Capítulo 2

Instruções: A Linguagem de Máquina

Conjunto de instruções MIPS

- Usado como exemplo ao longo do livro
- Stanford MIPS comercializado pela MIPS Technologies (www.mips.com)
- Grande parte do mercado embarcados
  - Aplicações em eletrônicos de consumo, equipamentos de rede/armazenamento, câmeras, impressoras, ...
- Típico de muitas ISA modernas
  - Veja MIPS Reference Data Card e os apêndices B e C

Operações Aritméticas

- Somar e subtrair, três operandos
  - Duas fontes e um destino

```
add a, b, c  # a recebe b + c
```

- Todas as operações aritméticas têm esta forma
- *Princípio de Projeto 1:* Simplicidade favorece regularidade
  - Regularidade faz implementação mais simples
  - Simplicidade permite maior desempenho e menor custo

Exemplo Operação Aritmética

- Código C :

```
f = (g + h) - (i + j);
```

- Código MIPS compilado:

```
add t0, g, h   # temp t0 = g + h
add t1, i, j   # temp t1 = i + j
sub f, t0, t1  # f = t0 - t1
```

Exemplo Operação Aritmética

- Código MIPS compilado (Linguagem Assembly)

```
 add $t0, $s1, $s2 # temp t0 = g + h
add $t1, $s3, $s4 # temp t1 = i + j
sub $s0, $t0, $t1 # f = t0 - t1
```

- Código MIPS compilado (Linguagem Máquina)

```
 0x02324020
 0x02744820
 0x01098022
```

```
 0000.0010.0011.0010.0100.0000.0010.0000
 0000.0010.0111.0100.0100.1000.0010.0000
 0000.0001.0000.1001.1000.0000.0010.0010
```

Operandos em Registradores

- Instruções aritméticas utilizam operandos em registradores
- MIPS tem um conjunto de 32 registradores 32 bits
  - Usado para dados acessados com frequencia
  - Numerados de 0 a 31
  - Dados de 32 bits chamados de “palavra” (“word”)
- Nomes no Assembler
  - $t0, $t1, …, $t9 para valores temporários
  - $s0, $s1, …, $s7 para variáveis a serem salvas
- *Princípio de Projeto 2:* Menor é mais rápido
  - Somente 32 registradores

A Constante Zero

- No MIPS registrador 0 ($zero) é a constante 0
  - Não pode ser alterado
- Útil em operaçõs comuns
  - Ex. mover entre registradores

```
add $t2, $s1, $zero
```

Exemplo Operando em Registrador

- Código C:

```
f = (g + h) - (i + j);
```

  - f, …, j no $s0, …, $s4

- Código compilado MIPS:

```
add $t0, $s1, $s2
add $t1, $s3, $s4
sub $s0, $t0, $t1
```

Operando em Memória

- Memória principal usada para dados compostos
  - Arrays, estruturas, dados dinâmicos
- Para aplicar operações aritiméticas
  - Carregar valores da memória para registradores
  - Armazenar resultado do registrador para memória
- Memória endereçada por byte
  - Cada endereço é identificado por um byte (8 bits)
- As palavras são alinhadas na memória
  - Endereço deve ser um múltiplo de 4
- MIPS é Big Endian
  - Byte mais significativo no menor endereço da word
  - Little Endian: byte menos significativo no menor endereço da word

Operando na Memória Ex. 1

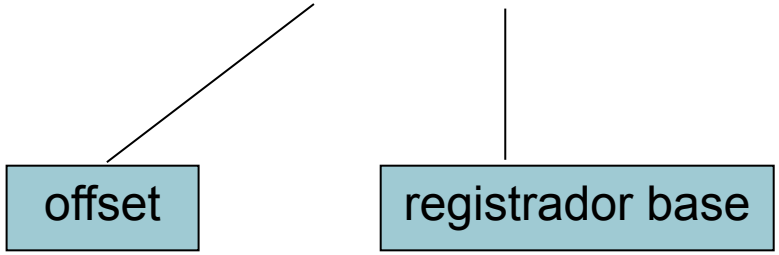
- Código C:

```
g = h + A[8];
```

  - g nem $s1, h em $s2, endereço base de A em $s3

- Código compilado MIPS:
  - Índice 8 requer offset de 32
    - 4 bytes por palavra

```
lw  $t0, 32($s3)     # load word
add $s1, $s2, $t0
```



