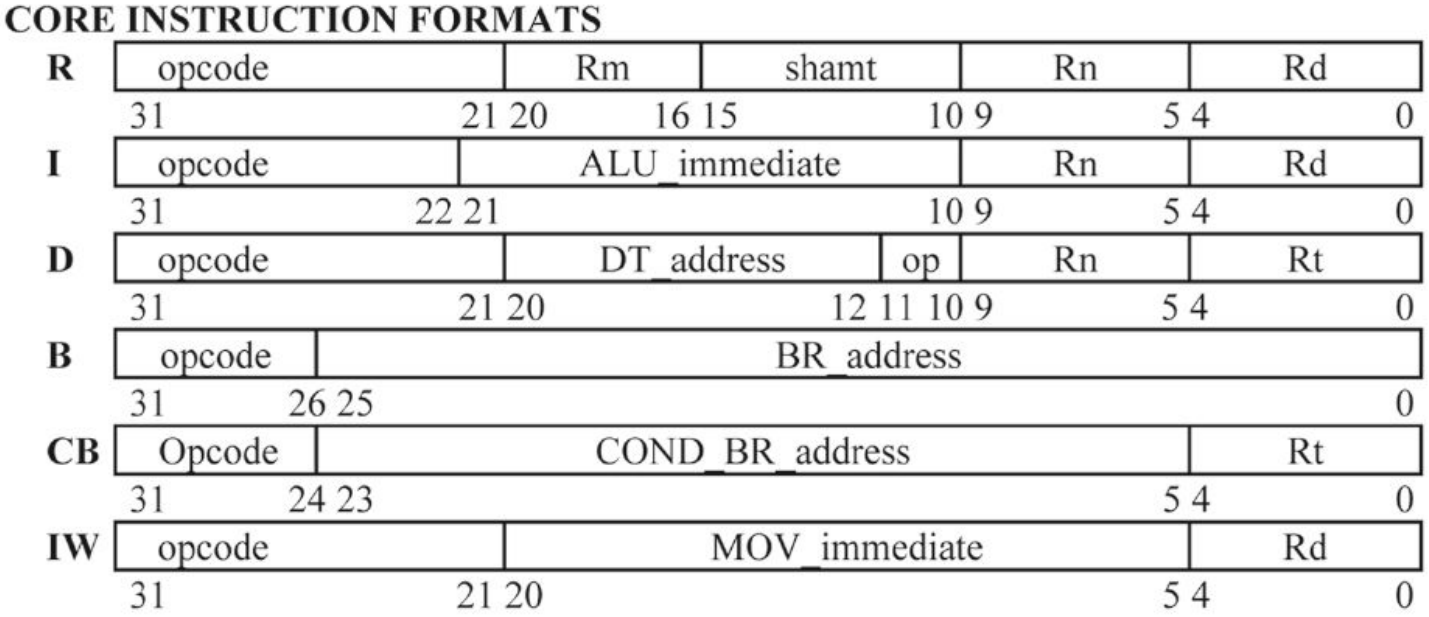
Ensamblado y desensamblado de LEGv8

Parte 1

OdC - 2021

Formatos de instrucción



- Todas las instrucciones son de 32 bits.
- Los 32 registros de LEGv8 se referencian por su número, de 0 a 31 (de 0b00000 a 0b11111).

Ensamblado de una instrucción

- Traducir una instrucción en assembler LEGv8 a una instrucción de máquina.
- Por ejemplo: La instrucción representada simbólicamente como

ADD X9, X20, X21

se ensambla en binario como:

10001011000	10101	000000	10100	01001
11 bits	5 bits	6 bits	5 bits	5 bits

- Cada segmento de la instrucción se llama campo.

Campos de LEGv8 - Instrucción tipo R

- opcode: Denotes the operation and format of an instruction.
- Rm: The second register source operand.
- shamt: Shift amount.
- Rn: The first register source operand.
- Rd: The register destination operand. It gets the result of the operation.

