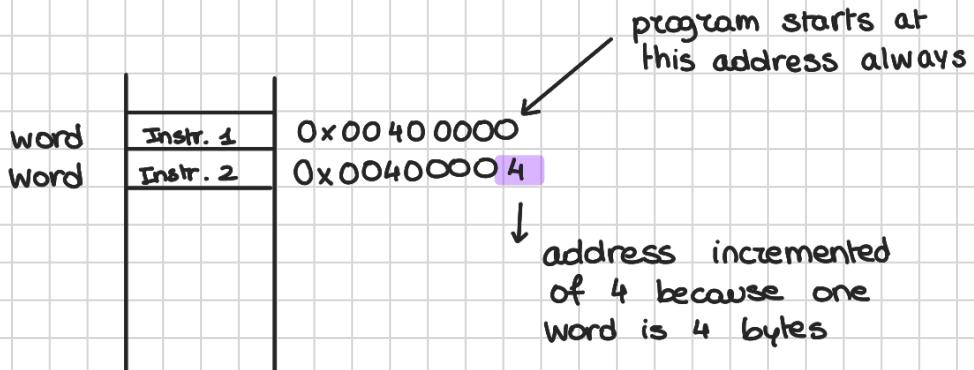


# Computer Architecture ( Unit 2 )

DATA : 25/02/2002



## ASSEMBLY INSTRUCTION

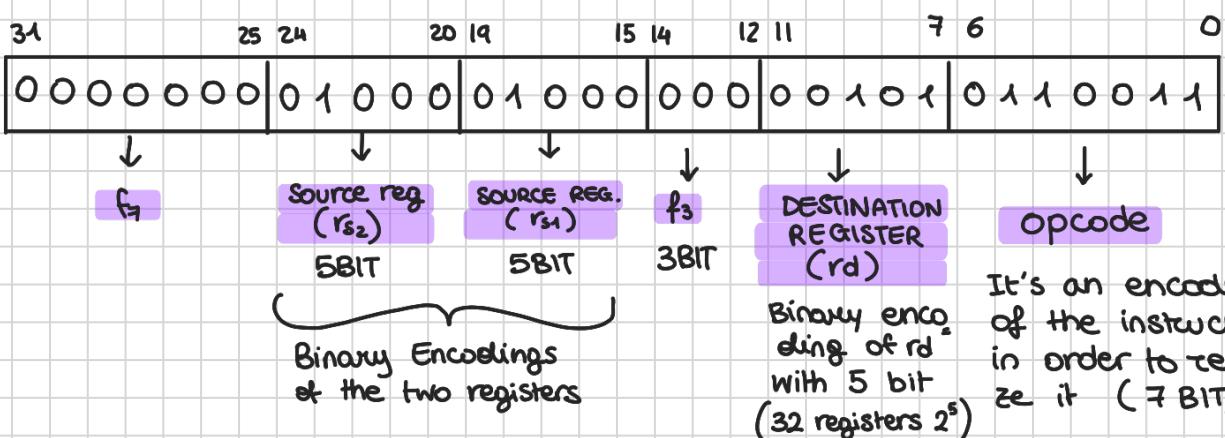
= instructions

### ADD

- Adds the content of two registers and stores into a destination register.

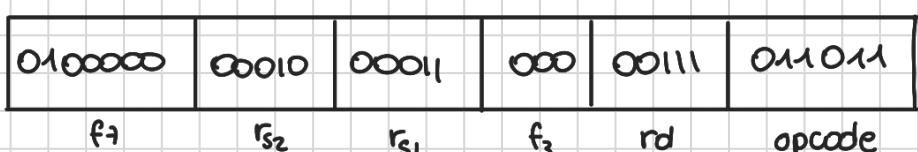
EXAMPLE:  $R[5] \leftarrow R[8] + R[8]$

destination                          source register 1                          source register 2