Operando na Memória Ex. 2

- Código C:

```
A[12] = h + A[8];
```

  - h no $s2, endereço base de A no $s3

- Código compilado MIPS:
  - Índice 8 requer offset de 32
  - Índice 12 requer offset de 48

```
lw  $t0, 32($s3)    # load word
add $t0, $s2, $t0
sw  $t0, 48($s3)    # store word
```

Operandos Imediatos

- Dados constantes especificados na instruçao

```
addi $s3, $s3, 4
```

- Não existe instrução de subtração imediata
  - Basta usar uma constante negativa

```
addi $s2, $s1, -1
```

- *Princípio Projeto 3:* Faça o caso comum rápido
  - Constantes pequenas são comuns
  - Operando imediato evita uma instrução load

A Constante Zero

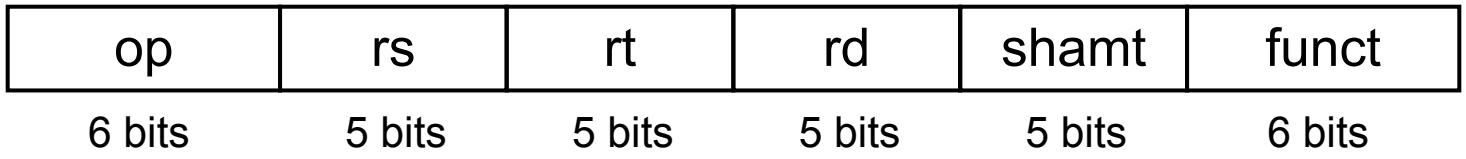
- No MIPS registrador 0 ($zero) é a constante 0
  - Não pode ser alterado
- Útil em operaçõs comuns
  - Ex. mover entre registradores

```
add $t2, $s1, $zero
```

Representando Instruções

- Instruções são codificadas em binário
  - Chamada de código de máquina
- Instruções MIPS
  - Codificadas como palavras de 32 bits
  - Pequeno número de formatos de código de operação (opcode), número registradores, …
  - Regularidade!
- Numero de registradores
  - $t0 – $t7 são os reg. 8 – 15
  - $t8 – $t9 são os reg. 24 – 25
  - $s0 – $s7 são os reg. 16 – 23

Instruções MIPS Formato R



- Campos da instrução
  - op: código da operação (opcode)
  - rs: registrador com primeiro operando origem
  - rt: registrador com segundo operando origem
  - rd: número do registrador destino
  - shamt: **sh**ift **am**oun**t** (00000 por enquanto)
  - funct: código de função (estende opcode)

Exemplo do Formato R

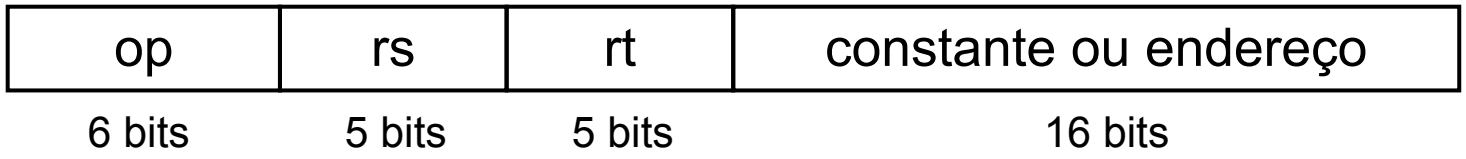
op	rs	rt	rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

```
add $t0, $s1, $s2
```

special	$s1	$s2	$t0	0	add
0	17	18	8	0	32
000000	10001	10010	01000	00000	100000

$0000.0010.0011.0010.0100.0000.0010.0000_2 = 02324020_{16}$

Instruções MIPS Formato I



- Instruções aritméticas com imediado e load/store
  - rt: número do registrador de destino ou origem
  - Constante: -2^{15} até $+2^{15} - 1$
  - Endereço: endereço adicionado ao endereço base no registrador rs
- *Princípio Projeto 4:* Bom projeto exige bons compromissos
  - Manter todas as instruções com o mesmo tamanho
  - Exigindo diferentes tipos de formatos para diferentes tipos de instruções

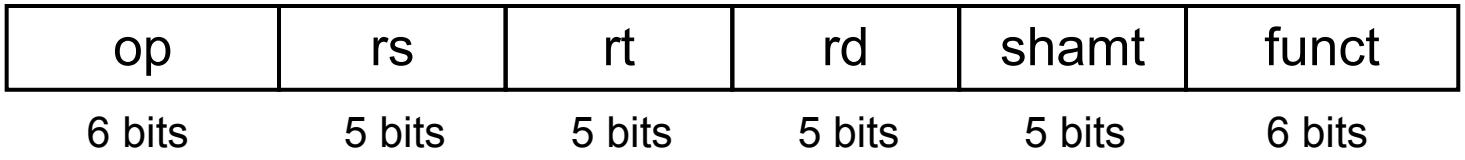
Operações Lógicas

- Instruções para manipulação bit a bit

Operação	C	Java	MIPS
Shift left	<<	<<	`sll`
Shift right	>>	>>>	`srl`
Bitwise AND	&	&	`and, andi`
Bitwise OR	|	|	`or, ori`
Bitwise NOT	~	~	`nor`

