Ejercicio 1

X0 : f = 1

X1 : g = 2

X2 : h = 3

X3 : i = 4

X4 : j = 5

a) f = g + h + i + j ;

```
ADD X0, X1, X2    // f = 2+3 = 5
ADD X0, X0, X3    // f = (2+3)+4 = 9
ADD X0, X0, X4    // f = ((2+3)+4) + 5 = 14
```

b) f = g + (h + 5)

```
ADDI X0, X2, #5   // f = 3+5 = 8
ADD X0, X1, X0    // f = 2 + (3+5) = 10
```

c) f = (g+h) + (g+h)

```
ADD X0, X1, X2    // f = 2 + 3 = 5
ADD X0, X0, X0    // f = (2+3) + (2+3) = 10
```

Ejercicio 2

X0 : f = 1

X1 : g = 2

X2 : h = 3

a) ADD X0, X1, X2

```
f = g + h    // f = 2+3 = 5
```

b) ADDI X0, X0, #1

```
ADD X0, X1, X2
f = f + 1    // f = 1+1 = 2
f = g + h    // f = 2+3 = 5
```

Ejercicio 3

X0 : f = 4

X1 : g = 5

a) f = -g - f ; ⇝ f = -(g+f)

```
ADD X0, X1, X0    // f = 5+4 = 9
SUB X0, XZR, X0   // f = 0-(5+4) = -9
```

b) f = g + (-f-5) ⇝ f = g - (f+5)

```
ADDI X0, X0, #5   // f = 4+5 = 9
SUB X0, X1, X0    // f = 5-(4+5) = 5-9 = -4
```

Ejercicio 4

X0 : f = 1

X1 : g = 2

X2 : h = 3

a) SUB X1, XZR, X1

ADD X0, X1, X2

---

g = - g // g = -2

f = - g + h // f = -2 + 3 = 1

b) ADDI X2, X0, #1

SUB X0, X1, X2

---

h = f + 1 // h = 1+1 = 2

f = g - (f+1) // f = 2 - (1+1) = 0

Ejercicio 5

X0 : f

X1 : g

X2 : i

X3 : j

X6: Dirección base de los arreglos A

X7: Dirección base de los arreglos B

a) f = -g - A[4];

LDUR X2, [X6, #32] // i = A[4]

SUB X0, XZR, X1 // f = -g

SUB X0, X0, X4 // f = -g - A[4]

b) B[8] = A[i-j]

SUB X0, X2, X3 // f = i - j

LSL X0, X0, #3 // f = (i-j) * 2^3

ADD X0, X6, X0 // f = A + [(i-j)*8]

LDUR X0, [X0, #0] // f = A[i-j]

STUR X0, [X7, #64] // B[8] = A[i-j]

5.2) Se utilizan 6 registros (GPRs): X0, X1, X2, X3, X6, X7.

Ejercicio 6

X0 : f

X1 : g

X2 : h

X3 : i

X4 : j

X6: Dirección base de los arreglos A

X7: Dirección base de los arreglos B

Registros temporales:

X9, X10, X11, X12

a)

LSL X2, X4, #1	h = j * 2;	⇝ h = 2j	h = j * 2;
ADD X0, X2, X4	f = (j * 2) + j;	⇝ f = h + j	f = h + j;
ADD X0, X0, X4	f = [(j * 2) + j] + j;	⇝ f = (h+j) + j f = f + j	f += j;

b)

LSL X9, X3, #3	x9 = i * 8;	→ Calcula el desplazamiento en bytes para el indice i.
ADD X9, X6, X9	x9 = &A[0] + (i*8);	→ Calcula la dirección A[i]. X9 ahora contiene &A[i]
LSL X10, X4, #3	X10 = j * 8;	→ Calcula el desplazamiento en bytes para el indice j.
ADD X10, X7, X10	X10 = &B[0] + (j*8);	→ Calcula la dirección B[j]. X10 ahora contiene &B[j]
LDUR X12, [X9, #0]	X12 = A[i];	→ Carga en X12 el valor que está en la dirección de memoria X9 (&A[i])
ADDI X11, X9, #8	X11 = &A[i] + 8	→ Calcula la dirección del siguiente elemento, &A[i+1]
LDUR X9, [X11, #0]	X9 = A[i+1];	→ Carga en X9 el valor que está en la dirección de memoria X11 (&A[i+1]).
ADD X9, X9, X12	X9 = A[i+1] + A[i];	→ El X9 actual es A[i+1]. X12 es A[i]
STUR X9, [X10, #0]	B[j] = A[i+1] + A[i];	→ Almacena el valor X9 en la dirección de memoria X10 (&B[j])

Ejercicio 7

X0 : f X6 : Dirección base del arreglo A (&A[0])

7.1)

ADDI X9, X6, #8	X9 = &A[0] + 8;	⇝ X9 = &A[1]
ADD X10, X6, XZR	X10 = &A[0] + 0;	⇝ X10 = &A[0]
STUR X10, [X9, #0]	A[1] = A[0];	
LDUR X9, [X9, #0]	X9 = &A[1];	
ADD X0, X9, X10	f = A[1] + A[0]	⇝ f = A[0] + A[0]

7.2)

Dirección	Valor
0x100	0x64
0x108	0xC8
0x110	0x12C

Secuencia paso a paso:

1. X9 = 0x108 , X6 = 0x100
2. X10 = 0x100 , X6 = 0x100
3. X9 = 0x100 ⇝ 0x108 = 0x100
4. X9 = 0x100
5. X0 = 0x100 + 0x100 = 0x200

