PRÁCTICO 6 | ORGANIZACIÓN DEL COMPUTADOR | ASSEMBLER V8 LEGV8 BÁSICO

ASSEMBLER DE LEGV8 (BÁSICO)

LEGV8 ES UNA ARQUITECTURA DE CONJUNTOS DE INSTRUCCIONES. DEFINE EL CONJUNTO DE INSTRUCCIONES Y LAS REGLAS DE FUNCIONAMIENTO QUE LOS PROCESADORES COMPATIBLES DE DGN SEGUIR.

EL ENSAMBLADOR PARA LEGV8 ES UN PROGRAMA QUE TRADUCE PROGRAMAS ESCRITOS EN LENGUAJE ENSAMBLADOR DE ARMV8 A CÓDIGO MAQUINA

OPERACIONES ARITMETICAS BÁSICAS

• ADD/SUB

SUMAR° RESTAR & REGISTROS Y ALMACENAR EL RESULTADO EN OTRO REGISTRO

ADD/SUB destino, origen1, origen2

EJ 1 → ADD R1, R2, R3 SUMA EL CONTENIDO DE R2 Y R3 Y ALMACENA EL CONTENIDO EN R1
EJ 2 → SUB R1, R2, R3 RESTA EL CONTENIDO DEL REGISTRO R3 DEL

EJ 1 - PRÁCTICO 6
c) $F = (g+h) + (g+h)$ (ESCRITO EN C) ESCRIBIR LA SECUENCIA MÍNIMA DE CÓDIGO ASSEMBLER LEGV8 ASUMIENDO QUE F, G, H, I, J SE ASIGNAN EN LOS REGS X0, X1, X2, X3, X4 Y DAR EL VALOR SI F=1, g=2, h=3, i=4, j=5

```
ADD X0, X1, X2 // F = 2+3 = 5
ADD X0, X0, X0 // F = 5+5 = 10
```

• ADDI/SUBI

SUMAR O RESTAR UN VALOR INMEDIATO (CONSTANTE) A UN REGISTRO Y ALMACENAR EL RESULTADO EN OTRO REGISTRO

ADDI/SUBI destino, origen, valor

EJ1 → ADDI R1, R2, #10 SUMA EL VALOR INMEDIATO '10' AL CONTENIDO DE R2 Y ALMACENA EN R1
EJ2 → SUBI R1, R2, #10 RESTA EL VALOR INMEDIATO '10' DEL CONTENIDO DE R2 Y ALMACENA EN R1

ACEPTA UN VALOR INMEDIATO COMO UNO DE SUS OPERANDOS, LO QUE PERMITE SUMAR O RESTAR UNA CONSTANTE DE UN REGISTRO

EJ 2 - PRACTICO 6
b)
```
ADDI X0, X0, #1 } F = F+1 // F = 1+1 = 2
ADD  X0, X1, X2 } F = g+h // F = 2+3 = 5
```

• XZR

ALIAS PARA EL REGISTRO GRAL. "0"

EJ 3 - PRÁCTICO 6
b) $F = g + (-f - 5)$;

```
SUB  X0, XZR, X0 // F = 0-F = -F => F = -4
SUBI X0, X0, #5  // F = -F-5 => F = -4-5 = -9
ADD  X0, X1, X0  // F = G + (-F-5) ⇒ F = 5+(-9) = -4
```

a) $F = -g - F$

```
SUB X0, XZR, X0 // F = 0-F = -F => F = -4
SUB X0, X0, X1  // F = -F-g => -4-5 = -9
```

PRÁCTICA EJ 4 - PRÁCTICO 6

a) SUB X1, XZR, X1
ADD X0, X1, X2

PRIMERO LO HAGO POR SEPARADO
g = 0 - g
F = g + h
F = h - g
g = -2
F = -2 + 3 = 1

b) ADDI X2, X0, #1
SUB X0, X1, X2

h = F + 1
F = g - h
F = g - (F+1) → F = 2 - (1+1) = 0

ACCEDIENDO A LA MEMORIA

LA MEMORIA ES UN GRAN ARRAY UNIDIMENSIONAL DONDE LA DIRECCIÓN ACTÚA COMO INDICE DE ESE ARRAY, COMENZANDO EN 0

ADDR	DATA
3	100
2	10
1	101
0	1

LA DIRECCIÓN DEL TERCER ELEMENTO ES 2.
EL CONTENIDO DE ESTE ES 10
↳ Ó VALOR DE LA MEMORIA

• INSTRUCCIÓN LOAD (CARGAR)

• LDUR (LOAD REGISTER UNSIGNED)

CARGAR UN VALOR DE LA MEMORIA EN UN REGISTRO

LDUR <registro de destino>, [<registro base>, <desplazamiento>]

- <registro de destino>: se almacenará el valor cargado
- <registro base>: contiene la dir. base de la memoria de la cual se cargarán los datos
- <desplazamiento>: ↳ valor opcional que especifica un valor adicional desde la dirección base

EJEMPLO GENERAL:

LDUR X1, [X2, #40] — COPIA AL REGISTRO X_1 EL CONTENIDO DE LA MEMORIA DIRECCIONADA POR EL CONTENIDO DEL REGISTRO X_2 SUM A #40

X1 = Memo[X2 + #40]

EJ 5 2) - PRÁCTICO 6

F = -g - A[4]

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

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

SUB X0, X0, X2 // F = -g - A[4]

SE UTILIZAN 4 REGISTROS (X_0, X_1, X_2, X_6)

• INSTRUCCIÓN STORE ("GUARDAR")

• STUR (STORE REGISTER UNSIGNED)

ALMACENAR UN VALOR CONTENIDO EN UN REGISTRO EN UNA DIRECCIÓN DE MEMORIA ESPECÍF.

