

LEGv8 avanzado

parte 1

OdC - 2020

Instrucciones para tomar decisiones

Podemos separar las instrucciones de saltos condicionales en dos grandes grupos:

- **CBZ / CBNZ:** El salto se efectúa o no, dependiendo del contenido de un registro que se pasa como argumento.
- **B.cond:** El salto se efectúa o no dependiendo del estado de las banderas del procesador.

CBZ: Compare and Branch if Zero

CBZ: El salto se realiza si el registro es cero

CBZ register, L1

```
        CBZ    x0, label
        ADDI   x1, x1, #8
label:   SUBI   x0, x0, #1
```

```
label:  SUBI   x0, x0, #1
        ADDI   x1, x1, #8
        CBZ    x0, label
```

CBNZ: Compare and Branch if Not Zero

CBNZ: El salto se realiza si el registro NO es cero

`CBNZ register, L1`

```
    CBNZ x0, label
    ADDI x1, x1, #8
label: SUBI x0, x0, #1
```

```
label: SUBI x0, x0, #1
    ADDI x1, x1, #8
    CBNZ x0, label
```

Ejercicio 2 - b

MOV X9, X0	// X9 = X0 (Se copia el valor de X0 a X9)
MOV X0, XZR	// X0 = 0 (Se copia el valor de XZR a X0)
loop: ADD X0, X0, X9	// x0 = x0 + x9
SUBI X9, X9, #1	// x9 = x9 - 1 (Resto 1 a x9)
CBNZ X9, loop	// Si x9 NO es cero, salto a las instrucción add
done:	

Ejercicio 2 - b

```
MOV  X9, X0          // X9 = X0 (Se copia el valor de X0 a X9)
MOV  X0, XZR          // X0 = 0 (Se copia el valor de XZR a X0)*
loop: ADD X0, X0, X9   // x0 = x0 + x9
      SUBI X9, X9, #1  // x9 = x9 - 1 (Resto 1 a x9)
      CBNZ X9, loop    // Si x9 NO es cero, salto a las instrucción add
done:
```

Si inicialmente $X0 = 10$

Iteración	0*	1	2	3	4	5	6	7	8	9
Valor de X9	10	9	8	7	6	5	4	3	2	1
Valor de X0	0	10	19	27	34	40	45	49	52	55

$$X0 = 10 + 9 + 8 + 7 + 6 + 5 + 4 + 3 + 2 + 1 = 55$$

$$X0 = X0 (X0 + 1) / 2$$

B.cond: Branch if condition

B.cond *label*

¿Y donde estan los registros que estoy comparando?

1

Condiciones:

- = Igual
- \neq Distinto
- > Mayor que
- \geq Mayor o igual que
- < Menor que
- \leq Menor o igual que

¿Como distingo si los números que estoy comparando son o no signado?

2

1 Comparación

- La comparación debe realizarse antes de ejecutarse la instrucción de salto
- Las únicas instrucciones que pueden utilizarse para la comparación son:
 - ADDS: ADD and set flags $R[Rd], \text{FLAGS} = R[Rn] + R[Rm]$
 - ADDIS: ADD Immediate and set flags $R[Rd], \text{FLAGS} = R[Rn] + \text{ALUImm}$
 - ANDS: AND and set flags
 - ANDIS: AND Immediate and set flags
 - SUBS: SUB and set flags
 - SUBIS: SUB Immediate and set flags
 - CMP: CoMPare $\text{FLAGS} = R[Rn] - R[Rm]$
 - CMPI: CoMpare Immediate $\text{FLAGS} = R[Rn] - \text{ALUImm}$

1 FLAGS

Banderas que reflejan algunas características particulares del resultado de una operación. Estas valen 1 cuando:

- **Negative (N)**: El resultado es negativo
- **Zero (Z)**: El resultado es cero
- **Carry (C)**: Existe un carry de salida o de entrada del bit más significativo.
(Para números **no signados**)
- **Overflow (V)**: El resultado de una operación **signada** genera overflow

El resultado se almacena en el registro CPSR (Current Program Status Register)

Bit 31	Bit 30	Bit 29	Bit 28	...
Negative	Zero	Carry	Overflow	...

