

PRÁCTICO 7 - Assembler de LEGv8 avanzado

	Signed numbers		Unsigned numbers	
Comparison	Instruction	CC Test	Instruction	CC Test
=	B.EQ	Z=1	B.EQ	Z=1
≠	B.NE	Z=0	B.NE	Z=0
<	B.LT	N!=V	B.LO	C=0
≤	B.LE	~(Z=0 & N=V)	B.LS	~(Z=0 & C=1)
>	B.GT	(Z=0 & N=V)	B.HI	(Z=0 & C=1)
≥	B.GE	N=V	B.HS	C=1

Signed and Unsigned numbers	
Instruction	CC Test
Branch on minus (B.MI)	N= 1
Branch on plus (B.PL)	N= 0
Branch on overflow set (B.VS)	V= 1
Branch on overflow clear (B.VC)	V= 0

- Negative (N): the result that set the condition code had a 1 in the most significant bit.
- Zero (Z): the result that set the condition code was 0.
- Overflow (V): the result that set the condition code overflowed. Signed numbers.
- Carry (C): the result that set the condition code had a carry out of the most significant bit or a borrow into the most significant bit. Unsigned numbers.

Operation	Operand A	Operand B	Result indicating overflow
$A + B$	≥ 0	≥ 0	< 0
$A + B$	< 0	< 0	≥ 0
$A - B$	≥ 0	< 0	< 0
$A - B$	< 0	≥ 0	≥ 0

Ubicación de las flags en el registro SPSR (En el dashboard mostrado como CPSR) [ref](#)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
N	Z	C	V							SS	IL											D	A	I	F		M			M [3:0]	

Ejercicio 1:

Para estos dos programas con entrada y salida en X0, decir que función realizan.

<pre> SUBIS X0, X0, #0 B.LT else B done else: SUB X0, XZR, X0 done: </pre>	<pre> MOV X9, X0 MOV X0, XZR loop: ADD X0, X0, X9 SUBI X9, X9, #1 CBNZ X9, loop done: </pre>
--	--

Ejercicio 2:

Dado el siguiente programa LEGv8, dar el valor final de X10, dado que inicialmente {X10=0x0000000000000001}.

```

        SUBIS XZR, X9, #0
        B.GE else
        B done
    else: ORRI X10, XZR, #2
    done:

```

1.1) Dado que inicialmente {X9=0x00000000000101000}.

1.2) Dado que inicialmente {X9=0x800000000001000}.

Ejercicio 3:

Dado el siguiente programa “C” y la asignación $i, j, k, N \leftrightarrow X0, X1, X2, X9$, escribir el programa LEGv8 que lo implementa. Notar que como se usa el operador `||` la [evaluación es por cortocircuito](#). **Opcional:** hacerlo con el operador `|` que no está cortocircuitado.

```

long i,j,k;
if (i==N || j==N) {
    ++k;
} else {
    ++i; ++j;
}

```

Ejercicio 4:

Dados los siguientes programas LEGv8:

<pre> loop: ADDI X0, X0, #2 SUBI X1, X1, #1 CBNZ X1, loop done: </pre>	<pre> loop: SUBIS X1, X1, #0 B.LE done SUBI X1, X1, #1 ADDI X0, X0, #2 B loop done: </pre>
--	--

- 4.1) Dar los valores finales de X0, teniendo en cuenta que inicialmente vale {X0=0, X1=10}.
- 4.2) Dada la asignación $X0, X1 \leftrightarrow \text{acc}, i$, escribir el programa “C” equivalente donde todas las variables son de tipo long.
- 4.3) Dado que inicialmente {X1=N} ¿Cuántas instrucciones LEGv8 se ejecutan?
- 4.4) Para el programa de la derecha. Si reemplazamos B.LE done por B.MI done ¿Cuál es el valor final de X0 suponiendo que inicialmente {X0=0}?
- 4.5) Dada la asignación $X0, X1 \leftrightarrow \text{acc}, i$, escribir el programa “C” equivalente del punto “4.4”, donde todas las variables son de tipo long.
- 4.6) **Opcional:** Mostrar que se puede reducir el número de instrucciones ejecutadas en el programa de la derecha, combinando SUBIS y SUBI. Ayuda: agregar una instrucción por fuera del lazo. Ayuda: es lo mejor de los dos mundos ;)

