# RISC V Architecture

**Mahesh Awati**
Department of Electronics and Communication Engg.

# RISC V ARCHITECTURE

## UNIT 2 – Instructions: The Language of Computer

**Mahesh Awati**
Department of Electronics and Communication Engineering

# Instructions – Language of Computer

## Signed and Unsigned Numbers

Let us consider a 32 bit (word) number representation

MSB LSB

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 |

(32 bits wide)

**Why didn't computers use decimal?**

How many bit pattern it can detect ?

What number they represent in decimal ?

$00000000\ 00000000\ 00000000\ 00000000_{two} = 0_{ten}$
$00000000\ 00000000\ 00000000\ 00000001_{two} = 1_{ten}$
$00000000\ 00000000\ 00000000\ 00000010_{two} = 2_{ten}$
... ...
$11111111\ 11111111\ 11111111\ 11111101_{two} = 4{,}294{,}967{,}293_{ten}$
$11111111\ 11111111\ 11111111\ 11111110_{two} = 4{,}294{,}967{,}294_{ten}$
$11111111\ 11111111\ 11111111\ 11111111_{two} = 4{,}294{,}967{,}295_{ten}$

## Signed and Unsigned Numbers

Let us understand Unsigned Number Reprentation using 32 bit binary number

| Binary Number (31 … 0) | Hexa-Decimal | |
|---|---|---|
| 0000 0000 0000 0000 0000 0000 0000 0000 | (0x 0000 0000) | $0_{ten}$ |
| 0000 0000 0000 0000 0000 0000 0000 0001 | (0x 0000 0001) | $1_{ten}$ |
| 0000 0000 0000 0000 0000 0000 0000 0010 | (0x 0000 0002) | $2_{ten}$ |
| … | | |
| … | | |
| 0111 1111 1111 1111 1111 1111 1111 1111 | (0x 7FFF FFFF ) | |
| 1000 0000 0000 0000 0000 0000 0000 0010 | (0x 8000 0000) | |
| 1000 0000 0000 0000 0000 0000 0000 0001 | (0x 8000 0001) | |
| … | | |
| …. | | |
| 1111 1111 1111 1111 1111 1111 1111 1110 | (0x FFFF FFFE ) | $4{,}294{,}967{,}293_{ten}$ |
| 1111 1111 1111 1111 1111 1111 1111 1111 | (0x FFFF FFFF ) | $4{,}294{,}967{,}294_{ten}$ |
| | | $4{,}294{,}967{,}295_{ten}$ |

## Signed and Unsigned Numbers

How Signed Number Reprentation is different from Unsigned Number Representation?

| 31 | 30 ... 0 |
|---|---|
| Sign | Magnitude |

2's complement representation indicates negative numbers have a 1 in the MSB. Thus, hardware needs to test only MSB to decide whether the Number is positive or negative.

| 31 | 30 | | | | | | | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | (0x 0000 0000) 0 |
| 0 | 000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0001 | (0x 0000 0001) 1 |
| 0 | 000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0010 | (0x 0000 0002) 2 |
| … | | | | | | | | | |
| … | | | | | | | | | |
| 0 | 111 | 1111 | 1111 | 1111 | 1111 | 1111 | 1111 | 1111 | (0x 7FFF FFFF ) .. |
| 1 | 000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0010 | (0x 8000 0000) .. |
| 1 | 000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0001 | (0x 8000 0001) |
| … | | | | | | | | | |
| …. | | | | | | | | | |
| 1 | 111 | 1111 | 1111 | 1111 | 1111 | 1111 | 1111 | 1110 | (0x FFFF FFFE )=-2 |
| 1 | 111 | 1111 | 1111 | 1111 | 1111 | 1111 | 1111 | 1111 | (0x FFFF FFFF ) =-1 |

**Positive Number Range**

**Negative Number Range & They are in 2's complement form**

## Signed and Unsigned Numbers

How Signed Number Reprentation is different from Unsigned Number Representation?

| 31 | 30 | 0 |
|---|---|---|
| Sign | Magnitude | |

2's complement representation indicates negative numbers have a 1 in the MSB. Thus, hardware needs to test only MSB to decide whether the Number is positive or negative.

