Ejercicio 4:

Transformar de binario a hexadecimal. ¿Qué instrucciones LEGv8 representan en memoria?

4.1) `1000 1011 0000 0000 0000 0000 0000 0000`

4.2) `1101 0010 1011 1111 1111 1111 1110 0010`

OP code

0x4 5 8 Rm Shamt Rn Rd

4.1) 1000 1011 0000 0000 0000 0000 0000 0000 = 0x8B000000

OPcode = 0x458 => Es un ADD tipo R

Rm = X0

Rn = X0

Rd = X0 => Esto es un ADD X0, X0, X0

6 9 5 Rd

4.2) 1101 0010 1011 1111 1111 1111 1110 0010 = 0xD2BFFFE2

OPcode: 0x695 = MOVZ (9 bits) tipo Im

LSL: 01 = #16

Rd: X2

MOV-inm: 1111 1111 1111 1111 = 0xFFFF

=> MOVZ X2, 0xFFFF, LSL #16

Ejercicio 5:

Ejecutar el siguiente código assembler que está en memoria para dar el valor final del registro X1. El contenido de la memoria se da como una lista de pares, *dirección de memoria: contenido,* suponiendo alineamiento de memoria del tipo *big endian*. Describa sintéticamente que hace el programa.

```
0x10010000: 0x8B010029
0x10010004: 0x8B010121
```

Primero desensamblo 4 5 8 Rm Shamt Rn Rd

0x8B010029 = 1000 1011 0000 0001 0000 0000 0010 1001

OPcode: 0x458 = Add tipo R

Rm = X1

Rn = X1

Rd = X5 -> ADD X5, X1, X1

ADD Rm Shamt Rn Rd

0x8B010121 = 1000 1011 0000 0001 0000 0001 0010 0001

Rm = X1 => ADD X1, X5, X1

Rn = X5

Rd = X1

=> 0x10010000: 0x8B010029 ADD X5, X1, X1

0x10010004: 0x8B010121 ADD X1, X5, X1

El programa hace X1 * 3

Ejercicio 6:

Decidir cuáles de las siguientes instrucciones en assembler se pueden codificar en código de máquina LEGv8. Explique qué falla en las que no puedan ser ensambladas.

1. `LSL XZR, XZR, 0` SÍ
2. `ADDI X1, X2, -1` NO, solo positivos
3. `ADDI X1, X2, 4096` NO, Hasta 4095
4. `EOR X32, X31, X30` NO, X32 no existe
5. `ORRI X24, X24, 0x1FFF` NO
6. `STUR X9, [XZR,#-129]` SÍ
7. `LDURB XZR, [XZR,#-1]` SÍ
8. `LSR X16, X17, #68` NO, max #63
9. `MOVZ X0, 0x1010, LSL #12` NO, LSL solo 00, 16, 32, 48
10. `MOVZ XZR, 0xFFFF, LSL #48` SÍ, es el límite.

0x1FFF = 0001 1111 1111 1111 = 8191

ALUIm = Maximo 4095

Ejercicio 7:

Ensamblar estos *delay loops*.

A)

```
    MOVZ X0, 0x1, LSL #48
L1: SUBI X0,X0,#1
    CBNZ X0, L1
```

B)

```
    MOVZ X0, 0xFFFF, LSL #32
L1: SUBIS X0,X0,#1
    B.NE L1
```

C)

```
      MOVZ X0, 0x2, LSL #16
L1:   SUBIS XZR,X0,#0
      B.EQ EXIT
      SUBI X0,X0,#1
      B L1
EXIT:
```

MOVZ X0, 0x1, LSL #48

opcode es de 9 bits = 1101 0010 1

LSL = 11

Mov_im = 0000 0000 0000 0001

Rd = 00000

=> 1101 0010 1110 0000 0000 0000 0010 0000 = 0xD2E00020

SUBI X0, X0, #1

opcode 10 bits = 1101 0001 00

ALU im 12 bits = 0000 0000 0001

Rn = 0 0000

Rd = 00000

=> 1101 0001 0000 0000 0000 0100 0000 0000 = 0xD1000400

D 1 0 0 0 4 0 0

CBNZ X0, L1 (-1)

opcode CB de 8 bits = 1011 0101

Cond_B_Adr 19 bits = 1111 1111 1111 1111 111

Rt = 0 000

=> 1011 0101 1111 1111 1111 1111 1110 0000 = 0xB5FFFFE0

4) 0xD2E00020
0xD1000400
0xB5FFFFE0

Ejercicio 8:

Dadas las siguientes direcciones de memoria:

`0x00014000`
`0x00114524`
`0x0F000200`

8.1) Si el valor del PC es `0x00000000`, ¿es posible llegar con **una** sola instrucción **conditional branch** a las direcciones de memoria arriba listadas?

