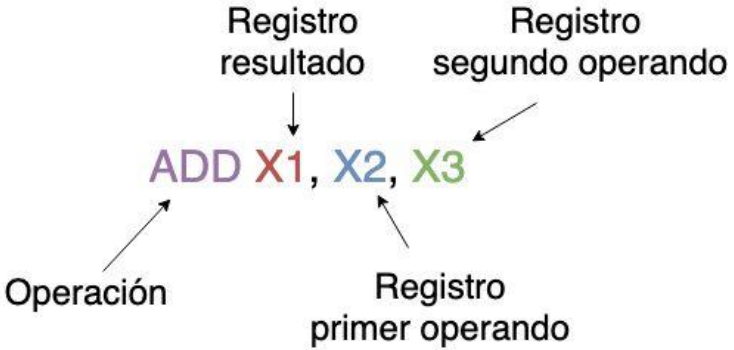
LEGv8 básico

OdC - 2021

Lenguaje ensamblador



Si se realiza la siguiente asignación de registros:

X1 → a
X2 → b
X3 → c

Esta instrución podria escribirse en lenguaje C como:

a = b + c

Conjunto de instrucciones - Aritmética

Category	InstructionExample		Meaning	Comments
Arithmetic	add	`ADD X1, X2, X3`	X1 = X2 + X3	Three register operands
	subtract	`SUB X1, X2, X3`	X1 = X2 − X3	Three register operands
	add immediate	`ADDI X1, X2, 20`	X1 = X2 + 20	Used to add constants
	subtract immediate	`SUBI X1, X2, 20`	X1 = X2 − 20	Used to subtract constants
	add and set flags	`ADDS X1, X2, X3`	X1 = X2 + X3	Add, set condition codes
	subtract and set flags	`SUBS X1, X2, X3`	X1 = X2 − X3	Subtract, set condition codes
	add immediate and set flags	`ADDIS X1, X2, 20`	X1 = X2 + 20	Add constant, set condition codes
	subtract immediate and set flags	`SUBIS X1, X2, 20`	X1 = X2 − 20	Subtract constant, set condition codes

Figura 2.1 - Computer Organization and design, Arm Edition - Patterson & Hennessy

Conjunto de instrucciones - Lógica

Category	Instruction	Example	Meaning	Comments
Logical	and	`AND X1, X2, X3`	`X1 = X2 & X3`	Three reg. operands; bit-by-bit AND
	inclusive or	`ORR X1, X2, X3`	`X1 = X2 | X3`	Three reg. operands; bit-by-bit OR
	exclusive or	`EOR X1, X2, X3`	`X1 = X2 ^ X3`	Three reg. operands; bit-by-bit XOR
	and immediate	`ANDI X1, X2, 20`	`X1 = X2 & 20`	Bit-by-bit AND reg. with constant
	inclusive or immediate	`ORRI X1, X2, 20`	`X1 = X2 | 20`	Bit-by-bit OR reg. with constant
	exclusive or immediate	`EORI X1, X2, 20`	`X1 = X2 ^ 20`	Bit-by-bit XOR reg. with constant
	logical shift left	`LSL X1, X2, 10`	`X1 = X2 << 10`	Shift left by constant
	logical shift right	`LSR X1, X2, 10`	`X1 = X2 >> 10`	Shift right by constant

Figura 2.1 - Computer Organization and design, Arm Edition - Patterson & Hennessy

Instalación QEMU

0-TENER ACTUALIZADOS LOS REPOSITORIOS

```
$ sudo apt update
```

1- SETTING UP AARCH64 TOOLCHAIN

```
$ sudo apt install gcc-aarch64-linux-gnu
```

2- SETTING UP QEMU ARM (incluye aarch64)

```
$ sudo apt install qemu-system-arm
```

3- FETCH AND BUILD AARCH64 GDB

```
$ sudo apt install gdb-multiarch
```

4- CONFIGURAR GDB PARA QUE HAGA LAS COSAS MÁS AMIGABLES

```
$ wget -P ~ git.io/.gdbinit
```

Ensamblado y simulación (1/4)

El código a simular se debe escribir en el archivo main.s

```
.data
b: .dword 15
c: .dword 2
```

Segmento de datos

```
.text
LDR X2, b
LDR X3, c
ADD X1, X2, X3
```

Segmento de código

Ensamblado y simulación (2/4)

Ensamblado

```
$ make
aarch64-linux-gnu-as -g --warn --fatal-warnings main.s -o main.o
aarch64-linux-gnu-ld main.o -T memmap -o main.elf -M > memory_map.txt
aarch64-linux-gnu-objdump -D main.elf > main.list
aarch64-linux-gnu-objcopy main.elf -O ihex main.hex
aarch64-linux-gnu-objcopy main.elf -O binary kernel.img
```

Inicio del simulador

```
$ qemu-system-aarch64 -s -S -machine virt -cpu cortex-a53 -machine type=virt
-nographic -smp 1 -m 64 -kernel kernel.img
```

Nota: Al ejecutar este comando, la terminal queda ejecutandolo. Para continuar con la ejecución del dashboard, se debe abrir una nueva terminal.