2s-Complement Signed Integers

Given an n-bit number

$$x = -x_{n-1}2^{n-1} + x_{n-2}2^{n-2} + \cdots + x_1 2^1 + x_0 2^0$$

Range: $-2^{n-1}$ to $+2^{n-1} - 1$

Example

$1111\ 1111 \ldots 1111\ 1100_2$

$= -1\times2^{31} + 1\times2^{30} + \ldots + 1\times2^2 + 0\times2^1 + 0\times2^0$

$= -2{,}147{,}483{,}648 + 2{,}147{,}483{,}644 = -4_{10}$

Using 64 bits: −9,223,372,036,854,775,808 to 9,223,372,036,854,775,807

## Signed and Unsigned Numbers

How Signed Number Reprentation is different from Unsigned Number Representation?

| | |
|---|---|
| 31 | 30 0 |
| Sign | Magnitude |

2s-Complement Signed Integers

Some specific numbers

0: 0000 0000 … 0000

–1: 1111 1111 … 1111

Most-negative: 1000 0000 … 0000

Most-positive: 0111 1111 … 1111

## Signed and Unsigned Numbers

What is the decimal value of this 64-bit two's complement number?

$11111111\ 11111111\ 11111111\ 11111111\ 11111111\ 11111111\ 11111111\ 11111000_{two}$

1) $-4_{ten}$
2) $-8_{ten}$
3) $-16_{ten}$
4) $18{,}446{,}744{,}073{,}709{,}551{,}608_{ten}$

What is the decimal value if it is instead a 64-bit unsigned number?

# Instructions – Language of Computer

## Load and Store Instructions

| Category | Instruction | Example | Meaning | Comments |
| --- | --- | --- | --- | --- |
| Arithmetic | Add | add x5, x6, x7 | x5 = x6 + x7 | Three register operands; add |
| | Subtract | sub x5, x6, x7 | x5 = x6 - x7 | Three register operands; subtract |
| | Add immediate | addi x5, x6, 20 | x5 = x6 + 20 | Used to add constants |
| Data transfer | Load word | lw x5, 40(x6) | x5 = Memory[x6 + 40] | Word from memory to register |
| | Load word, unsigned | lwu x5, 40(x6) | x5 = Memory[x6 + 40] | Unsigned word from memory to register |
| | Store word | sw x5, 40(x6) | Memory[x6 + 40] = x5 | Word from register to memory |
| | Load halfword | lh x5, 40(x6) | x5 = Memory[x6 + 40] | Halfword from memory to register |
| | Load halfword, unsigned | lhu x5, 40(x6) | x5 = Memory[x6 + 40] | Unsigned halfword from memory to register |
| | Store halfword | sh x5, 40(x6) | Memory[x6 + 40] = x5 | Halfword from register to memory |
| | Load byte | lb x5, 40(x6) | x5 = Memory[x6 + 40] | Byte from memory to register |
| | Load byte, unsigned | lbu x5, 40(x6) | x5 = Memory[x6 + 40] | Byte unsigned from memory to register |
| | Store byte | sb x5, 40(x6) | Memory[x6 + 40] = x5 | Byte from register to memory |

# Instructions – Language of Computer

## Signed and Unsigned Numbers

### Sign extension

- **Representing a number using more bits**
  Ex: Reading a variable of 16 bit (half word) size and loading into a 32 bit register
- The variable may be a Unsigned / signed number

✓ **Where is the 16 bit data from memory is loaded in destination register ???**

✓ **What will happen to remaining 16 bits of the destination register i.e. reg[31:16]???**

✓ **reg[31:16] will change the content and It differs based on whether variable is Signed and Unsigned ?**

**Memory**

| 15 12 | 11 8 | 7 4 | 3 0 |
|---|---|---|---|
| 0111 | 0101 | 1011 | 0001 |
| | | | |
| | | | |
| | | | |

Loading 16 bit data from memory into a 32 bit register i.e., **Representing a number using more bits**

**Register**

| 31 28 | | | | 15 12 | 11 8 | 7 4 | 3 0 |
|---|---|---|---|---|---|---|---|
| 1010 | 1111 | 1011 | 1000 | 0111 | 0101 | 1011 | 0001 |

