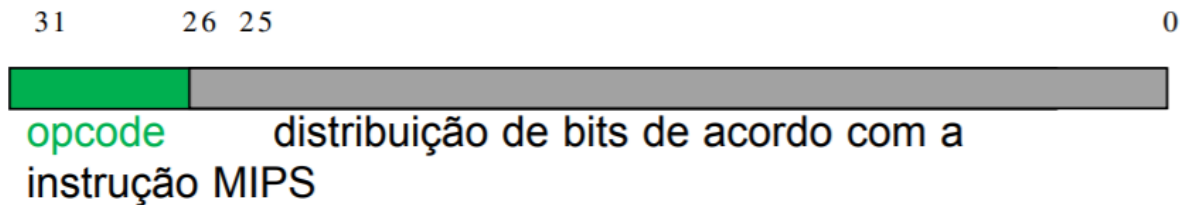


Instruções MIPS

Por definição, todas as instruções são no formato de 32 bits e tem o seu “opcode” (código de instrução) no tamanho de 6 bits, o seu modo de endereçamento é codificado juntamente com o opcode.



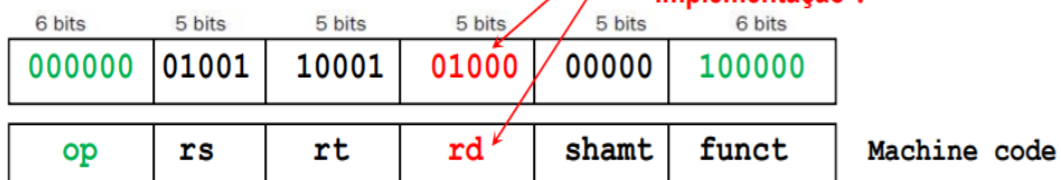
Aritmética em linguagem MIPS

Instruções, assim como registradores de palavras de dados, tem tamanho de 32 bits

Exemplo: add \$t0, \$t1, \$s1 (assembly MIPS)

Nesse formato, podemos perceber que rs e rt está igual na ordem dos operandos, já o rd

Formato de uma Instrução R típico:



Onde:

- op – código de operação
- rs e rt – registradores fonte (rs = \$t1, rt = \$s1);
- rd – registrador destino (rd = \$t0);
- shamt – (shift amount) é usado em instruções de deslocamento;
- funct – código de função da ULA (opcode para qualquer operação da ULA=00000)


Instruções tipo-I

São instruções utilizadas para a transferência de dados (lw e sw) e possuem formato de instrução diferente do tipo-R (registradores).

Nesse exemplo, o destino é o rt (\$s2) e a origem é rb(\$t1) e o deslocamento imediato (40)

Exemplo: `lw $t1, 40($s2)` `# $t1 = MEM[$s2+40]`

100011	10010	01001	0000000000101000
op	rb	rt	16 bit number (immediate)



- rb – registrador-base para o cálculo do endereço de memória
- rt – registrador-destino (para lw) ou registrador-fonte (para sw)

Modelos de Referenciamento

Endereçamento de operandos

- imediato – valor já faz parte da instrução;
- registrador – registrador usado para dados.

Referência à Memória (instruções load e store)

- label – endereço fixo que faz parte da instrução;
- Indireto – registrador contém um endereço;
- Endereçamento de Base – campo de um registrador;
- Endereçamento indexado – elementos de um vetor

Controle de Fluxo

Quando uma decisão é tomada em tempo de execução de acordo com o que foi codificado para definir a próxima instrução do programa (valor de PC).

Instruções MIPS de desvio condicional:

`bne $t0, $t1, Label` `#($s<>$t)→PC=PC+(endif1<<2) else PC=PC+4`

`beq $t0, $t1, Label` `#($s==$t)→PC=PC+(endif1<<2) else PC=PC+4`

Em C: `if (i!=j) h=i+j;` **Em MIPS:** `bne $s0, $s1, Label`

`...`
 `Label: add $s3, $s0, $s1`
 `...`

As instruções `bne` e `beq` obedecem ao princípio da regularidade, operando sobre registradores Base para construções do tipo if-then-else.

Nesse exemplo, o registrador base seria (`$s2`) e o registrador a ser comparado seria o (`$t1`)

Tradução da Expressão: se o `$s2` for igual a `$t1`, multiplica o endereço (`end`) por 4 e soma ao PC.

`end` = quantas instruções deve ir ou voltar.

Ex: `beq $s2, $t1, end`

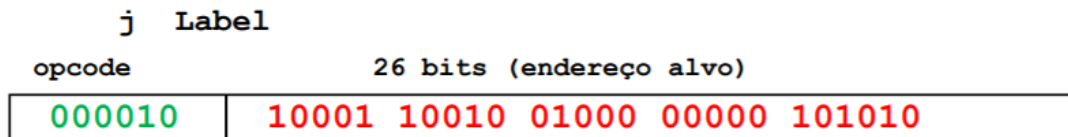
`if1 #($s2==$t1)→PC=PC+(endif1<<2)else PC=PC+4;`

000100	10010	01001	0000010000111000
op	rs	rt	16 bit number (target)

Onde:

- rs – registrador-base para a comparação;
- rt – registrador-teste a ser comparado;
- label – endereço-alvo do desvio (usado no cálculo do endereço a ser escrito em PC em caso do desvio se realizar)

Instruções MIPS de desvio incondicional (jumps):



- Instruções usadas para a construção de loops, como em:

```
if (i!=j)                beq $s4, $s5, Lab1
    h=i+j;              add $s3, $s4, $s5
else                    j Lab2
    h=i-j;              Lab1:  sub $s3, $s4, $s5
                        Lab2:  ...
```

Instruções MIPS de desvio incondicional (Se Menor Igual):

Temos: beq, bne. E para uma nova instrução tipo *branch-if-less-than*?

```
slt $t0,$s1,$s2                      if $s1 < $s2 then $t0 = 1
                                    else $t0 = 0
```

Usando beq, bne e \$t0 pode se implementar a "lógica requerida"

000000	10001	10010	01000	00000	101010
--------	-------	-------	-------	-------	--------

Instruções MIPS com constantes: tipo-I (1 operando imediato)

```
addi $29, $29, 4
slti $8, $18, 10
andi $29, $29, 6
ori $29, $29, 4
```

Instruções grandes

O opcode ocupa 6 bits e assim restam apenas 26 bits para o armazenamento de uma instrução, quando a constante excede o valor de 16 bits, é necessário dividir ela em duas instruções de 16 bits (lui e ori).

Text Segment				
Bkpt	Address	Code	Basic	Source
	0x00400000	0x3c010040	lui \$1,0x00000040	\$: li \$s0, 0x00400000 # save return adress in \$s0
	0x00400004	0x34300000	ori \$16,\$1,0x00000000	

Porque a constante excede o valor de 16 bits, então o lui irá colocar os 16 bits mais significativos e o ori, os 16 bits menos significativos, passando assim a constante para os registradores de 32 bits.