Operações Lógicas

Cinco são as operações lógicas definidas no MIPS, conforme mostra a Tabela 1:

Tabela1: Operações Lógicas MIPS

Operações Lógicas	Instruções MIPS
Shift à direita	SLL
Shift à esquerda	SRL
And bit a bit	AND, ANDI
Or bit a bit	OR, ORI
Not bit a bit	NOR

As instruções de operações lógicas ajudam a simplificar o empacotamento e o desempacotamento dos bits em **words**. Todos lembram o que é uma word? Sim? Não? Vale a pena relembrar né? A unidade de informação que trabalhamos em máquinas digitais, como a grande maioria já deve saber, são os bits, isto é, os zeros e uns. Quando os zeros e uns são agrupados em uma determinada quantidade eles recebem nomes, conforme mostra a Tabela 2:

Tabela 2: Bits

Bit	1 bits	0
Nibble	4 bits	0000
Byte	8 bits	0000 0000
Word	16 bits	0000 0000 0000 0000
Double Word	32 bits	0000 0000 0000 0000 0000 0000 0000 0000
Quad Word	64 bits	0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000

Um **bit** é correspondente a um único bit, isto é, ou ele tem o valor zero, ou tem o valor um, um **nibble** corresponde a 4 bits, um byte são 8 bits, e assim, consecutivamente, até chegarmos no número de bits máximo.

A Tabela 2 apresenta Word como uma quantidade de 16 bits, entretanto, Word é usada atualmente como sinônimo de PALAVRA. Uma Word (ou Palavra) é a quantidade de bits que se consegue transmitir, portanto, não é obrigatoriamente 16 bits, pode ser 32 bits. Esclarecida esta questão, vamos estudar agora a operação SHIFT e, nos artigos posteriores, veremos as outras operações.

Operação SHIFT

Shift significa **deslocamento**, assim um **shift à esquerda** significa um deslocamento à esquerda e, um **shift à direita**, significa um deslocamento à direita. Mas, você está aí se perguntando: "ora, bolas, carambolas, deslocamento de quem?". Pois bem, é o deslocamento dos bits da palavra. Os **shifts** movem todos os bits de uma palavra, preenchendo os bits que ficaram vazios com zeros. Vamos ver um exemplo para entender melhor. Suponha que um registrador tivesse armazenado o valor 9:

0000 0000 0000 0000 0000 0000 **1001**

Agora imagine que é executada uma instrução de deslocamento que comanda o deslocamento de 4 bits à esquerda. O novo valor desse registrador será:

0000 0000 0000 0000 0000 **1001** 0000

Conseguiram notar a diferença? Agora o valor não é mais o 9, mas sim 144. Portanto, ao realizarmos o deslocamento de 4 bits à esquerda, mudamos o valor decimal que estava armazenado no registrador também.

SLL significa **Shift Left Logical** (deslocamento lógico à esquerda) e SRL significa **Shift Right Logical** (deslocamento lógico à direita), que são os mnemônicos e nomes das instruções MIPS. Assim, o exemplo acima fica da seguinte forma na linguagem de montagem:

```
SLL $t2, $s0, 4                # $t2 = $s0 << 4 bits
```

A linguagem de máquina fica da seguinte forma:

```
SLL $10, $16, 4
```

A representação fica assim:

opcode	rs	rt	rd	shamt	funct
0	0	16	10	4	0
6 bit	5 bits	5 bits	5 bits	5 bits	6 bits

Para finalizar a compilação, o código da máquina:

000000 00000 10000 01010 00100 000000

Assim, 4 é o número de bits que deve ser deslocado, portanto, ele deve ser colocado no campo **SHAMT**, que significa **Shift Amount**. Esse campo armazena a quantidade de deslocamento e é usado apenas para instruções de deslocamento. $t2 ($10) é o registrador que armazenará o resultado, portanto, registrador destino e, por isso, é colocado no campo RD. $s0 ($16) deve ser inserido no campo RT. O campo RS não é utilizado, por isso colocamos zero. A codificação padrão para SLL é zero pra o campo de código e para funct. Portanto, SLL é uma instrução do tipo R.

Em nosso exemplo o número decimal inicial era 9 e o resultado do deslocamento em 4 bits foi 144. Se vocês fizerem 9 x 2^4 o resultado será 144, ou seja, a instrução SLL é equivalente a uma multiplicação por 2^i.

A instrução SRL é muito parecida com a SLL, a diferença básica é que a SRL desloca os bits para a direita. O código de FUNCT para SLL é 2 e o opcode é 0. Vejamos um exemplo:

0000 0000 0000 0000 0000 **1001** 0000 = 144_{10}

Deslocando 4 bits a direita:

0000 0000 0000 0000 0000 0000 **1001** = 9_{10}

Assim a instrução na linguagem de montagem fica da seguinte forma:

```
SRL $t2, $s0, 4                    # $t2 =  $s0 >> 4 bits
```

Na linguagem de máquina fica:

SRL $10, $16, 4

A representação fica da seguinte forma:

opcode	rs	rt	rd	shamt	funct
0	0	16	10	4	2
6 bit	5 bits	5 bits	5 bits	5 bits	6 bits

Finalizando a compilação, o código de máquina:

000000 00000 10000 01010 00100 000010