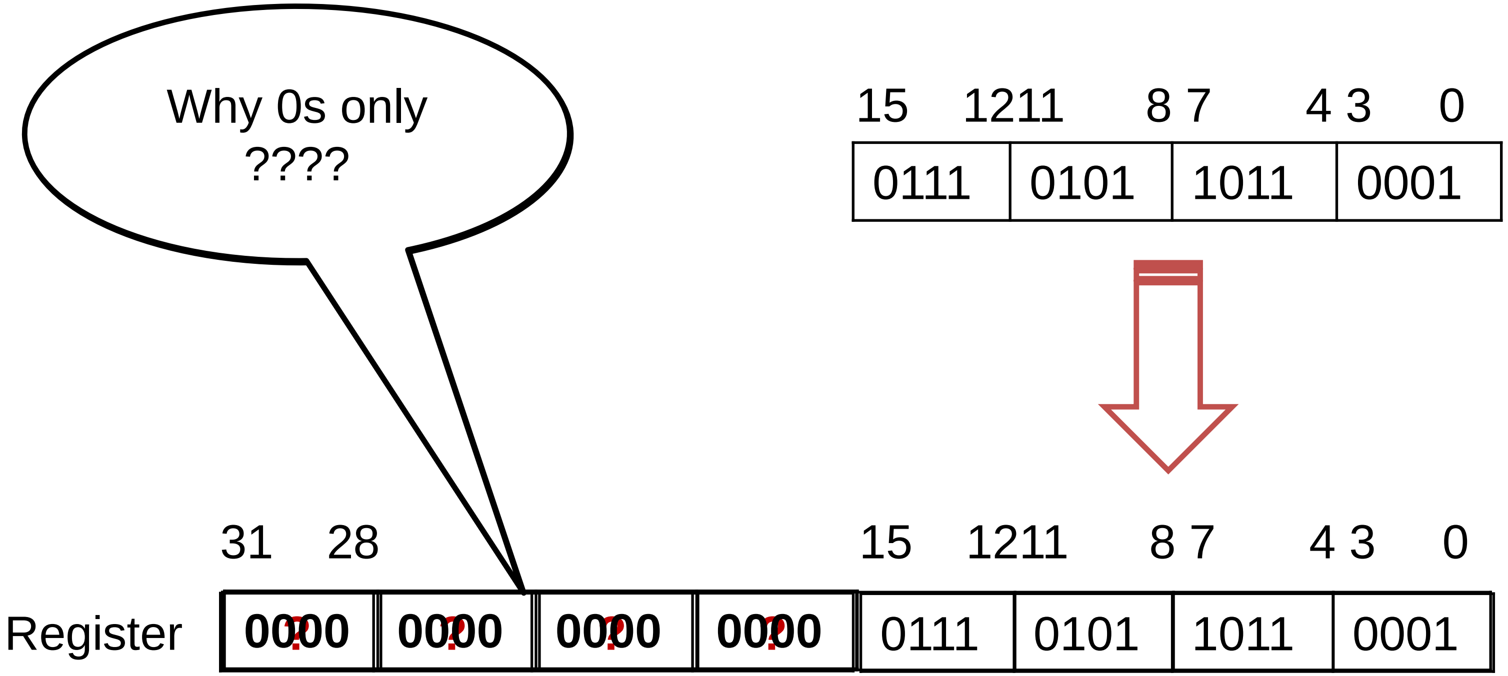
**Data in Register in Hexa**

**0xAFB875B1**

## Sign extension

How a Signed and Unsigned Load into a Register differs ?