8.2) Si el valor del PC es `0x00000600`, ¿es posible llegar con **una** sola instrucción **branch** a las direcciones de memoria arriba listadas?

8.3) Si el valor del PC es `0x00000000` y quiero saltar al primer GiB de memoria `0x40000000` . Escribir exactamente 2 instrucciones contiguas que posibilitan el **salto lejano** (*far jump*).

El conditional branch tiene un campo de 19 bits pero admite negativos por lo tanto su rango es de $[-2^{18}, 2^{18}-1]$. Tomemos solo su rango positivo.

=> $2^{18}-1$ = 0 11 1111 1111 1111 1111 = 0x3FFFF

luego hay que multiplicarlo por 4.

=> 0x FFFFC Ahora comparamos con las distintas direcciones.

`0x00014000` SÍ
`0x00114524` NO
`0x0F000200` NO

8.2) Si el valor del PC es `0x00000600`, ¿es posible llegar con **una** sola instrucción **branch** a las direcciones de memoria arriba listadas?

El branch address es de 26 bits, con el rango $[-2^{25}, 2^{25}-1]$

=> 2 01 1111 1111 1111 1111 1111 1111 $= 2^{25}-1$

=> 0x1FFFFFF * 4 = 0x7FFFFFC

Ahora le sumamos 0x600

=> 0x7FFFFFC
0x600
0x8000 5FC

Ahora Comparamos
0x08000 5FC

`0x00014000` SÍ
`0x00114524` SÍ
`0x0F000200` NO

8.3) Si el valor del PC es `0x00000000` y quiero saltar al primer GiB de memoria `0x40000000` . Escribir exactamente 2 instrucciones contiguas que posibilitan el **salto lejano** (*far jump*).

```
MOVZ X1, 0x40, LSL #16
BR X1
```

Ejercicio 9:

Suponiendo que el PC está en la primera palabra de memoria 0x00000000 y se desea saltar a la última instrucción de los primeros 4 GiB o sea a 0xFFFFFFFC, ¿Cuántas instrucciones **B** son necesarias? (no se puede usar BR).

Recordemos que el rango positivo del B es $0x8000\,05FC$

Sabemos que $0xFFFFFFFC \geq 0x080000\,5FC$

Podemos hacer la división entre los dos números para ver cuantas instrucciones necesitamos.

Pasemos los números a decimal:

$15.16^7+15.16^6+15.16^5+15.16^4+15.16^3+15.16^2+15.16+12$

4 026 531 840
251 658 240
15 728 640
983 040
61 440
3 840
240
12

4 294 967 292 $= 0x$ FFFFFFFC

$8.16^6+5.16^2+15.16+12$

134 217 728
1 280
240
12

134 219 260 $= 0x080000\,5FC$

$\Rightarrow 4294967292 \,/\, 134219260 \approx 31.99\ldots$

Hacen falta 32 instrucciones de salto // En realidad 33. La dirección era 0xFFFFFFFC no $0x080000\,5FC$

Ejercicio 10:

¿Qué valor devuelve en X0 este programa?

```
        .org 0x0000
        MOVZ X0, 0x0400, LSL #0
        MOVK X0, 0x9100, LSL #16
        STURW X0, [XZR,#12]
0x0000000C: STURW X0, [XZR,#12]
```

$\Rightarrow X0 = 0x9100\,0400$

$0x0000000C = 0x9100\,0400$

Desensamblo 0x9100 0400

Opcode $0x\,988$ = ADDI (los bits) bien :)

$\Rightarrow$ ADDI X0, X0, #1

Por lo tanto X0 = 0x91000401 al final del programa.