Ejercicio 8

8.1) a) X9 = 0x 0000000055555555 , X10 = 0x 0000000012345678

LSL X11, X9, #4 → X11 = 0x 0000000555555550

ORR X11, X11, X10 → X11 = 0101 0101 0101 0101 0101 0101 0101 0101 0000

X10 = 0000 0001 0010 0011 0100 0101 0110 0111 1000

0101 0101 0111 0111 0101 0101 0111 0111 1000
5 5 7 7 5 5 7 7 8

X11 = 0x 0000000 55 77 55 78 ↓

b) X9 = 0x 00000000 AAAAAAAA , X10 = 0x 12345678 12345678

LSL X11, X9, #4 → X11 = 0x 0000000A AAAAAAA0

ORR X11, X11, X10 → X11 = 1010 1010 1010 1010 1010 1010 1010 1010 0000

"X10" = 1000 0001 0010 0011 0100 0101 0110 0111 1000

(faltan los primeros 7 bits de X10)

1010 1011 1010 1011 1110 1111 1110 1111 1000
A B A B E F E F 8

X11 = 0x 0000000 ABABEFEF8 ↓

8.2) a) X9 = 0x 0000000055555555 , X10 = 0x 0000000012345678

LSL X11, X10, #4 → X11 = 0x 0000000123456780

AND1 X11, X11, # 0xFFF →

0001 0010 0011 0100 0101 0110 0111 1000 0000

0000 0000 0000 0000 0000 0000 1111 1111 1111

0000 0000 0000 0000 0000 0000 0111 1000 0000 X11 = 0x 780 ↓

b) X9 = 0x 00000000 AAAAAAAA , X10 = 0x 1234567812345678

LSL X11, X10, #4 → X11 = 0x 2345678123456780

AND1 X11, X11, # 0xFFF →

0111 1000 0000

1111 1111 1111

0111 1000 0000 X11 = 0x 780 ↓

8.3) a) X9 = 0x 0000000055555555 , X10 = 0x 0000000012345678

LSR X11, X9, #3 → X9 = 0101 0101 0101 0101 0101 0101 0101 0101

(elimino estos 3 últimos bits y agrego 3 ceros al comienzo)

Entonces X11 = 0000 1010 1010 1010 1010 1010 1010 1010

0101 0101 0101

0000 0000 0000 X11 = 0x0 ↓

b) X9 = 0x00000000AAAAAAAA , X10 = 0x1234567812345678

LSR X11, X9, #3 → X9 = 1010 1010 1010 1010 1010 1010 1010 1010 (elimino estos 3 últimos bits y agrego 3 ceros al comienzo)

AND1 X11, X11, #0x555 Entonces X11 = 0001 0101 0101 0101 0101 0101 0101 0101

0101 0101 0101

0101 0101 0101 X11 = 0x555 ↓

Ejercicio 9

EXCEPTION SYNDROME REGISTER (ESR)

Exception Class (EC)	Instruction Length (IL)	Instruction Specific Syndrome field (ISS)
31 — 26	25 — 24	0

LSR X10, X9, #26

Ejercicio 10

X9 es un número entero representado en complemento a 2 y es negativo.

Entonces si el número es negativo empieza en 1. Para que X10 devuelva un 1 entonces usemos el 1 del X9.

Esto se podría lograr haciendo un desplazamiento de bits.

LSR X10, X9, #63

Ejercicio 11

48 32 16 0

10.1) {X0 = 0x1234000000000000}
10.2) {X1 = 0xBBB0000000000AAA}
10.3) {X2 = 0xA0A0B1B10000C2C2}
10.4) {X3 = 0x0123456789ABCDEF}

10.1) MOVZ X0, 0x1234, LSL 48

10.2) MOVZ X1, 0xBBB0, LSL 48

MOVZ X1, 0x0AAA, LSL 0

10.3) MOVZ X2, 0xA0A0, LSL 48

MOVZ X2, 0xB1B1, LSL 32

MOVZ X2, 0xC2C2, LSL 0

10.4) MOVZ X3, 0x0123, LSL 48

MOVZ X3, 0x4567, LSL 32

MOVZ X3, 0x89AB, LSL 16

MOVZ X3, 0xCDEF, LSL 0

Ejercicio 12

Ejercicio 12:
Suponiendo que el microprocesador LEGv8 está configurado en modo LE *little-endian*, decir que valores toman los registros X0 a X7 al terminar este programa.

```
MOVZ X9, 0xCDEF, LSL 0
MOVK X9, 0x89AB, LSL 16
MOVK X9, 0x4567, LSL 32
MOVK X9, 0x0123, LSL 48
STUR X9, [XZR, #0]
LDURB X0, [XZR, #0]
.
.
.
LDURB X7, [XZR, #7]
```

¿Qué valores toman los registros X0 a X7 si el microprocesador LEGv8 está configurado en modo BE *big-endian*?

X9 = 0x 0123456789 ABCDEF

XZR = 0x 0123456789 ABCDEF

Little - endian :	Big - endian :
X0 = 0x EF (LSB)	X0 = 0x 01 (MSB)
X1 = 0x CD	X1 = 0x 23
X2 = 0x AB	X2 = 0x 45
X3 = 0x 89	X3 = 0x 67
X4 = 0x 67	X4 = 0x 89
X5 = 0x 45	X5 = 0x AB
X6 = 0x 23	X6 = 0x CD
X7 = 0x 01 (MSB)	X7 = 0x EF (LSB)