Ensamblado y simulación (3/4)

Inicio del debugger GDB

```
$ gdb-multiarch -ex "set architecture aarch64" -ex "target remote
localhost:1234"
```

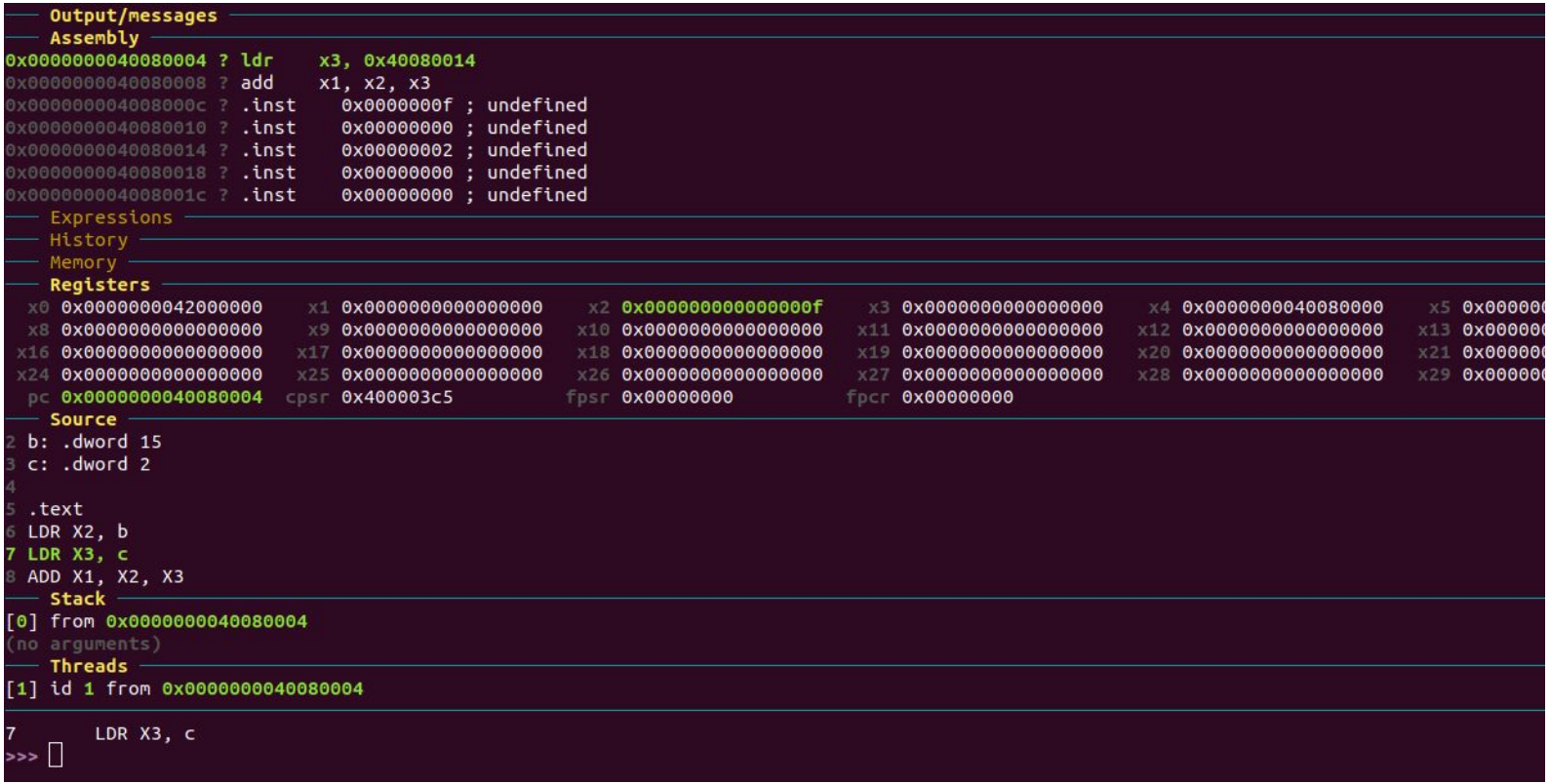
Configurar la arquitectura a utilizar

```
>>> set architecture aarch64
```