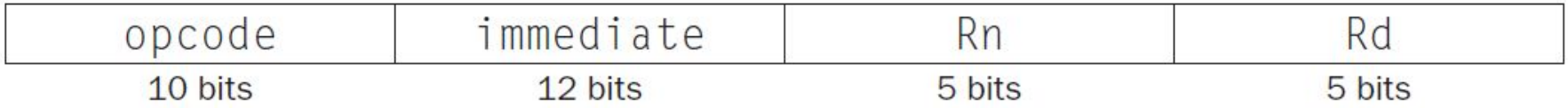
Campos de LEGv8 - Instrucción tipo D

opcode	address	op2	Rn	Rt
11 bits	9 bits	2 bits	5 bits	5 bits

- Data transfer instructions (loads and stores).
- The 9-bit address means a load register instruction can load any doubleword within a region of $\pm 2^8$ of the address in the base register Rn.
- The last field of D-type is called Rt instead of Rd because for store instructions, the field indicates a data source and not a data destination.
- op2: expands opcode field (will be 0 in LEGv8).

$$(5) \quad \text{DTAddr} = \{ 55\{\text{DT\_address}[8]\}, \text{DT\_address} \}$$

Campos de LEGv8 - Instrucción tipo I

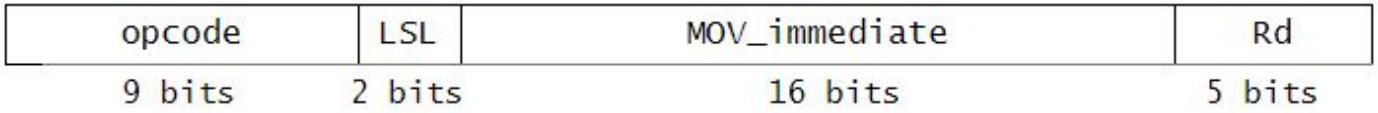


- The ARMv8 architects decided it would be useful to have a larger immediate field for these instructions, even shaving a bit from the opcode field to make a 12-bit immediate.
- The LEGv8 immediate field in I-format is zero extended.

(2) ALUImm = { 52'b0, ALU_immediate }

Nota para QEMU: The immediate fields for ANDI, ORRI, and EORI of the full ARMv8 instruction set are not simple 12-bit immediates. It has the unusual feature of using a complex algorithm for encoding immediate values. This means that some small constants are valid, while others are not.

Campos de LEGv8 - Instrucción tipo IM



bit 22	bit 21	LSL
0	0	0
0	1	16
1	0	32
1	1	48

The machine language version of `MOVZ X9, 255, LSL 16`:

110100101	01	0000 0000 1111 1111	01001

Contents of register `X9` after executing `MOVZ X9, 255, LSL 16`:

0000 0000 0000 0000	0000 0000 0000 0000	0000 0000 1111 1111	0000 0000 0000 0000

The machine language version of `MOVK X9, 255, LSL 0`:

111100101	00	0000 0000 1111 1111	01001

Given value of `X9` above, new contents of `X9` after executing `MOVK X9, 255, LSL 0`:

0000 0000 0000 0000	0000 0000 0000 0000	0000 0000 1111 1111	0000 0000 1111 1111

(6) MOVImm = { 48'b0, MOV_immediate }

Ejercicio 1

Extender los siguientes números de 26 bits en complemento a dos a 64 bits. Si el número es negativo verificar que la extensión a 64 bits codifica el mismo número original de 26 bits.

1.1) 00 0000 0000 0000 0000 0000 0001
1.2) 10 0000 0000 0000 0000 0000 0000

Rta:

1.1) 000000… 00 0000 0000 0000 0000 0000 0001 (64 bits)

1.2) 111111… 10 0000 0000 0000 0000 0000 0000 (64 bits)

Ejercicio 2

Tenemos las siguientes instrucciones en assembler LEGv8:

```
ADDI X9, X9, #0

STUR X10, [X11,#32]
```

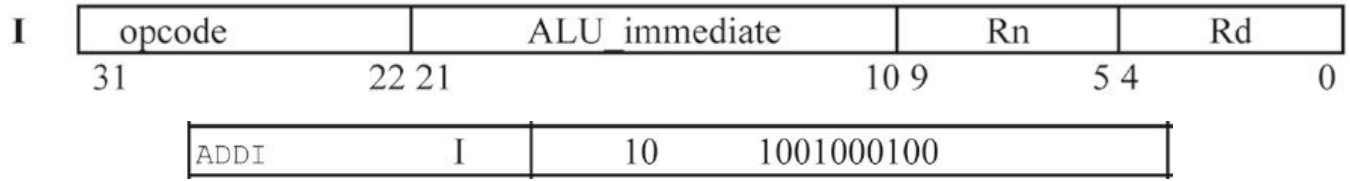
2.1) ¿Qué formato (R, I, D, B, CB, IM) de instrucciones son?