STUR <registro fuente>, [<registro base>, <desplazamiento>]

- Registro que contiene el valor que se va almacenar en la memoria
- Registro que contiene la dir. base de la memo, donde se almace dicho valor
- Valor opcional que especifica un desplazamiento adicional desde la dirección base

EJEMPLO GENERAL

STUR X1, [X2, #40] — COPIA EL CONTENIDO DEL REGISTRO X_1 EN LA POS. DE MEMORIA DIRECCIONADA POR EL CONTENIDO DEL REG. X_2 SUMADO A LA CONSTANTE #40

memo[x2 + #40] = X1

Ej ⑤ b)

B[8] = A[i-j]

SUB X9, X2, X3 // x9 = i-j

LSL X9, x9, #3 // x9 = (i-j)*8

ADD X10, X6, X9 // x10 = A + [(i-j)*8]

LDUR x11, [x10, #0] // x11 = A[i-j]

STUR x11, [x7, #64] // B[8] = A[i-j]

• LSL (LOGICAL, SHIFT, LEFT)

- REALIZAR DESPLAZAMIENTOS LÓGICOS A LA IZQUIERDA EN EL CONTENIDO DE UN REGISTRO.
- MOVER UNA CIERTA CANTIDAD DE BITS HACIA LA IZQUIERDA

LSL destino, origen, cantidad de bits

- destino: DÓNDE SE ALMACENARÁ EL RESULTADO
- origen: REG. QUE CONTIENE EL VALOR A DESPLAZAR
- cantidad de bits: CANTIDAD DE POSICIONES A DESPLAZAR

// ejemplo multiplicar el contenido en R0 por 2

LSL R0, R0, #1

LSL X3, X0, #3 // $X_3 = X_0 * 2^3$

A CONTINUACIÓN DEJO ALGUNOS EJERCICIOS DEL PRÁCTICO 6 RESUELTOS QUE AYUDARÁN A ENTENDER MEJOR

Ej ⑥ a) (X_0 = F, X_1 = g, X_2 = h, X_3 = i, X_4 = j, X_6 = A, X_7 = B) b)

LSL X2, X4, #1 // h = j * 2

ADD X0, X2, X4 // F = (j*2) + j

ADD X0, X0, X4 // F = [(j*2)+j] + j

MINIMIZANDO:

F = 2j + 2j = 4j

SERÍA LO MISMO:

LSL X0, X4, #2

LSL X9, X3, #3 // x9 = i * 2^3 = 8i

ADD X9, X0, X9 // x9 = 8A + 8i = 8A[i]

LSL X10, X4, #3 // X10 = j * 2^3 = 8j

ADD X10, X7, X10 // X10 = 8B + 8j = 8B[j]

LDUR X12, [X9, #0] // X12 = A[i]

ADDI X11, X9, #8 // X11 = 8A[i] + 8 = 8A[i+1]

LDUR X9, [X11, #0] // X9 = A[i+1]

ADD X9, X9, X12 // X9 = A[i+1] + A[i]

STUR X9, [X10, #0] // B[j] = A[i+1] + A[i]

⑦ $X_0 = F$, $X6 = A$ $\quad X_0 = 0x A$, $X_6 = 0x100$

ADDI x9, x6, #8 // $X_9 = \&A + 8$, $X_9 = A[1]$ $\quad F = 0x100 + 0x100$

ADD X_{10}, X_6, X2R // $X_{10} = \&A + 0$, $X_{10} = A[0]$ $\quad F = 0x200$

STUR X_{10}, [x9, #0] // $\&A[1] + 0 = A[0]$, $A[1] = A[0]$

LDUR x9, [x9, #0] // $A[1] = A[0]$

ADD X_0, x9, X_{10} // $F = A[0] + A[0]$

- ORR (OPERATE OR)

REALIZA OPS. LÓGICAS BIT A BIT

ORR DESTINO, OPERANDO1, OPERANDO2

SE ALMACENARÁ EL RESULTADO ↓ REGISTROS Q CONTIENEN LOS VALORES

EL BIT RESULTADO SERÁ 1 SI ALGUNO ES 1

0	OR	0 = 0
0	OR	1 = 1
1	OR	0 = 1
1	OR	1 = 1

⑧ $x_9 = 0x55555555$ $\quad x_{10} = 0x12345678$

$x_9 = 0x0000000055555555$ $\quad x_{10} = 0x1234567812345678$

LSL x_{11}, x_9, #4 // $x_{11} = x_9 \cdot 2^4 = x_{11} = 16 * 0x0000000055555555 = 0x0000555555550000$

