Review on Numbering System

- The binary number system has base 2.
- The value of digit is determined by its position in the number.
- The two binary digits are: 1 and 0.

Full — binary Number : 101

Fractional- binary — Number: 101.11

$2^3\ 2^2\ 2^1\ 2^0 \cdot 2^{-1}\ 2^{-2}\ 2^{-3} \ldots 2^{-n}$ Binary - point

TABLE 2-2

Binary weights.

Positive Powers of Two (Whole Numbers)									Negative Powers of Two (Fractional Number)					
2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}
256	128	64	32	16	8	4	2	1	1/2 0.5	1/4 0.25	1/8 0.125	1/16 0.625	1/32 0.03125	1/64 0.015625

Example 2: Convert 10.111 to decimal number.

Weight : $2^1\ 2^0 \cdot 2^{-1}\ 2^{-2}\ 2^{-3}$

Binary-bit : 1 0 1 1 1

Sum-of-weights = $2^1 + 2^{-1} + 2^{-2} + 2^{-3}$

$= 2 + 0.5 + 0.25 + 0.125$

$= 2.875$

Converting whole decimal numbers to binary (Sum-of-weights Method)

- Determine the set of binary weights whose sum is equal to the decimal number.
- Place 1's and 0's on the appropriate weight positions determines the binary number for that decimal number.

16 + 8 + 1 = 25.

Example 1: Convert decimal number 25 to binary using Sum-of-Weights Method

25 =

weights	16	8	4	2	1
	1	1	0	0	1

$25_{10} = 11001_2$

32 + 16 + 8 + 2 = 58

Example 2: Convert decimal number 58 to binary using Sum-of-Weights Method.

weights	64	32	16	8	4	2	1
	0	1	1	1	0	1	0

$58_{10} = 111010_2$

Hexadecimal Numbers

Base - 16

- The hexadecimal number system has sixteen characters; it is used primarily as a compact way of displaying or writing binary numbers because it is very easy to convert between binary and hexadecimal.

TABLE 2-3

Decimal	Binary	Hexadecimal
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

Convert the following binary numbers to hexadecimal: (a) 1100101001010111

C A 5 7

$(CA57)_{16}$

Determine the binary numbers for the following hexadecimal numbers: (a) $10A4_{16}$

1 0 A 4

0001 0000 1010 0100 ₂

Chapter 2 (Instruction Set)

X1 → C
X2 → a
X3 → b

Operands

Low level language (Assembly Language):

- An operand is a value (an argument) on which the instruction operates. The operand may be a processor register, a memory address, or a literal constant.

operands

```
ADD X1, X2, X3        // X1 = X2 + X3
                      c = a + b
```

operator.

High level language (C/C++/Java):

operands, operator

- Operands are the constants or variables which the operators operate upon.

```
c = a + b ;
```

operands

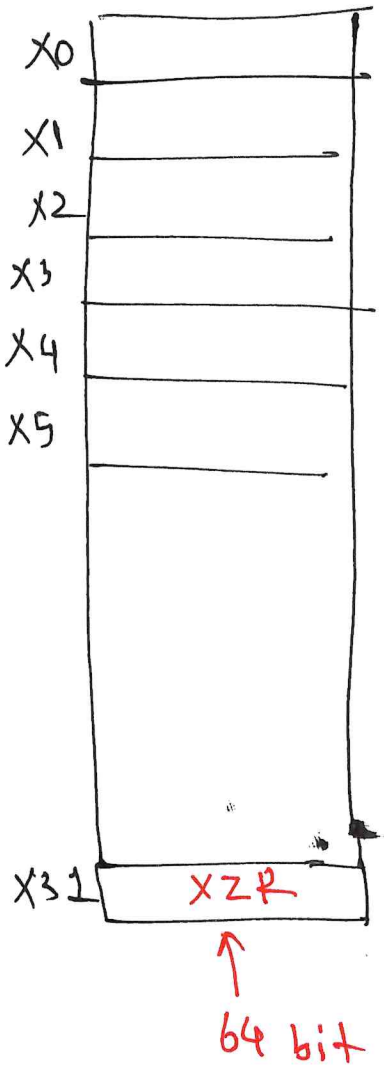
operator.

