ASSEMBLER DE LEGV8 AVANZADO

FLAGS DE CONDICIÓN EN ARMv8

SON BITS ESPECIFICOS EN EL REGISTRO DE ESTADO DEL PROCESADOR (PSTATE) QUE INDICAN EL RESULTADO DE OPERACIONES ARITMETICAS Y LÓGICAS

- **N (NEGATIVE)**
  - SE PONE EN 1, SI EL RES. DE LA OP. ES NEGATIVO
- **Z (ZERO)**
  - SE ESTABLECE (SE PONE EN 1) SI EL RESULTADO DE LA OPERACIÓN ES 0
- **C (CARRY)**
  - SE ESTABLECE SI HUBO UN ACARREO EN UNA OPERACIÓN DE ADICIÓN O UN "PRESTAMO" (BORROW) EN UNA RESTA.
- **V (OVERFLOW)**
  - SE ESTABLECE SI HUBO UN DESBORDAMIENTO (OVERFLOW) EN UNA OPERACIÓN ARITMETICA. (SI EL RES. ES MUY GRANDE O MUY CHICO PARA EL TIPO DE DATOS USADO)

INSTRUCCIONES QUE ACTUALIZAN LOS FLAGS

LAS INSTRUCCIONES QUE TERMINAN EN "S" ACTUALIZAN ESTOS FLAGS

EJEMPLOS: ADDIS, ADDS, ANDIS, ANDS, SUBIS, SUBS

TIENEN EL MISMO FORMATO PARA ESCRIBIRLOS SOLO QUE CON LA "S" AL FINAL.

USO DE LOS FLAGS EN LAS INSTRUCCIONES DE SALTO CONDICIONAL

LAS INSTRUCCIONES DE SALTO CONDICIONAL UTILIZAN LOS FLAGS DE CONDICIÓN PARA DECIDIR SI DEBEN REALIZAR EL SALTO.

LA INTERPRETACIÓN DE LOS FLAGS ES DIFERENTE SI SE TRATA DE NÚMEROS CON SIGNO O SIN SIGNO.

COMPARACIÓN	SIGNED INSTRUCCION	NUMBERS CC TEST	UNSIGNED INSTRUCCIÓN	NUMBERS CC TEST
$=$	B.EQ	$Z=1$	B.EQ	$Z=1$
$\neq$	B.NE	$Z=0$	B.NE	$Z=0$
$<$	B.LT	$N \neq V$	B.LO	$C=0$
$\leq$	B.LE	$\neg(Z=0 \wedge N=V)$	B.LS	$\neg(Z=0 \wedge C=1)$
$>$	B.GT	$Z=0 \wedge N=V$	B.Hi	$Z=0 \wedge C=1$
$\geq$	B.GE	$N=V$	B.HS	$C=1$

EXPLICACIÓN DE LOS NOMBRES:

- GT → GREATER THAN
- NE → NOT EQUAL •
- EQ → EQUAL •
- LT → LESS THAN
- LE → LESS OR EQUAL
- GE → GREATER OR EQUAL
- LO → LOWER
- LS → LOWER OR SAME
- Hi → HIGHER
- HS → HIGER OR SAME

• INDICA QUE SE USA PARA NUM. CON SIGNOS O SIN SIGNOS POR IGUAL

(faltan otras instrucciones explicadas en la hoja 3)

Papel de fibra de caña de azúcar

EJERCICIO 1

a) SUBIS $X_0, X_0, \#0$ // $X_0 = X_0 - 0$ (ADEMÁS ACTUALIZA LOS FLAGS)
B.LT else // SI EL RESULTADO ANTERIOR ES < 0 SALTA A ELSE
B done // SALTA A DONE INCONDICIONALMENTE
else: SUB X_0, XZR, X_0 // $X_0 = 0 - X_0 = -X_0$
done: // SIGUE CON EL CÓDIGO

