

# Codificação de instruções

ASC I 2015/2016

Teresa Gonçalves



## Resumo

Sabemos que todas as instruções MIPS são codificadas em números de 32 bits.

Vamos ver como são codificadas.

- As instruções do tipo R são, genericamente, as que envolvem só registos (add, or, and, etc).
- O código máquina é uma palavra de 32 bits com a estrutura seguinte:

OpCode	Rs	Rt	Rd	SA	Funct
6	5	5	5	5	6

**OpCode** é o código de operação. Vale sempre 000000 para instruções do tipo R.

**Funct** codifica a função da instrução (código de 6 bits).

**Rs, Rt, Rd** correspondem aos registos usados numa instrução como “sub Rd, Rs, Rt”.  
Contêm os números 0 a 31 correspondentes aos registos **source**, **target** e **destination**.

**SA** é apenas usado nas instruções de shift (sll, srl, sra) para indicar o **Shift Amount**.

1 add \$t0, \$t1, \$t2 → add \$8, \$9, \$10

000000	01001	01010	01000	00000	100000
OpCode	Rs	Rt	Rd	SA	Funct

1 add \$t0, \$t1, \$t2 → add \$8, \$9, \$10

000000	01001	01010	01000	00000	100000
OpCode	Rs	Rt	Rd	SA	Funct

2 slt \$at, \$a1, \$s0 → slt \$1, \$5, \$16

000000	00101	10000	00001	00000	101010
OpCode	Rs	Rt	Rd	SA	Funct

1 add \$t0, \$t1, \$t2  $\rightarrow$  add \$8, \$9, \$10

000000	01001	01010	01000	00000	100000
OpCode	Rs	Rt	Rd	SA	Funct

2 slt \$at, \$a1, \$s0  $\rightarrow$  slt \$1, \$5, \$16

000000	00101	10000	00001	00000	101010
OpCode	Rs	Rt	Rd	SA	Funct

3 sll \$v0, \$t0, 3  $\rightarrow$  sll \$2, \$8, 3

000000	00000	01000	00010	00011	000000
OpCode	Rs	Rt	Rd	SA	Funct

A instrução nop não é uma instrução real. Sabendo que código máquina é 0x00000000, que instrução real é usada para no lugar do nop?

1 add \$t0, \$t1, \$t2 → add \$8, \$9, \$10

000000	01001	01010	01000	00000	100000
OpCode	Rs	Rt	Rd	SA	Funct

2 slt \$at, \$a1, \$s0 → slt \$1, \$5, \$16

000000	00101	10000	00001	00000	101010
OpCode	Rs	Rt	Rd	SA	Funct

3 sll \$v0, \$t0, 3 → sll \$2, \$8, 3

000000	00000	01000	00010	00011	000000
OpCode	Rs	Rt	Rd	SA	Funct

A instrução nop não é uma instrução real. Sabendo que código máquina é 0x00000000, que instrução real é usada para no lugar do nop? "sll \$zero, \$zero, 0".

- Genericamente, as instruções do tipo I contêm em si próprias um número que é imediatamente usado na sua execução. Exemplos de instruções deste tipo são `addi`, `lw`, `beq`.
- O código máquina tem a seguinte estrutura:

OpCode	Rs	Rt	Immediate
6	5	5	16

**OpCode** é o código da operação, cada instrução do tipo I tem um código diferente.

**Rs, Rt** o registo **source** sempre lido, enquanto **target** pode ser lido ou escrito consoante a operação.

**Immediate** número de 16 bits a usar imediatamente na execução da instrução. O número pode ser interpretado com ou sem sinal, dependendo da instrução: aritméticas com sinal, lógicas sem sinal.



```
addi Rt, Rs, Immediate  
lw Rt, Immediate(Rs)  
sw Rt, Immediate(Rs)
```

Exemplos:

**1** addi \$t0, \$t1, 7  $\longrightarrow$  addi \$8, \$9, 7

001000	01001	01000	0000000000000111
OpCode	Rs	Rt	Immediate

```
addi Rt, Rs, Immediate  
lw Rt, Immediate(Rs)  
sw Rt, Immediate(Rs)
```

Exemplos:

1 addi \$t0, \$t1, 7  $\longrightarrow$  addi \$8, \$9, 7

001000	01001	01000	0000000000000111
OpCode	Rs	Rt	Immediate

2 lw \$t0, 4(\$s0)  $\longrightarrow$  lw \$8, 4(\$16)

100011	10000	01000	0000000000000100
OpCode	Rs	Rt	Immediate

As instruções de **branch** também são do **tipo I**:

`beq Rs, Rt, LABEL`

onde os 16 bits “immediate” correspondentes à LABEL são o número de instruções que se pretende avançar caso o branch seja executado. Se for negativo então é um recuo (e.g. num ciclo)

As instruções a saltar contam a partir do “delay slot”. Exemplo:

```
beq $t0, $t1, A      # instruções a saltar
nop                  # <- contam a partir desta linha.
```

```
add $t1, $t2, $t3
```

**A:** or \$v0, \$t0, \$t3

Do “delay slot” até à label A são 2 instruções:

000100	01000	01001	0000000000000010
OpCode	Rs	Rt	Immediate

As instruções j e jal são do tipo J.

j LABEL           # salta para instrução com endereço  
nop               # correspondente à LABEL

Supondo que o endereço correspondente à LABEL é 0x0040019c,  
que em binário se escreve

0000 0000 0100 0000 0000 0001 1001 1100

o código máquina é construído apenas com a parte a azul deste  
endereço:

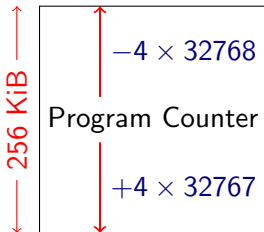
OpCode	0000 0100 0000 0000 0001 1001 11
6	26

Os dois bits da direita são omitidos porque são sempre zero  
(endereços são múltiplos de 4).

Os quatro bits da esquerda são os do PC no momento em que o  
salto é tomado (i.e., vem da instrução seguinte ao jump).

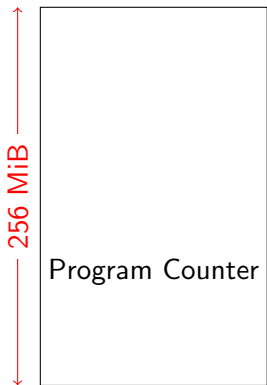
A gama de endereços atingíveis com jumps ou branches é diferente devido aos diferentes modos de representação e cálculo do endereço de destino.

Branch:



O endereçamento é relativo ao Program Counter.

Jump:



O endereçamento é absoluto.