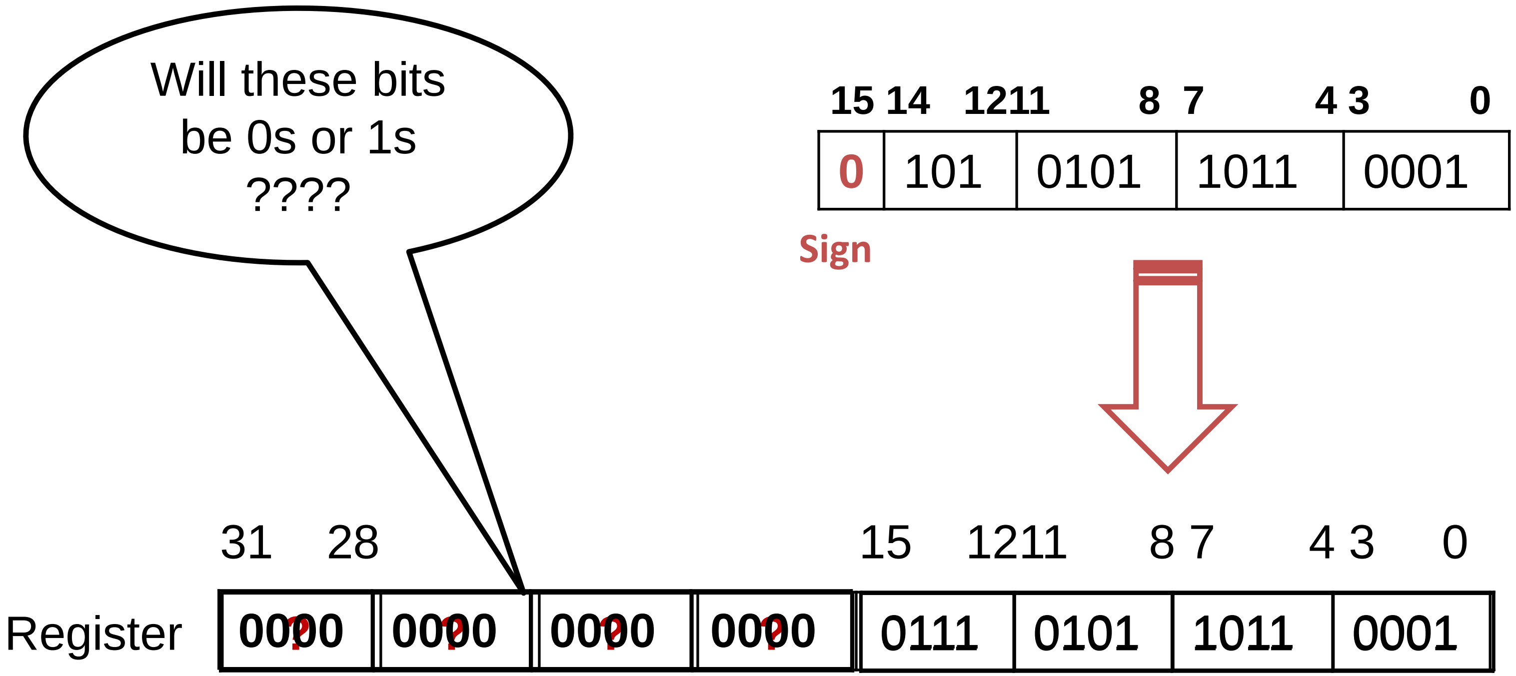
Let us consider loading 32 bit register with a **16 bit unsigned value** ?



### Sign extension

How a Signed and Unsigned Load into a Register differs ?