Ejercicio 5:

Dados los siguientes programas en LEGv8:

<pre> ADD X10, XZR, XZR loop: LDUR X1, [X0,#0] ADD X2, X2, X1 ADDI X0, X0, #8 ADDI X10, X10, #1 CMPI X10, #100 B.LT loop </pre>	<pre> ADDI X10, XZR, #50 loop: LDUR X1, [X0,#0] ADD X2, X2, X1 LDUR X1, [X0,#8] ADD X2, X2, X1 ADDI X0, X0, #16 SUBI X10, X10, #1 CBNZ X10, loop </pre>
---	---

5.1) ¿Cuántas instrucciones LEGv8 ejecuta cada uno?

5.2) Reescribir en “C” dada la asignación $X10, X1, X2, X0 \leftrightarrow i, a, result, MemArray$.

5.3) Opcional: optimizar los códigos assembler para reducir el número de instrucciones LEGv8 ejecutadas.

Ejercicio 6:

Traducir el siguiente programa en “C” a ensamblador LEGv8 dada la asignación de variables a registros $X0, X1, X2, X9 \leftrightarrow str, found, i, N$. El número 48 se corresponde con el carácter ‘0’ en ASCII, por lo tanto el programa cuenta la cantidad de ‘0’s que aparecen en una cadena de caracteres de longitud N.

```

#define N (1<<10)
char *str;
long found, i;
for (found=0, i=0; i!=N; ++i)
    found += (str[i]==48);

```

Ejercicio 7:

Traducir el siguiente programa “C” a LEGv8. La asignación de variables a registros $X0, X1, X2, X3, X9 \leftrightarrow A, s, i, j, N$. Notar que en “C” los arreglos bidimensionales se representan en memoria usando un **orden por filas**, es decir $\&A[i][j] = A + 8*(i*N+j)$.

```

#define N (1<<10)
long A[N][N], s, i, j;
s=0;
for (i=0; i<N; ++i)
    for (j=0; j<N; ++j)
        s += A[i][j];

```

Ejemplo de orden por filas:

$A[3][3] =$

1	7	2
44	3	21
1	2	3

$A \rightarrow$

1	7	2	44	3	21	1	2	3
---	---	---	----	---	----	---	---	---

Opcionales:

7.1) Hacer lineal el acceso al arreglo y recorrerlo con un solo lazo.

7.2) Se puede hacer lo mismo sin usar ninguna variable índice i , j .

Ejercicio 8:

Mostrar cómo se implementarían las siguientes **pseudoinstrucciones** con la mínima cantidad de instrucciones LEGv8, pudiendo usar el registro X9 para almacenar valores temporales.

Nemónico	Operación	Semántica
CMP	comparación	FLAGS = $R[Rn] - R[Rm]$
CMPI	comparación con operando inmediato	FLAGS = $R[Rn] - \text{ALUImm}$
MOV	copia de valores entre registros	$R[Rd] = R[Rn]$
NOP	no-operación, el skip de LEGv8	
NOT	operador lógico de negación bit a bit	$R[Rd] = \sim R[Rn]$

Ejercicio 9:

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]
```

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

Ejercicio 10:

Verificar las implementaciones de los ejercicios con la combinación `as+qemu+gdb`. Incluyendo los ejercicios del práctico 6.

Opcionalmente emitir código ARMv8 desde "C" con `gcc`:

```
echo "long foo(long a) {return a*16;}" | aarch64-linux-gnu-gcc -O2 -S -o- -xc -
```