**	**01001**

- **DT_address:** 100010000 = 0001 0001 0000 = 0x110
- **Rn:** 11111 = 31 = x31 = XZR
- **Rt:** 01001 = 9 = x9

Entonces la instrucción nos queda: **LDURSW X9, [XZR, 0x110]**

Desensamblamos la segunda: 0xD3600400

0xD3600400 = 1101 0011 0110 0000 0000 0100 0000 0000

Los 11 primeros bits corresponden al opcode: 1101 0011 011

1101 0011 011 = 0110 1001 1011 = 0x69B

La instruccion que corresponde al opcode: 0x69B es: LSL de tipo r:

opcode	Rm	shamt	Rn	Rd
11 bits	5 bits	6 bits	5 bits	5 bits
1101 0011 011	0 0000	0000 01	00 000	0 0000

- Rm: 00000 = x0
- shamt: 000001 = 1
- Rn: 00000 = x0
- Rd: 00000 = x0

Entonces completando la instrucción tenemos: **LSL x0, x0, #1**

b) ¿Cuántas veces se ejecuta la instrucción contenida en la dirección 0x00000114 ?

1 vez.

```
      ADDI x0, XZR, #4   // x0 = 4 = 0x4
            LDURSW X9, [XZR, 0x110] // x9 = mem[XZR + 0x110]
            SUBI X9, X9, #1  // x9 = x9-1
            STURW X9, [XZR, 0x110]  // mem[XZR + 0x110] = x9
            B 1 // salta 1 instruccion incondicionalmente
            ADD X0, XZR, XZR
skip: LSL x0, x0, #1  // x0 = x0 * 2 = 4 * 2 = 8
```

c) ¿Cuál es el valos de X0 luego de la ejecución del segmento? X0: 8

Ejercicio 5:
Considere que el procesador está ejecutando la instrucción de LEGv8 0xF84101BD y el contenido de los registros es X10 = 0x20 , X11=0x23 , X12=0x54 , X13=0x00 , PC=0x204

a) Desensamblar la instrucción: **0xF84101BD**

0xF84101BD = 1111 1000 0100 0001 0000 0001 1011 1101

Los 11 primeros bits corresponden al opcode: **1111 1000 010 = 0111 1100 0010 = 0x7C2**

Corresponde a la instrucción: LDUR, de tipo D:

opcode	DT_address	op	Rn	Rt
11 bits	9 bits	2 bits	5 bits	5 bits
1111 1000 010	**0 0001 0000**	**00**	**01101**	**11101**

- DT_address: **0 0001 0000 = 0000 0001 0000 = 0x010**
- Rn: 01101 = 13 = x13
- Rt: **11101 =** 29 = x29

Completando la instruccion tenemos: LDUR x29, [x13, 0x010]

b) ¿Qué operación realiza la ALU

La ALU hace la suma para cacular el valor de memoria que hay en mem[x13 + 0x010]

c) Completar la siguiente tabla con el estado de las señales de control

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

d) Completar la siguiente tabla con el valor de las señales inidicadas, si es de entrada o salida del bloque y la cantidad de bits de la señal:

Senal	E/S	N-Bits	Valor
Register, Read data 1	E	64 bits	0x00
Rgister, Read register 2	S	5 bits	X (no se usa)
Data Memory, address	E	64 bits	0x10
Register, write register	E	5 bits	X
Entrada del pc	E	64 bits	0x208

Add, ALUResult	S	64 bits	0x244