2.2) Ensamblar a código de máquina LEGv8, mostrando sus representaciones en binario y luego en hexadecimal.

Ejercicio 2 - ADDI X9, X9, #0

NAME, MNEMONIC		FOR-MAT	OPCODE (9) (Hex)	OPERATION (in Verilog)
ADD	ADD	R	458	R[Rd] = R[Rn] + R[Rm]
ADD Immediate	ADDI	I	488-489	R[Rd] = R[Rn] + ALUImm

2.1) Formato de instrucción: tipo I



2.2)
- opcode (10 bits): 1001000100
- ALU_immediate (12 bits): 000000000000
- Rn (5 bits) = X9: 01001
- Rd (5 bits) = X9: 01001

Ejercicio 2 - ADDI X9, X9, #0

2.2)
- opcode (10 bits): 1001000100
- ALU_immediate (12 bits): 000000000000
- Rn (5 bits) = X9: 01001
- Rd (5 bits) = X9: 01001

Instrucción ensamblada en binario:

0b **1001 0001 0000 0000 0000 0001 0010 1001**

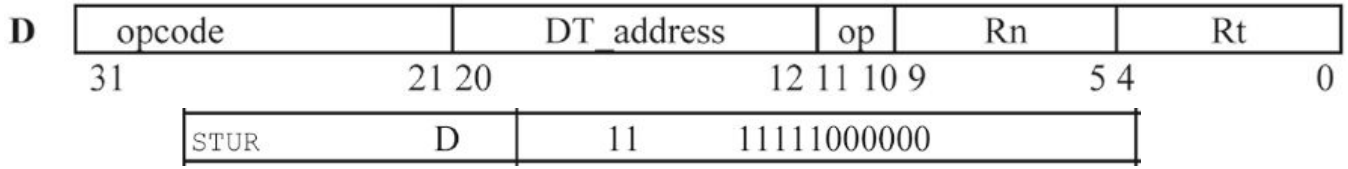
Instrucción ensamblada en hexadecimal:

0x **91000129**

Ejercicio 2 - STUR X10, [X11,#32]

STore Register Unscaled offset	STUR	D	7C0	M[R[Rn] + DTAddr] = R[Rt]

2.1) Formato de instrucción: tipo D



2.2)
- opcode (11 bits): 11111000000
- DT_address (9 bits): 000100000
- op (2 bits): 00
- Rn (5 bits) = X11: 01011
- Rt (5 bits) = X10: 01010

Ejercicio 2 - STUR X10, [X11,#32]

2.2)
- opcode (11 bits): 11111000000
- DT_address (9 bits): 000100000
- op (2 bits): 00
- Rn (5 bits) = X11: 01011
- Rt (5 bits) = X10: 01010

Instrucción ensamblada en binario:

0b **1111 1000 0000 0010 0000 0001 0110 1010**

Instrucción ensamblada en hexadecimal:

0x **F802016A**

Ejercicio 3.1

Dar el tipo de instrucción, la instrucción en assembler y la representación binaria de los siguientes campos de LEGv8:

op=0x658, Rm=13, Rn=15, Rd=17, shamt=0

SUB	R	11	11001011000	658

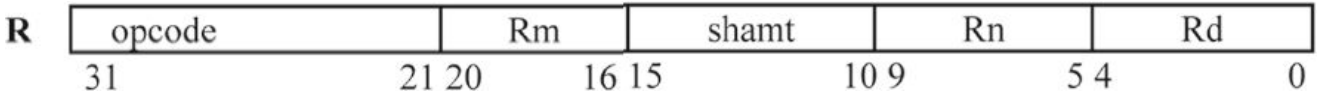
1) Con el opcode identificamos la instrucción: SUB y el **tipo: R**.

SUBtract	SUB	R	658	R[Rd] = R[Rn] – R[Rm]

2) Instrucción en assembler: **SUB X17, X15, X13**

Ejercicio 3.1

3) Sabiendo el tipo de instrucción identificamos el orden de los campos.



4)
- opcode (11 bits) = 0x658 = 0b11001011000
- Rm (5 bits) = 13 = 0b01101
- shamt (6 bits) = 0 = 0b000000
- Rn (5 bits) = 15 = 0b01111
- Rd (5 bits) = 17 = 0b10001

5) Representación binaria de la instrucción:

0b **1100 1011 0000 1101 0000 0001 1111 0001**

Desensamblado de una instrucción

Decoding Machine Code:

- To determine the opcode, you just take the first 11 bits of the instruction, convert it to hexadecimal representation, and then look up the table to find the instruction and its format.
- We see that opcode corresponds to the “XX” instruction, which uses the “X”-format. Thus, we can parse the binary format into fields.
- We decode the rest of the instruction by looking at the field values.