ORR x_{11}, x_{11}, x_{10} // $0x0000555555550000$ OR $0x1234567812345678$ $x_{11} = 0x1234FEFABABE5678$

```
0000 0000 0000 0000 1010 1010 1010 1010
0001 0010 0011 0100 0101 0110 0111 1000
0001 0010 0011 0100 1111 1110 1111 1010   ← OR
1010 1010 1010 1010 0000 0000 0000 0000
0001 0010 0011 0100 0101 0110 0111 1000
1011 1010 1011 1110 0101 0110 0111 1000   ← OR
```

EXCEPTION SYNDROME REGISTER (ESR)

SE UTILIZA PARA REGISTRAR INFORMACIÓN SOBRE LAS EXCEPCIONES (INTERRUPCIONES) QUE OCURREN DURANTE LA EJECUCIÓN

EXCEPTION CLASS	INSTRUCTION LENGTH (IL)	INSTRUCTION SPECIFIC SYNDROME FIELD (ISS)

31 26 25 24 0

- LSR (LÓGICA SHIF RIGHT)

DESPLAZAMIENTOS LÓGICOS A LA DERECHA EN EL CONTENIDO DE UN REGISTRO

LSR destino, origen, cant de bits (SE UTILIZA PARA DIVIDIR UN NUM POR UNA POTENCIA DE 2)

ej: LSR X_0, X_0, #2 $\quad x_0 / 2^2 = x_0 / 4$

- MOVZ (Move with Zero)

MOVZ destino, inmediato, desplazamiento

SE USA PARA MOVER UN VALOR INMEDIATO (CONSTANTE) A UN REGISTRO LLENANDO LOS BITS NO ESP. CON 0's

- MOVK (MOVE with KEEP)

MOVK destino, inmediato, desplazamiento

MODIFICAN SELECTIVAMENTE CIERTOS BITS EN UN REGISTRO MANTENIENDO LOS OTROS BITS SIN CAMBIO

EJEMPLO DE MOVZ Y MOVK DEL PRÁCTICO

(11) (10) $X_0 = 0x1234000000000000$

MOVZ X_0, #0x1234, LSL #48

$X_1 = 0xBBB0000000000AAA$

MOVZ X_1, #BBB0, LSL #48

MOVK X_1, #0AAA, LSL #0

$X_2 = 0xA0A0B1B10000C2C2$

MOVZ X_2, #A0A0, LSL #48

MOVK X2, #B1B1, LSL #32

MOVZ X2, #C2C2, LSL #0

$x_3 = 0x0123456789ABCDEF$

MOVZ X3, #0123, LSL #48
MOVK X3, #4567, LSL #32
MOVK X3, #89AB, LSL #16
MOVK X3, #CDEF, LSL #0

EXPLICACIÓN EN CRIOLLO DEL MOVZ Y DEL MOVK

MOVZ: X_1, #0x1234, LSL #16
EL NUM QUE LE DAS (0x1234) DESPLAZALO 16 BITS HACIA LA IZQUIERDA Y PONELO EN X_1
TODOS LOS DEMAS BITS DEL REGISTRO VALEN 0

TIENE UN CAMPO DE DESPLAZAMIENTO DE 0, 16, 32, 48 BITS, ESTOS SON MULTIPLOS DE 16 BITS YA QUE CADA DESPLAZAMIENTO DE 16 BITS CORRESPONDE A LA POSICIÓN DE UN DIGITO HEXADECIMAL DENTRO DE UN REG 64

EN RESUMEN PODEMOS MOVER HASTA 16 BITS A LA IZQUIERDA Y PONERLOS EN EL LUGAR QUE QUERAMOS LOS OTROS DATOS SON 0's

MOVK
MUY PARECIDO A MOVZ CON LA DIFERENCIA QUE TODO LO DEMÁS, EN LAS OTRAS POS. QUEDA COMO ESTABA, A EXCEPCIÓN DE DONDE MUVIMOS LOS NÚMEROS QUE SE REEMPLAZA POR LO QUE QUERÉS.

• AND
REALIZA OPERACIONES LÓGICAS BIT A BIT

AND destino, registro1, registro2

HACE LA OPERACIÓN ENTRE REG1 Y REG2 Y ALMACENA EL RESULTADO EN "destino"

0 AND 0 = 0
0 AND 1 = 0
1 AND 0 = 0
1 AND 1 = 1

SI AMBOS BIT SON 1's EL BIT CORRESPONDIENTE SERA 1, SINO SERÁ 0

• ANDI / ORI (I = INMEDIATO)

FUNCIONAN EXACTAMENTE IGUAL QUE EL AND Y EL OR CON LA EXCEPCIÓN DE QUE EN VEZ DE HACER LA OPERACIÓN CON UN REGISTRO LA HACES CON UN VALOR INMEDIATO.

ANDi, destino, registro, Valor_inmediato

de azúcar.