ARMv8

LEGv8 Registers

Name	Register number	Usage
X0-X7	0–7	Arguments/Results
X8	8	Indirect result location register
X9-X15	9–15	Temporaries
X16 (IP0)	16	May be used by linker as a scratch register; other times used as temporary register
X17 (IP1)	17	May be used by linker as a scratch register; other times used as temporary register
X18	18	Platform register for platform independent code; otherwise a temporary register
X19-X27	19–27	Saved
X28 (SP)	28	Stack Pointer
X29 (FP)	29	Frame Pointer
X30 (LR)	30	Link Register (return address)
XZR	31	The constant value 0

Register .



Arithmetic Instructions

ADD-

- The ADD instruction adds two operands and place the result on destination register. Both operands are register.

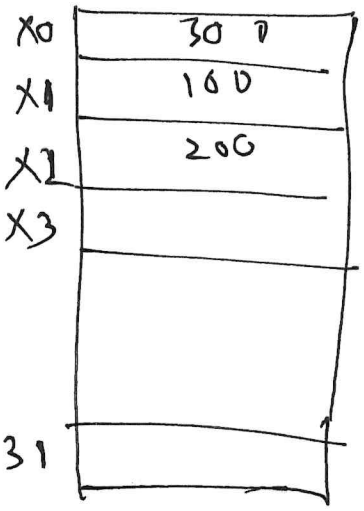
operand 2

```
ADD  X0, X1, X2
```

X0 = X1 + X2

X1 = 100, X2 = 200

Destination Operand 1 Register




```
ADD  X0, X0, X1.
```

300 + 100 = 400

Immediate.

ADDI –

- The ADDI instruction adds two operands and place the result on destination register. Operand 1 is a register, operand 2 is an immediate value.

operand1

suppose, X1 = 100;

```
ADDI  X0, X1, #4
```

X0 = X1 + 4.

X0 = 100 + 4 = 104

Destination Register

operand 2

SUB-

- The SUB instruction subtracts operand 2 from operand 1 and place the result on destination register. Both operands are register.

operand 2

```
SUB  X0, X1, X2        //   X0 = X1 - X2
```

operand 1

suppose, X1 = 500 X2 = 200

SUBI-

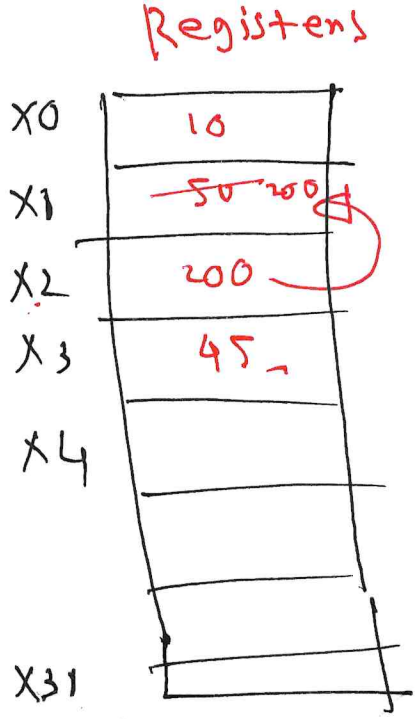
- The SUBI instruction subtracts operand 2 from operand 1 and place the result on destination register. Operand 1 is a register, operand 2 is an immediate value.

```
SUBI  X0, X1, #4       //   X0 = X1 - 4.
```

MOV-

- This instruction loads a 64-bit value into the destination register from another register.

```
MOV X1, X2      // X1 = X2
```



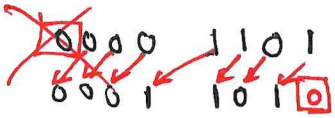
MOVI

- This instruction loads a 64-bit value into the destination register from an immediate value.

```
MOVI X1, #8     // X1 = 8
```

Destination Register

Immediate value.



LSL – Logical shift left

- LSL instruction effectively multiply the contents of a register by 2^i.
- Each bit of the register is shifted left, the MSB is removed and empty bits are filled with zeros.

X2 = 8

operand 1

```
LSL   X0, X2, #2        // X0 = X2 * 2^2
```

if X2 = 8

X0 = 8 * 4 = 32

```
X6 = 2
LSL   X5, X6, #4
```

X5 = X6 * 2^4

= 2 * 16 = 32

Hex MSB LSD

Problem: Assume that X2 = 2 (0X0000000000000002 in hexadecimal). What will be the value of X0 after running the following instruction: LSL X0, X2, #2

```
LSL   X0, X2, #2        // X0 = X2 * 2^2 = 2*4 = 8
```

0X0000 0000 0000 0002

0000 0010

2 =

1 shift ⟶ 2 * 2^1 = 4

2 shift ⟶ 2 * 2^2 = 8