1 Carry (Unsigned)

Las reglas para que la bandera de carry sea uno, son:

1. La suma de dos números genera un carry de salida del bit más significativo.

$$1111 + 0001 = 0000$$

2. La resta de dos números requiere un préstamo (carry in) en los bits más significativos restados.

$$0000 - 0001 = 1111$$

En cualquier otro caso, la bandera es cero:

$$0111 + 0001 = 1000$$

$$1000 - 0001 = 0111$$

1 Overflow (Signed)

Las reglas para que la bandera de overflow sea uno, son:

1. La suma de dos números positivos da como resultado un número negativo
 $0100 + 0100 = 1000$
2. La suma de dos números negativos da como resultado un número positivo
 $1000 + 1000 = 0000$

En cualquier otro caso, la bandera es cero:

$$\begin{aligned}0100 + 0001 &= 0101 \\0110 + 1001 &= 1111 \\1000 + 0001 &= 1001 \\1100 + 1100 &= 1000\end{aligned}$$

1 Overflow (Signed)

Operación	Operando A	Operando B	Resultado que genera Overflow
A + B	≥ 0	≥ 0	< 0
A + B	< 0	< 0	≥ 0
A - B	≥ 0	< 0	< 0
A - B	< 0	≥ 0	≥ 0

2 Instrucciones de salto

	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	$\sim(Z=0 \ \& \ N=V)$	B.LS	$\sim(Z=0 \ \& \ C=1)$
>	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

Ejercicio 2 a

```
        SUBS XZR, XZR, X0      // FLAGS = 0 - X0 (CMPI X0, 0)
        B.LT else             // Si X0 es menor que 0 salto a else
        B done                // Salto incondicional a done
else: SUB X0, XZR, X0          // X0 = 0 - X0
done:
```

- El salto condicional se toma si el valor almacenado en X0 es menor que cero, es decir, es negativo.
- En este caso, antes de finalizar se ejecuta la instrucción SUB donde se cambia el signo de X0
- Si no se toma el salto condicional, la instrucción *branch* me lleva directamente al fin del programa.

Este programa devuelve el **módulo** del valor almacenado en X0

Ejercicio 1

```
.data
a: .dword 0x0000000000000001
b: .dword 0x0000000000010100    //b: .dword 0x8000000000000100
.text
        SUBIS XZR, X9, #0        // FLAGS = x9 - 0
        B.GE else                // Salto a "else" X9 es mayor a cero
        B done                   // Salto incondicional a "done"
else:    ORRI X10, XZR, #2        // X10 = 0 || 2
done:
```

- El salto condicional depende del valor de X9, si este es mayor a cero se toma el salto.
- El valor de X10 depende de si el salto fue tomado o no.

Ejercicio 4

```
loop:    ADDI X0, X0, #2    // X0 = X0 + 2
        SUBI X1, X1, #1    // X1 = X1 - 1
        CBNZ X1, loop      // Si X1 no es cero, salto a loop
```

done:

```
loop:    SUBIS X1, X1, #0    // X1 = X1 - 0 (Seteo de flags)
        B.LE done           // Si X1 es menor o igual que 0 salto a done
        SUBI X1, X1, #1      // X1 = X1 + 1
        ADDI X0, X0, #2      // X0 = X0 + 2
        B loop              // Salto incondicional a loop
```

done:

```
for (long i=N; 0<=i; --i)
    acc+=2;
```


Ejercicio 5 - a

X10 \leftrightarrow i
X1 \leftrightarrow a
X2 \leftrightarrow result
X0 \leftrightarrow &MemArray[0].

```
i = 0;
do
{
    result += MemArray[i];
    i++;
}
while (i<100)
```

```
ADDI X10, XZR, 100 // i = 0
loop: LDUR X1, [X0,#0] // a = MemArray[0]
      ADD X2, X2, X1 // result += a
      ADDI X0, X0, #8 // X0 = &MemArray[0] + 8 = &MemArray[1]
      SUBI X10, X10, #1 // i += 1
      //CMPI X10, #100 // FLAGS = X10 - 100
      cbnz loop // goto loop si X10 < 100
```

Ejercicio 5 - b

X10 \leftrightarrow i
X1 \leftrightarrow a
X2 \leftrightarrow result
X0 \leftrightarrow &MemArray[0].

```
i = 50;
do
{
    result += MemArray[100 - i*2];
    result += MemArray[100 - i*2 + 1];
    i--;
}
while (i!=0)
```

```
loop:    ADDI X10, XZR, #50 // i = 50
        LDUR X1, [X0,#0]  // a = MemArray[0]
        ADD X2, X2, X1    // result += a
        LDUR X1, [X0,#8]  // a = MemArray[1]
        ADD X2, X2, X1    // result += a
        ADDI X0, X0, #16  // x0 = x0 + 16 = &MemArray[2]
        SUBI X10, X10, #1 // i -= 1
        CBNZ X10, loop    // salta si x10 != 0
```