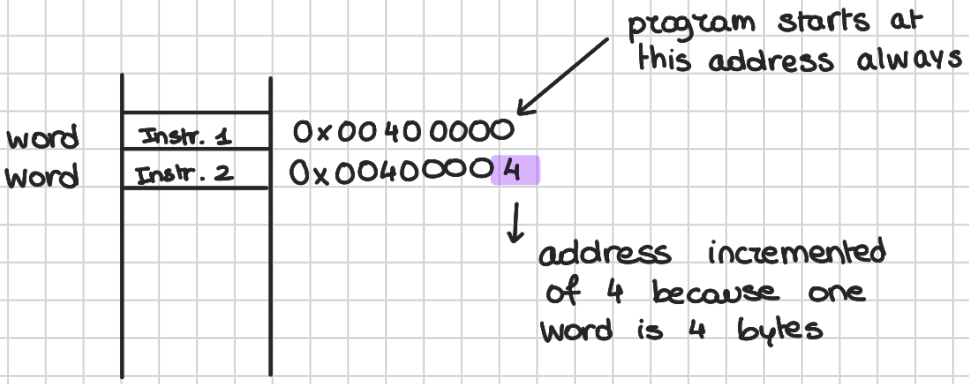
Computer Architecture (Unit 2)

DATA: 25/02/2002



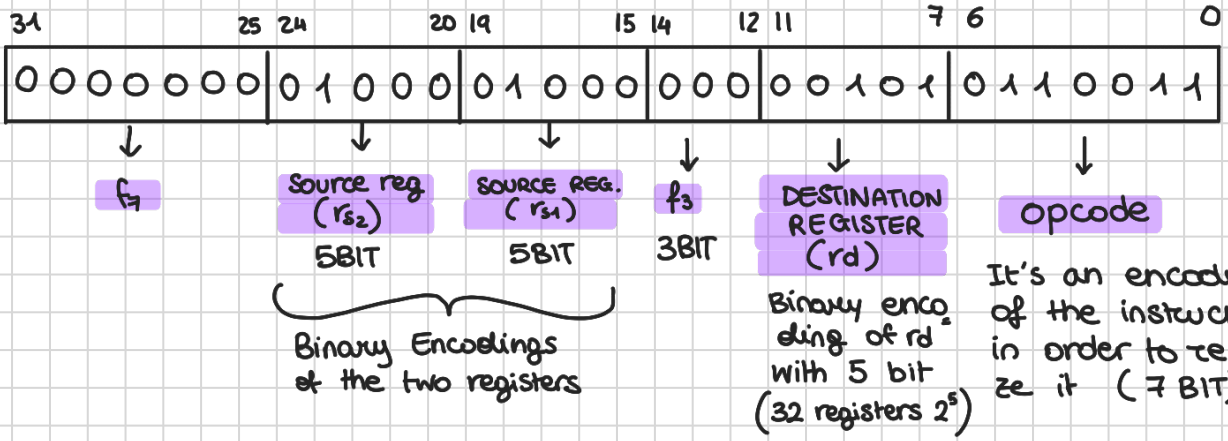
ASSEMBLY INSTRUCTION

= instructions

ADD

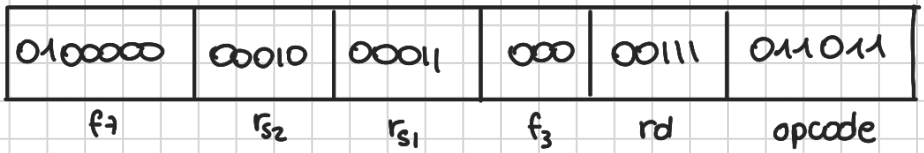
- Adds the content of two registers and stores into a destination register.