Ejercicio 4.2

Dado el número en binario: 1101 0010 1011 1111 1111 1111 1110 0010

a) Transformar de binario a hexadecimal.

b) ¿Qué instrucciones LEGv8 representan en memoria?

a) 0x **D2BFFFE2**

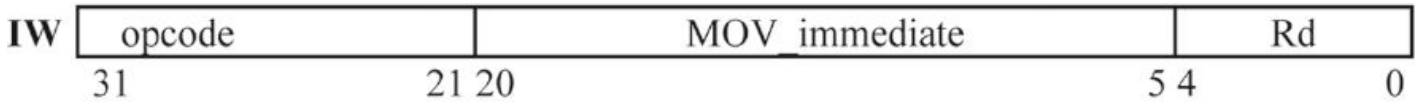
b) 1101 0010 1011 1111 1111 1111 1110 0010

- Campo de opcode de 11 bits = 11010010101 = 0x695

MOVZ	IM	9	110100101	694	697

- Opcode de la instrucción MOVZ = 110100101

Ejercicio 4.2



1101 0010 1011 1111 1111 1111 1110 0010

- opcode: 110100101 -> MOVZ
- LSL: 01 -> LSL 16
- MOV_immediate: 1111111111111111 -> 0xFFFF
- Rd: 00010 -> X2

bit 22	bit 21	LSL
0	0	0
0	1	16
1	0	32
1	1	48

Instrucción en assembler: **MOVZ X2, #0xFFFF, LSL 16**

Ejercicio 5

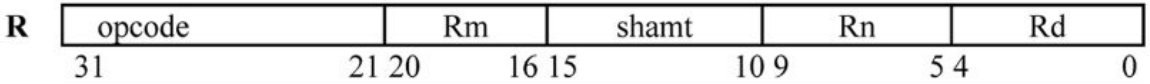
Ejecutar el siguiente código assembler que está en memoria para dar el valor final del registro X1. El contenido de la memoria se dá como una lista de pares, dirección de memoria: contenido, suponiendo alineamiento de memoria del tipo big endian. Describa sintéticamente que hace el programa.

0x10010000: 0x8B010029

0x10010004: 0x8B010121

```
0x10010000: 0x8B
0x10010001: 0x01
0x10010002: 0x00
0x10010003: 0x29

0x10010004: 0x8B
0x10010005: 0x01
0x10010006: 0x01
0x10010007: 0x21
```



Ejercicio 5

0x10010000: 0x8B010029 -> **10001011000** 00001 000000 00001 01001
op: **10001011000** -> 0x458 -> ADD
Rm: 00 001 -> X1
sh: 000000 -> 0
Rn: 00 001 -> X1
Rd: 01 001 -> X9
ADD X9, X1, X1

0x10010004: 0x8B010121 -> **10001011000** 00001 000000 01001 00001
op: **10001011000** -> 0x458 -> ADD
Rm: 00 001 -> X1
sh: 000000 -> 0
Rn: 01 001 -> X9
Rd: 00 001 -> X1
ADD X1, X9, X1

Resultado final: X1= 3*X1

Ejercicio 6

Decidir cuáles de las siguientes instrucciones en assembler se pueden codificar en código de máquina LEGv8. Explique qué falla en las que no puedan ser ensambladas.

1. LSL XZR, XZR, #0 (si)
2. ADDI X1, X2, #-1 (no signados)
3. ADDI X1, X2, #4096 (no, hasta 4095)
4. EOR X32, X31, X30 (no hay X32)
5. ORRI X24, X24, #0x1FFF (no)
6. STUR X9, [XZR,#-129] (si)
7. LDURB XZR, [XZR,#-1] (si)
8. LSR X16, X17, #68 (no, shamt 6 bits)
9. MOVZ X0, 0x1010, LSL #12 (no LSL 12)
10. MOVZ XZR, 0xFFFF, LSL #48 (si)

Introducción a la Experiencia CARDIAC

https://cs.famaf.unc.edu.ar/~nicolasw/cardiac_2.mp4

Bibliografía

Patterson and Hennessy, “Computer Organization and Design: The Hardware/Software Interface ARM Edition”, Morgan kaufmann, 2016.