- Útil para extração e inserção de grupos de bits em uma palavra

Operações de Shift

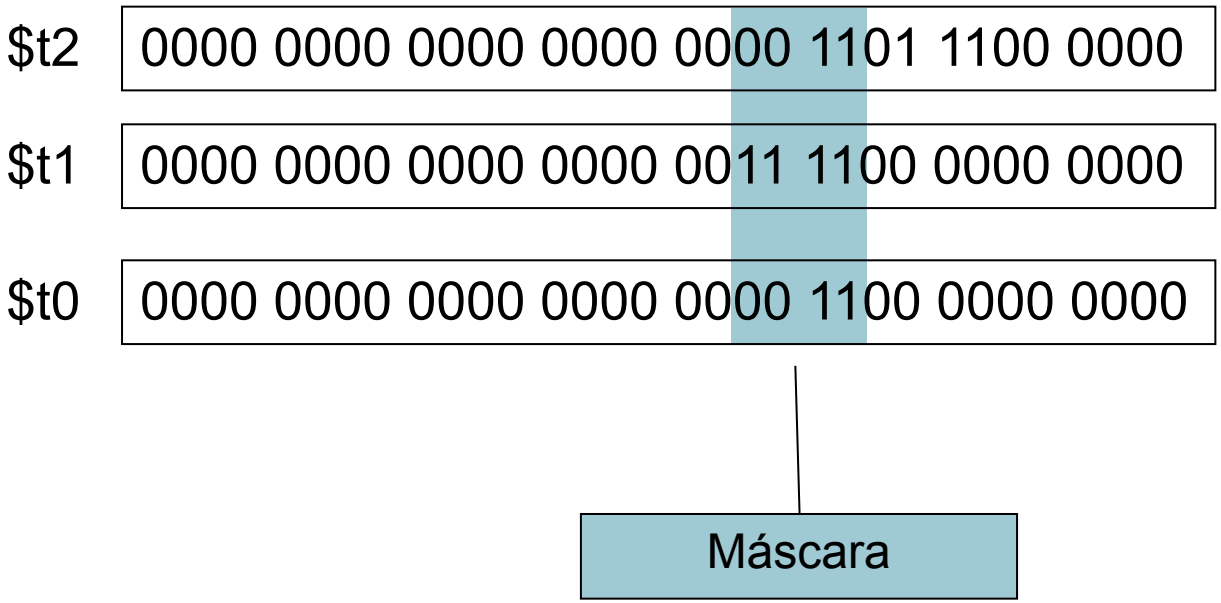


- shamt: quantas posições deslocar
- Shift left logical
  - Desloca para esquerda preenchendo com bits 0
  - sll por i bits multiplica por 2^i
- Shift right logical
  - Desloca para direita preenchendo com bits 0
  - srl por i bits divide por 2^i (somente unsigned)

Operações AND

- Útil para mascarar bits de uma palavra
  - Seleciona alguns bits, limpa outros com 0