Importar al GDB los símbolos de debug en la dirección de memoria donde se encuentra el programa

```
>>> add-symbol-file main.o 0x0000000040080000
```

Ensamblado y simulación (4/4)



Ejercicio 1

Dadas las siguientes sentencias en “C”:

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

b) f = g + (h + 5);

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

1.1) Escribir la secuencia **mínima** de código assembler LEGv8 asumiendo que f, g, h, i y j se asignan en los registros X0, X1, X2, X3 y X4 respectivamente.

1.2) Dar el valor de cada variable en cada instrucción assembler si f, g, h, i y j se inicializan con valores de 1, 2, 3, 4, 5, en base 10, respectivamente.

Resolución ejercicio 1.1-a

Para poder resolver esta sentencia de C con instrucciones de LEGv8:

f = g + h + i + j;

Es necesario descomponerla en sentencias de dos operandos:

f = g + h;

f = f + i;

f = f + j;

Dado que f, g, h, i y j se asignan en los registros X0, X1, X2, X3 y X4 respectivamente, el código es:

```
ADD X0, X1, X2
ADD X0, X0, X3
ADD X0, X0, X4
```

Resolución ejercicio 1.2-a con QEMU (1/2)

Si f, g, h, i y j se inicializan con valores de 1, 2, 3, 4, 5

```
X0 ←f                    // f ←1
X1 ←g                    // g ←2
X2 ←h                    // h ←3
X3 ←i                    // i ←4
X4 ←j                    // j ←5

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

Resolución ejercicio 1.2-a con QEMU (1/2)

```
.data
f: .dword 1
g: .dword 2
h: .dword 3
i: .dword 4
j: .dword 5
```

```
↲
.text
LDR X0, f
LDR X1, g
LDR X2, h
LDR X3, i
LDR X4, j
ADD X0, X1, X2
ADD X0, X0, X3
ADD X0, X0, X4

infloop: B infloop
```

Resolución ejercicio 1.1-b

Para poder resolver esta sentencia de C con instrucciones de LEGv8:

f = g + (h + 5);

Es necesario descomponerla en sentencias de dos operandos:

f = h + 5;

f = f + g;

Dado que f, g, h, i y j se asignan en los registros X0, X1, X2, X3 y X4 respectivamente, el código es:

```
ADDI X0, X2, #5
ADD X0, X0, X1
```

Resolución ejercicio 1.2-b con QEMU

```
.data
f: .dword 1
g: .dword 2
h: .dword 3
i: .dword 4
j: .dword 5
```

```
↲
.text
LDR X0, f
LDR X1, g
LDR X2, h
LDR X3, i
LDR X4, j
ADDI X0, X2, #5
ADD X0, X0, X1

infloop: B infloop
```

Ejercicio 2

Luego, dadas las siguientes sentencias en assembler LEGv8:

a)
```
ADD  X0, X1, X2
```

b)
```
ADDI X0, X0, #1
   ADD  X0, X1, X2
```

2.1) Escribir la secuencia mínima de código “C” asumiendo que los registros X0, X1 y X2 contienen las variables f, g y h respectivamente.

2.2) Dar el valor de cada variable en cada instrucción assembler si f, g y h se inicializan con valores de 1, 2, 3, en base 10, respectivamente.

Ejercicio 2.1

```
a) ADD  X0, X1, X2                    // f = g + h
               f = g + h

b) ADDI X0, X0, #1                  // f = f + 1
  ADD  X0, X1, X2                   // f = g + h

               f = g + h
```

Ejercicio 3

Ejercicio 3:

Dadas las siguientes sentencias en “C”:

a) `f = -g - f;`

b) `f = g + (-f - 5);`

3.1) Escribir la secuencia mínima de código assembler LEGv8 asumiendo que f y g se asignan en los registros X0 y X1 respectivamente.

3.2) Dar el valor de cada variable en cada instrucción assembler si f y g se inicializan con valores de 4 y 5, en base 10, respectivamente.

Ejercicio 3.1

a)
```
f = -g - f;
f = - (g + f);

ADD X0, X1, X0
SUB X0, XZR, X0
```

b)
```
f = g + (-f - 5);
f = g - ( f + 5);

ADDI X0, X0, #5
SUB  X0, X1, X0
```