EXAMPLE: R[5] ← R[8] + R[8]
(R[5]: destination; R[8]: source register 1; R[8]: source register 2)



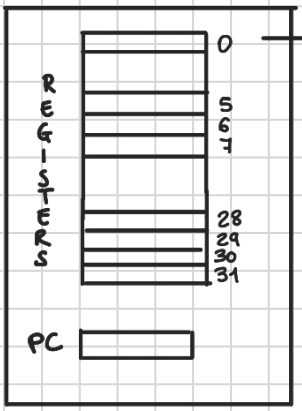
EXAMPLE: sub 7, 3, 2 (rd, rs1, rs2)

sub substracts content between two registers and stores it in another register



NOTE: f3 and f7 change according to the instruction used

RISC-V



Register 0, its content is always zero

THOSE REGISTERS ARE USABLE

THOSE REGISTERS ARE USABLE

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]

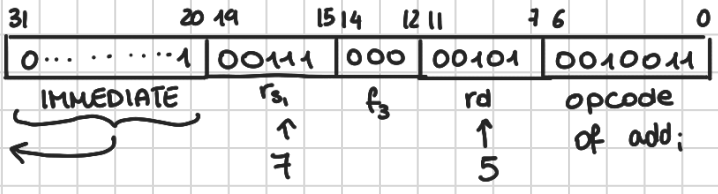
- we can do this with the "add" instruction (add 7, 5, 0)
- This is why we don't have "mov" instruction

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 substract a number too, as long as it's a number representable in 12 bits.

EX: addi 5, 7, 1 (R[5] ← R[7] + 1)

12 bits ↓ $[-2^{11}; 2^{11}-1]$



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 an 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

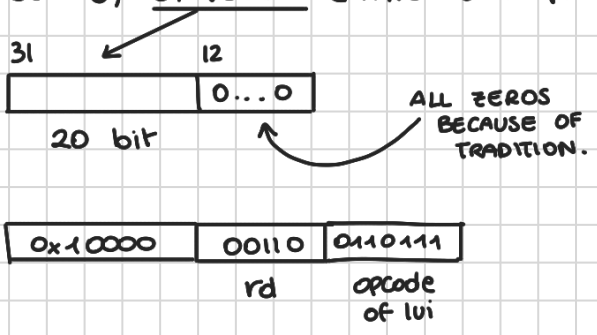
EXAMPLE:

lw 5, offset(7) ⟶ R[5] ⟵ MEM[R[7] + offset]

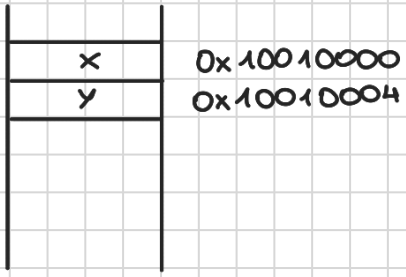
LUI (LOAD UPPER IMMEDIATE)

- Loads a big number inside a register instead of using addi

Ex: lui 6, 0x10000 (this is a number, not address)



EXERCISE:



Add x+y and move it to register 5.

Write the instructions.

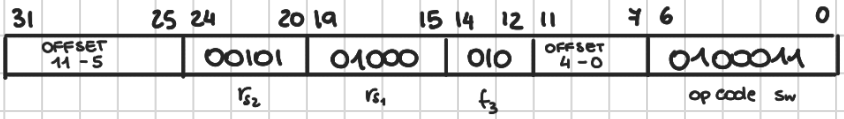
SOLUTION :

1. `lui 7, 0x10010`
   - so what i'm doin is storing the address of "x" inside the register
2. `lw 5, 0(7)`
   - load word reads the ADDRESS inside register 7, finds its content (which is our "x") and stores it inside register 5.
3. `lw 6, 4(7)`
   - reads the address inside register 7 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)



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

5. `sw 5, 8(7)`

   So, now we are storing the content of R[5] into the memory at the address found in R[7] 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 ← PC + offset

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

EXAMPLE :

```
add 5,0,0     0x00400000
add 6,0,0     0x00400004
beq 5,6,8     0x00400008
add 5,5,1     0x0040000C
add 6,6,1     0x00400010
```

addresses of the instructions

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

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