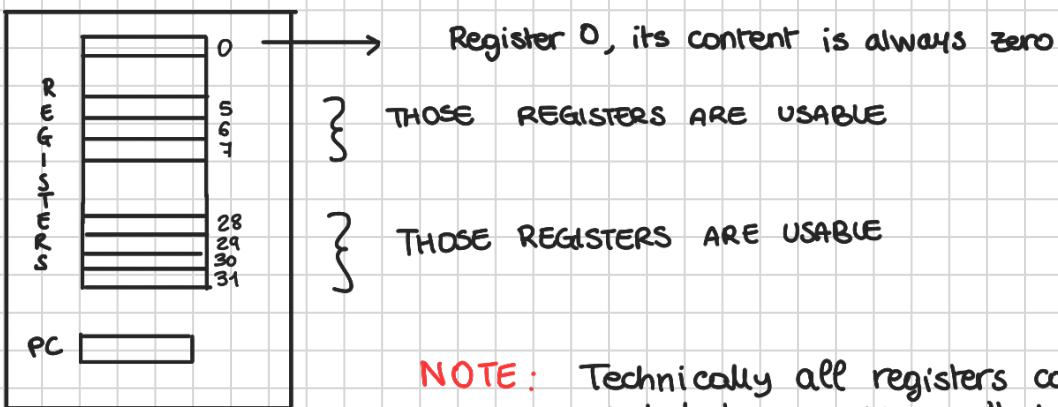
EXAMPLE: SUB 7, 3, 2

[ sub subtracts content between two registers and stores it in another register ]



**NOTE :** f<sub>3</sub> and f<sub>7</sub> change according to the instruction used

## RISC-V



**NOTE:** Technically all registers can be used but we assume that they are used for other purposes

## EXERCISE

$$R[5] \leftarrow R[28] + R[29] + R[30]$$

## SOLUTIONS:

- ① add 6, 28, 29
- ② add 5, 6, 30

or

- ① add 5, 28, 29
- ② add 5, 5, 30

more efficient because we don't use other registers and potentially overwrite them (as we did with register 6 in the previous one)

- HOW TO MOVE THE CONTENT OF ONE REGISTER INTO ANOTHER?

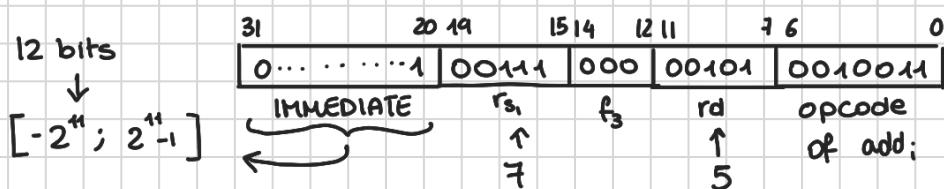
Example : move content of  $R[5]$  into  $R[7]$

- We can do this with the "add" instruction  
(add 7, 5, 0)
- This is why we don't have "mov" instruction

## ADDI (ADD IMMEDIATE)

- adds a number (IMMEDIATE) to the content of a register
- The immediate number is composed by 12 bits
- You can use addi to subtract a number too, as long as it's a number representable in 12 bits.

EX: addi 5, 7, 1 ( $R[5] \leftarrow R[7] + 1$ )

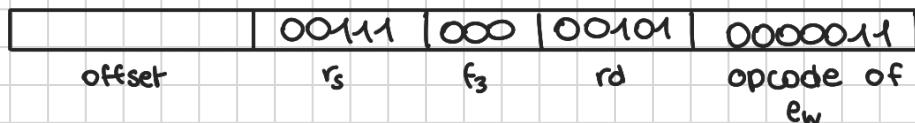


## LW ( LOAD WORD )

- Loads the content of an address in the memory (RAM) inside a register in the CPU and adds an offset (not always)
- Offset is the difference in terms of bits starting from our address. ( can be negative)
  - Ex : Register 7 has address of 0x10010000 so if i do, "lw 6, 4(7)" this won't load the content in 0x10010000 BUT 0x10010004

