Aula 05 - 02/12/22 - Formatos de instruções

Representação em linguagem de máquina

Todas as instruções são traduzidas para binário pelo montador (assembler). Os códigos binários geradps são chamados de linguagem de máquina. A conversão é pautada em 3 formatos de representação: tipo R, I e J.

Formato tipo R

A instrução do tipo R são representadas num binário de 32 bits segregado da seguinte forma:
op | rs | rt | rd | shamt | funct

op	rs	rt	rd	shamt (shift amount)	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

onde:

→ **op**: código da operação (opcode), detecta que é uma instrução do tipo R.

→ **rs**: núm. do 1° registrador de origem, é um operando.

→ **rt**: núm. do 2° registrador de origem, é um operando.

→ **rd**: núm. do registrador de destino.

→ **shamt (shift amount)**: tamanho do deslocamento.

→ **funct**: código da função (complementa o opcode), determina a operação da instrução do tipo R.

São instruções do tipo R:

→ aritméticas

→ de deslocamento

→ lógicas

Ex.: **add $t0, $s1, $s2**

cód	$s1	$s2	$t0	0	add

0	17	18	8	0	32

⬇️

000000	10001	10010	01000	00000	10000

⬇️

00000010001100100100000000010000

Formato tipo I

São representados num binário de 32 bits da seguinte forma:

op	rs	rt	const. ou end.
6 bits	5 bits	5 bits	16 bits

→op: códg da operação

→ rs: registrador de origem

→ rt: registrador de destino (ou origem p/ sw (store word))

São instruções do tipo I:

→ imediatas.

→ de acesso à memória.

Obs: A capacidade máxima de uma constante é -2^{15} a $2^{15} - 1$.

sw $t0, 0($s0) → $t0 é um **rt**, 0 é uma **const** e $s0 é um **rs**.

Operações Lógicas

São instruções para manipulação de bits.

→ **shift left**: sll (shift left logical) reg1, reg2, shamt

→ **shift right**: srl (shift right logical) reg1, rg2, shamt

→ **e** lógico (bit a bit): and reg1, reg2, reg3 ⇒

```
and reg1, reg2, reg3
; reg1 = reg2 e reg3
; andi reg1, reg2, const -> instrução imediata
```

→ **ou** lógico (bit a bit): or reg1, reg2, reg3 ⇒

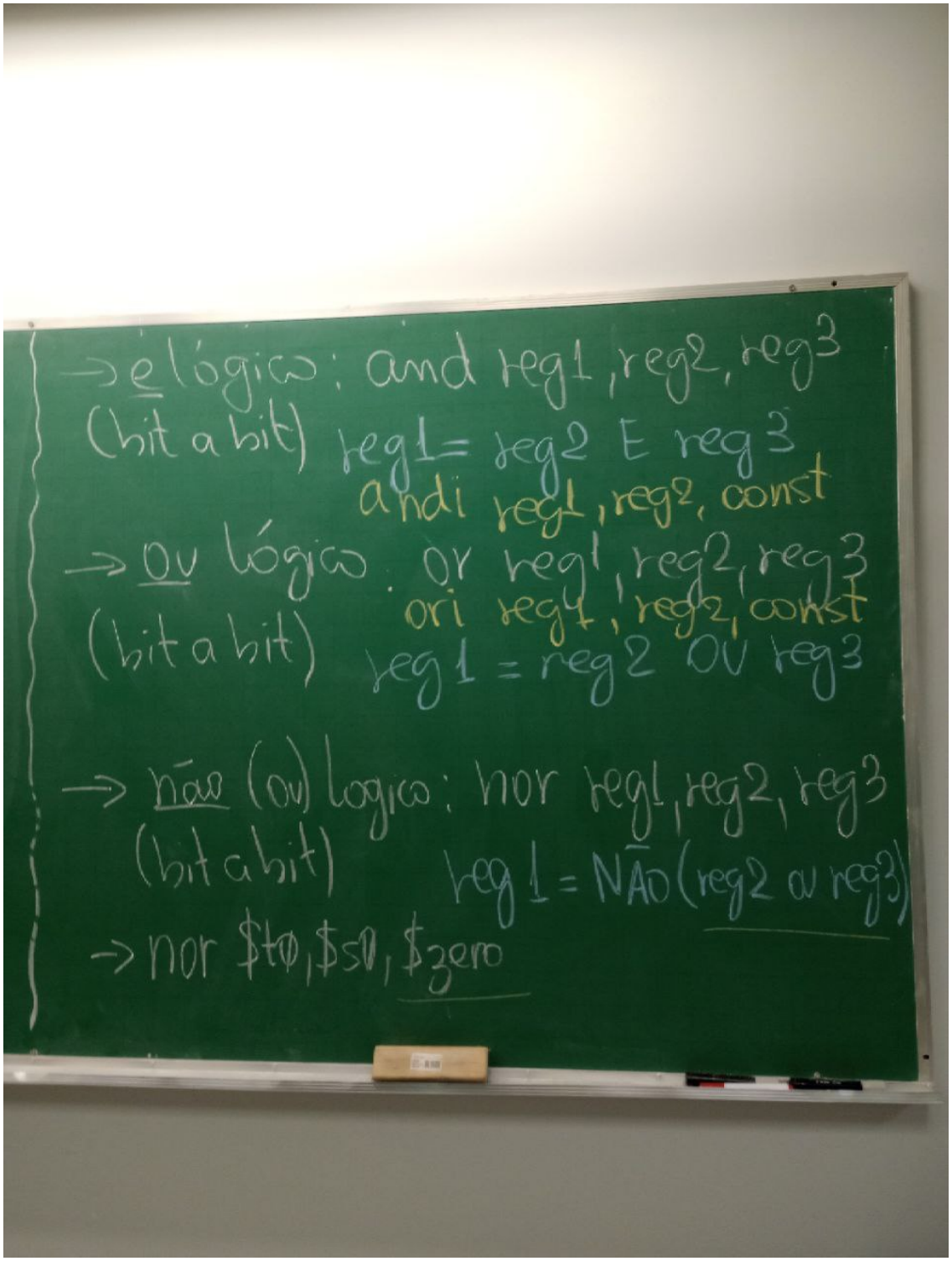
```
or reg1, reg, reg3
; reg1 = reg2 ou reg3
; ori reg1, reg2, const -> instrução imediata
```

→ **não** (ou) lógico (bit a bit): nor reg1, reg2, reg3 ⇒

```
nor reg1, reg2, reg3
; reg1 = não (reg2 ou reg3)
```

- **Obs:** Ele realizar bit a bit, significa pegar cada bit e usar o operador lógico, veja com operador **e** lógico:

```
; reg 2 = 0101
; E
; reg3 = 1101
;-------------
; reg1 = 0101 (fez E de 1 com 1, depois de 0 com 0, depois de 1 com 1 e por último de 0 com 1).
```


→ e lógico: and reg1, reg2, reg3
(bit a bit) reg1 = reg2 E reg3
andi reg1, reg2, const
→ ou lógico: or reg1, reg2, reg3
(bit a bit) ori reg1, reg2, const
reg1 = reg2 OU reg3
→ não (ou) lógico: nor reg1, reg2, reg3
(bit a bit) reg1 = NÃO (reg2 ou reg3)
→ nor $t0, $s0, $zero