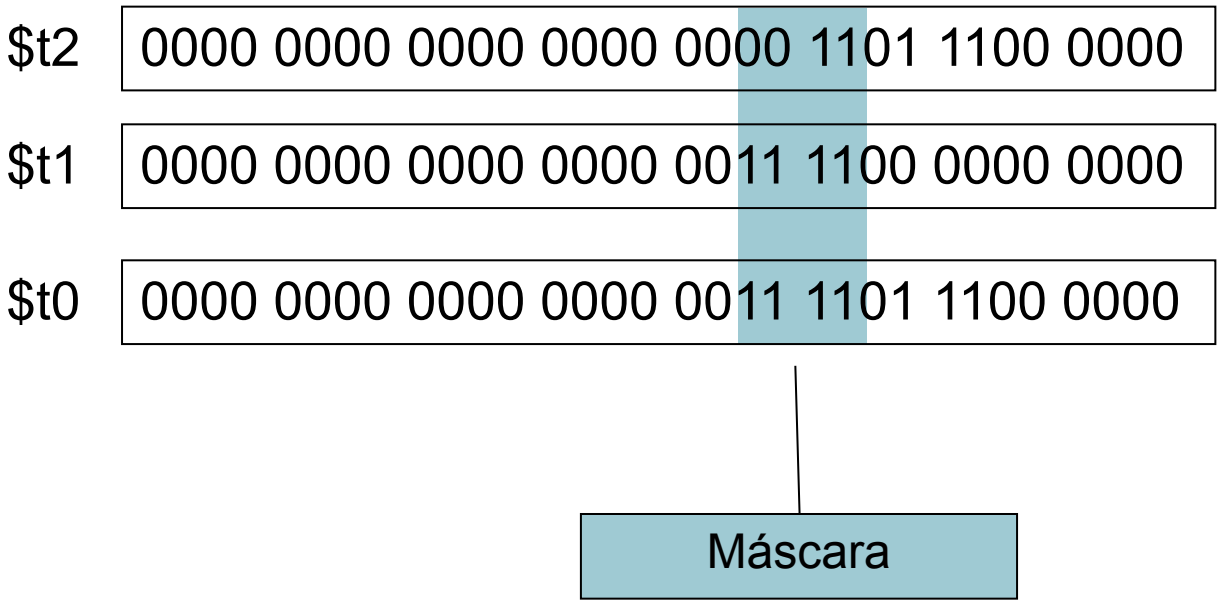
```
and $t0, $t1, $t2
```



Operações OR

- Útil para incluir bits em uma palavra
  - Definir alguns bits em 1 e deixar outros inalterados

```
or $t0, $t1, $t2
```



Operações NOT

- Útil para inverter bits de uma palavra
  - Mudar 0 para 1, e 1 para 0
- MIPS tem a operação NOR
  - a NOR b == NOT (a OR b)

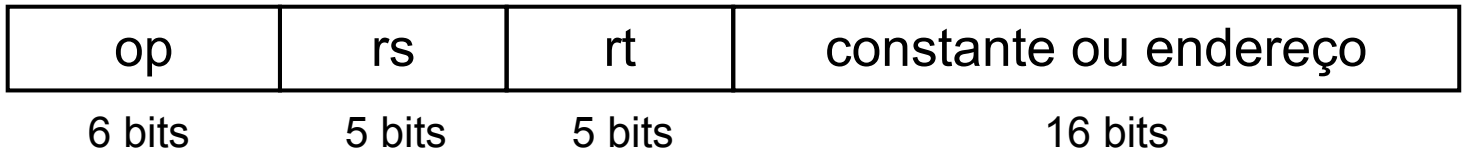
```
nor $t0, $t1, $zero
```

Registrador 0: sempre lê zero

$t1	0000 0000 0000 0000 0011 1100 0000 0000
$t0	1111 1111 1111 1111 1100 0011 1111 1111

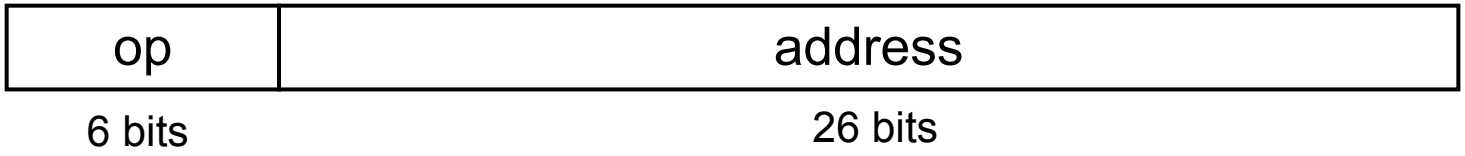
Operações Condicionais - Formato I

- Desvia para instrução marcada se condição for verdade
  - Caso contrário, continue sequencialmente
- `beq rs, rt, L1 # branch if equal`
  - if (rs == rt) desvia para instrução marcada L1;
- `bne rs, rt, L1 # branch if not equal`
  - if (rs != rt) desvia para instrução marcada L1;



Intruções MIPS Formato J

- Jump (`j` e `jal`) poder ser qualquer local no segmento text (código programa)
  - Desvio incondicional
  - Codificado o endereço completo na instrução



- Endereçamento pseudodireto
  - Endereço destino = $PC_{31\ldots28}$: (endereço × 4)
  - O endereço de jump são os 26 bits da instrução concatenados com os bits mais altos do PC

Referências

- Capítulo 2 - “Organização e Projeto de Computadores - A Interface Hardware/Software, David A. Patterson & John L. Hennessy, Campus, 4 edição, 2013.