GUARDA EN X_0, EL VALOR ABSOLUTO DE X_0 (EXPLICACIÓN)

b) MOV X_9, X_0 // COPIA EL VALOR DEL REGISTRO X_0 AL REGISTRO X_9, $X_9 = X_0$
MOV X_0, XZR // COPIA EL VALOR DEL REGISTRO XZR AL REGISTRO X_0 $X_0 = 0$
LOOP: ADD X_0, X_0, X_9 // $X_0 = X_0 + X_9$
SUBI $X_9, X_9, \#1$ // $X_9 = X_9 - 1$ (Y ACTUALIZA LOS FLAGS)
CBNZ X_9, LOOP // SI $X_9 \neq 0$ SALTA A LOOP
DONE:

GUARDA EN X_0, LA SUMA DE LOS PRIMEROS X_0 VALOR INICIAL NÚMEROS. (EXPLICACIÓN)

CBNZ
COMPARE AND BRANCH IF NOT ZERO
COMPARA EL CONTENIDO DE UN REGISTRO CON 0 Y REALIZA UN SALTO SI EL VALOR DEL REGISTRO NO ES 0

CBNZ registro label
ejemplo CBNZ X0, LOOP
SI EL CONTENIDO DEL REGISTRO X_0 NO ES 0, SALTA A LOOP.
SI EL CONTENIDO = 0 LA EJECUCIÓN CONTINUA CON LA SIGUIENTE INSTRUCCIÓN SECUENCIALMENTE

CBZ (COMPARE AND BRANCH IF ZERO)
COMPARA EL CONTENIDO DE UN REGISTRO CON 0 Y REALIZA UN SALTO SI EL VALOR DEL REGISTRO ES CERO

EJERCICIO 2

(2.1) $X_{10} = 0x0000000000000001$
$X_9 = 0x0000000000101000$

SUBIS XZR, X_9, #0
B.GE else
B.DONE
else: ORRI X_{10}, XZR, #2
done:
$X_{10} = 0x0000000000000002$

(2.2) $X_9 = 0x8000000000001000$

$X_{10} = 0x0000000000000001$

EJERCICIO 3

LONG i, J, K

IF (i==N || J==N){
++K;
else {
++i; ++J;
}

i = X_0, J = X_1, K = X_2, N = X_9

SUBIS XZR, $X_0^{(i)}$, $X_9^{(N)}$ // RESTO i CON N Y ACTUALIZO LOS FLAGS
B.EQ else // SI SON IGUALES SALTO A else
SUBIS XZR, $X_1^{(J)}$, $X_9^{(N)}$ // RESTO J CON N Y ACT LOS FLAGS
B.EQ else // SI SON IGUALES SALTO A else
ADDI $X_0, X_0, \#1$ // i++
ADDI $X_1, X_1, \#1$ // J++
B end
else: ADDI $X_2, X_2, \#1$ // K++
end:

(HAY MUCHAS FORMAS DE HACERLO)

ORGANIZACIÓN DEL COMPUTADOR

EJERCICIO 4

a)
```
LOOP: ADDI X0, X0, #2
      SUBI X1, X1, #1
      CBNZ X1, LOOP
done:
```

(4.1) $X_0 = 0$, $X_1 = 10$

1) $X_0 = 2$, $X_1 = 9$ 2) $X_0 = 4$, $X_1 = 8$ 3) $X_0 = 6$, $X_1 = 7$

... SIGUIENDO ESTO VEMOS QUE AL TERMINAR $X_0 = 20$

(4.2) $X_0 \rightarrow$ acc, $X_1 \rightarrow$ i

```
long acc = 0;
long i = 10;
while (i != 0){
  acc += 2;
  i--;
}
```

(4.3) $X_1 = N$

DADO QUE EN CADA LOOP EJECUTAMOS 3 INSTRUCCIONES Y EL LOOP SE VA A REPETIR HASTA QUE $N \neq 0$ Y N. VA DISMINUYENDO EN 1, SE EJECUTAN $3*N$ INSTRUCCIONES

(4.4) -
(4.5) - EJERCICIOS PARA EL PROGRAMA b) :) :) :D D:
(4.6) -

INSTRUCCIONES DE SALTO CONDICIONALES PARA NÚMEROS CON Y SIN SIGNO

- B.Mi label (MINUS)
  SALTA A LA ETIQUETA label SI EL RESULTADO ES NEGATIVO (N=1)
- B.Pl label (PLUS)
  REALIZA EL SALTO SI ES ≥ 0 (N=0)
- B.VS label (OVERFLOW SET)
  SALTA SI OCURRIÓ UN DESBORDAMIENTO (V=1)
- B.VC label (OVERFLOW CLEAN)
  SALTO SI NO OCURRIÓ UN DESBORDAMIENTO (V=0)

b)
```
LOOP: SUBIS X1, X1, #0
      B.LE done
      SUBI X1, X1, #1
      ADDI X0, X0, #2
      B LOOP
done:
```