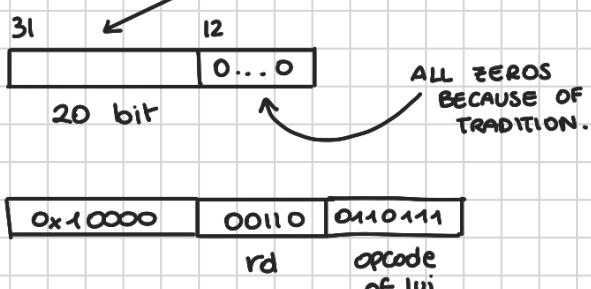
### EXAMPLE :

lw 5, offset(7) →  $R[5] \leftarrow \text{MEM}[R[7] + \text{offset}]$



## LUI ( LOAD UPPER IMMEDIATE )

- Loads a big number inside a register instead of using add;
- EX: lui 6, 0x10000 (this is a number, not address)



### EXERCISE :

x	0x10010000
y	0x10010004

Add x+y and move it to register 5.

Write the instructions.

## SOLUTION :

① lui  $\neq$ , 0x10010

- so what i'm doing is storing the address of "x" inside the register

② lw 5, 0( $\neq$ )

- Load word reads the ADDRESS inside register  $\neq$ , finds its content (which is our "x") and stores it inside register 5.

③ sw 6, 4( $\neq$ )

- reads the address inside register  $\neq$  but adds 4 to it ( $0x10010000 + 4 = 0x10010004$ ) This is now the address of "y" so this content is now stored in register 6.

④ add 5, 5, 6

- now we can add the contents of R[5] and R[6] ( $x+y$ ) and store it in R[5].

Now, if we want to take this result ( $x+y$ ) and store it in the memory at the address 0x10010008 we need a new instruction: STORES WORD (sw)

Takes the content of a register and stores inside the memory by reading the address (inside another register) ←

SW EXAMPLE : SW 5, offset(8)

31	25 24	20 19	15 14	12 11	$\neq$	6	0
offset 11-5	00101	01000	010	offset 4-0	0100011	op code sw	

$r_2$        $r_1$        $f_3$

Offset is made of 12 bits which are separated as you can see above

⑤ sw 5, 8( $\neq$ )

So, now we are storing the content of R[5] into the memory at the address found in R[ $\neq$ ] with an offset of 8 ( $0x10010000 + 8 = 0x10010008$ )

# RARS (RISC-V ASSEMBLER & RUNTIME SIMULATOR)

- HOW TO WRITE IN RARS:

TO ENTER DATA {

- .data # starts at 0x10010000
  - .word 13
  - .word 25 → content of words, can be any number
- .text # program starts at 0x00400000

lui 7, 0x10010  
lw 5, 0(7)  
lw 6, 4(7)  
add 5, 5, 6  
sw 5, 8(7)

CLOSE PROGRAM { addi 17, 0, 10 → encoding of exit  
ecall

## BEQ (BRANCH ==, BRANCH EQUAL)

- This instruction is like an if condition with jump. IF two registers have the same content then it jumps to another instruction at another address. The jump will be the offset.

EXAMPLE: beq 5, 6, offset

↳ if  $R[5] == R[6]$ :

$PC \leftarrow PC + \text{offset}$

**NOTE:** PC is the Program Counter and it contains the address of the next instruction

EXAMPLE:

		addresses of the instructions
	add 5, 0, 0	0x00400000
	add 6, 0, 0	0x00400004
	beq 5, 6, 8	0x00400008
	<u>add 5, 5, 1</u>	<u>0x0040000C</u>
	add 6, 6, 1	0x00400010

→ since  $R[5] = R[6]$ , there will be a jump of 8.  
So  $0x00400008 + 8 = 0x00400010$ .  
So the last instruction will be executed and the second-last will be discarded

## BNE ( BRANCH != , BRANCH NOT EQUAL)

- Opposite of Beq. If the content of two registers is different then jump to another instruction.

NOTE : you can implement loops with beq, bne

### EXAMPLE :

works with  
add too

```
← add; 5,0,0
      add; 6,0,-10
      add 5,5,6
      add; 6,6,-1
      bne 6,0,-8
```

```
add; 5,0,0
      add; 6,0,-10
      add 5,5,6
      add; 6,6,-1
      bne 6,0, ciclo
```

Basically, those programs will end  
only when R[6] will be equal to zero.