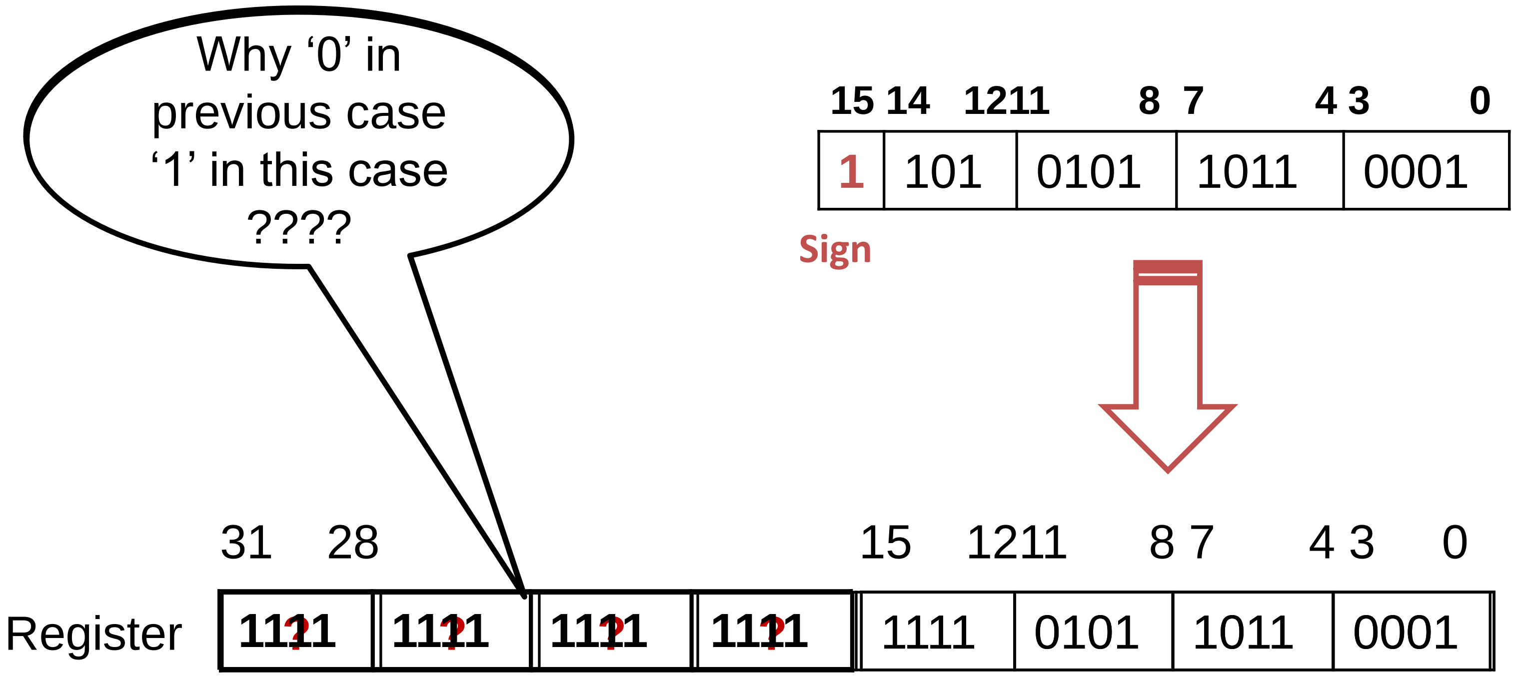
Let us consider loading 32 bit register with a **16 bit Signed value** ?



### Sign extension

How a Signed and Unsigned Load into a Register differs ?

Let us consider loading 32 bit register with a **16 bit Signed value** ?




- When loading a 32-bit word into a 32-bit register, the point is moot ( Matter of No importance); signed and unsigned loads are identical.
- When loading 16 bit half word into a 32 bit register, the signed and unsigned loads are not identical.

## Sign extension

- RISC-V does offer two flavors of byte loads:
  1. ***load byte unsigned*** **(lbu)** treats the byte as an unsigned number and thus zero-extends to fill the leftmost bits of the register,
  2. ***load byte*** **(lb)** works with signed integers.

  Since C programs almost always use bytes to represent characters rather than consider bytes as very short signed integers, lbu is used practically exclusively for byte loads.
- The binary representation can be used as data and Address

  **Does Signed Number Representation makes any sense when it is used to address and data ????**

### Working with two's complement numbers.

Negate a two's complement binary number ?

**Ex: Negation of (-1) is (+1)**

X= (-1) = 1111 1111 1111 1111 1111 1111 1111 1111 (0xFFFF FFFF)

(+1)= 0000 0000 0000 0000 0000 0000 0000 0001 (0x0000 0001)

How can you find it quickly ????

**Solution :**

a) X' + 1 = -X
b) This shortcut is based on the observation that the **sum of a number and its inverted representation must be $(111 \ldots 111)_2$**, which represents −1. [ X+X' =-1]

X= (-1) = 1111 1111 1111 1111 1111 1111 1111 1111 (0xFFFF FFFF)

X' = 0000 0000 0000 0000 0000 0000 0000 0000 (0x0000 0000)

-------------------------------------------------------------------------------------

(-1) = 1111 1111 1111 1111 1111 1111 1111 1111

Since $x + x' = -1$ , therefore $x + x' + 1 = 0$ or $x' + 1 = -x$ .

# THANK YOU

**Mahesh Awati**
Department of Electronics and Communication

**mahesha@pes.edu**

+91 9741172822