(4.1) $X_0 = 0$, $X_1 = 10$

1) $X_1 = 10$, $X_0 = 2$ 2) $X_1 = 9$, $X_0 = 4$ 3) $X_1 = 8$, $X_0 = 6$

... SIGUIENDO ESTE PATRON OBSERVAMOS QUE AL IGUAL QUE EN a) X0 AL TERMINA VALE 20

(4.2) $X_0 \rightarrow$ acc, $X_1 \rightarrow$ i

```
long acc = 0, i = 10;
while (true){
  if (i<0) Break
  i--; acc+=2;
}
```

(4.3) $X_1 = N$

VEMOS QUE LOOP REALIZA 4 INSTRUCCIONES, Y EL LOOP SE VA A EJECUTAR N VECES YA QUE X_1 SE DECREMENTA EN 1 HASTA QUE SEA < 0, => SE EJECUTAN $4*N$ INSTRUCCIONES

(4.4) B.LE → B.Mi done
EL VALOR FINAL DE $X_0 = 22$ YA QUE VA A REALIZAR UN RECORRIDO MÁS PORQUE ANTES PARABA CUANDO ERA =0 Y AHORA CUANDO ES <0.

(4.5)
```
long acc = 0, i = 10
while (true){
  if(i<0) break
  i--; acc+=2;
}
```

(4.6) ???

Papel de fibra de caña de azúcar.

Ejercicio 5

a)

```
LOOP: LDUR X1, [X0, #0]
      ADD X2, X2, X1
      ADDI X0, X0, #8
      ADDI X10, X10, #1
      CMPI X10, #100
      B.LT LOOP
```

5.1) SABEMOS QUE EN EL LOOP SE EJECUTAN 6 INSTRUCCIONES.
TAMBIEN VEMOS QUE X10 SE INICIALIZA CON 0 (XZR). VEMOS QUE X10 SE INCREMENTA EN 1 CADA VEZ QUE OCURRE EL LOOP.
EN SINTESIS SE VA A EJECUTAR MIENTRAS X10 = 0 < 100, ENTONCES SE EJECUTAN 100 LOOPS Y CADA UNO 6 INSTRUCCIONES
6 * 100 = 600 INSTRUCCIONES. NO NOS OLVIDEMOS DE LA PRIMERA DONDE "INICIALIZA A X10" ENTONCES SON 601

5.2) X10 = i, X1 = a, X2 = result, X0 = MemArray

```
long i = 0, result = 0, a;
long MemArray[100] // Capaz que inicializar
long *MemArrayPtr = MemArray //APUNTA AL
                             //INICIO DEL
While (i < 100){             //ARREGLO
  a = *MemArrayPtr // CARGAMOS EL VALOR APUNTADO
  result += a;
  MemArrayPtr += 1
  i++;
}
```

5.3) ??

b)

```
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
```

5.1) VEMOS QUE EN EL LOOP SE REALIZAN 7 INSTRUCCIONES.
TAMBIEN OBSERVAMOS QUE X10 "SE INICIALIZA" CON 50 Y EN CADA EJECUCIÓN DEL LOOP SE DISMINUYE EN 1.
GRACIAS AL CBNZ NOS DAMOS CUENTA QUE SE VA A EJECUTAR HASTA QUE EL CONTENIDO EN X10 = 0 Y PARA QUE OCURRA ESO DEBEMOS EJECUTAR 50 VECES EL BUCLE
50 * 7 = 350 + 1 = 351 SUMO 1 POR LA INSTRUCCIÓN DE INICIALIZACIÓN

5.2)

```
long i=50, result, a,
long MemArray[100],
long *MemArrayPtr = MemArray,
While (i ≠ 50){
  a = *MemArrayPtr,
  result += a;
  a = *(MemArrayPtr+1),
  result += a;
  MemArrayPtr += 2
  i--
}
```

5.3)