Linguagem de montagem

3. Representação de inteiros, formato de instruções, instruções lógicas e deslocamento

Prof. John L. Gardenghi
Adaptado dos slides do livro

O zero

- O registrador MIPS `$zero` representa a constante 0
  - Não deve ser sobrescrita
- Evitar utilizar a constante 0 em instruções imediatas
  - E.g., mover dados entre registradores

    `add $t2, $s1, $zero  (move $t2, $s1)`
  - E.g., inicializar com zero

    `add $t1, $zero, $zero (move $t1, $zero)`

Inteiros binários sem sinal

- Dado um número binário $x = x_{n-1}x_{n-2}\ldots x_1x_0$

$$x = x_{n-1}2^{n-1} + x_{n-2}2^{n-2} + \cdots + x_1 2^1 + x_0 2^0$$

- Varia de 0 a $+2^n - 1$
  - Com n=32, de 0 to +4,294,967,295
- Exemplo
  - $0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 1011_2$
    $= 0 + \ldots + 1\times2^3 + 0\times2^2 + 1\times2^1 + 1\times2^0$
    $= 0 + \ldots + 8 + 0 + 2 + 1 = 11_{10}$

Inteiros com sinal (comp. a 2)

- Dado um número binário $x = x_{n-1}x_{n-2}\ldots x_1x_0$

$$x = -x_{n-1}2^{n-1} + x_{n-2}2^{n-2} + \cdots + x_1 2^1 + x_0 2^0$$

- Varia de -2^{n-1} a $+2^{n-1} - 1$
  - Com n=32, de –2,147,483,648 to +2,147,483,647
- Exemplo
  - $1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1100_2$
    $= -1\times2^{31} + 1\times2^{30} + \ldots + 1\times2^2 + 0\times2^1 + 0\times2^0$
    $= -2{,}147{,}483{,}648 + 2{,}147{,}483{,}644 = -4_{10}$

Inteiros com sinal (comp. a 2)

- O bit 31 é o bit de sinal
  - 1 para negativo
  - 0 para não-negativo
- Números não negativos tem a mesma representantação em inteiros sem sinal ou complemento a 2
- Números específicos
  - 0: 0000 0000 … 0000
  - –1: 1111 1111 … 1111
  - Mais negativo (- 2^{31}): 1000 0000 … 0000
  - Mais positivo(2^{31} - 1): 0111 1111 … 1111

Negação com sinal

- Calcula o complemento e soma 1
  - Complemento: transformar 1 → 0, 0 → 1

$$x + \bar{x} = 1111\ldots111_2 = -1$$

$$\bar{x} + 1 = -x$$

- Exemplo: negar +2
  - $+2 = 0000\ 0000\ \ldots\ 0010_2$
  - $-2 = 1111\ 1111\ \ldots\ 1101_2 + 1$
    $= 1111\ 1111\ \ldots\ 1110_2$

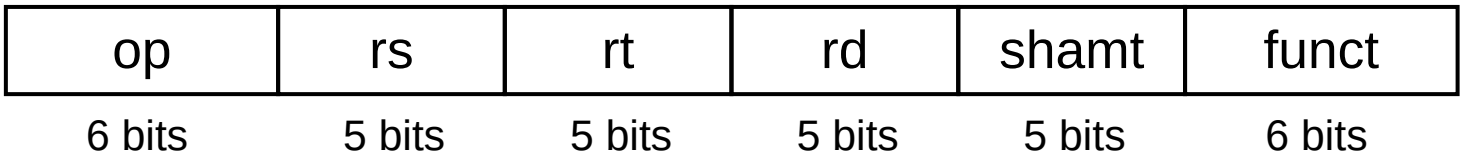
Extensão de sinal

- Consiste em representar um número com mais bits
  - Objetivo: preservar o valor numérico
- No conjunto de instruções MIPS
  - `addi`: extende o valor imediato
  - `lb`, `lh`: extende o byte/meia palavra carregado
- Replica o bit de sinal para a esquerda
- Exemplos: 8-bit para 16-bit
  - +2: 0000 0010 => 0000 0000 0000 0010
  - –2: 1111 1110 => 1111 1111 1111 1110

Representando instruções

- As instruções são codificadas em binário
  - Processo de montagem: gera código de máquina
- Instruções MIPS
  - Codificadas como palavras de 32 bits
  - Formato de codificação padrão
- Para codificar, os registradores são numerados
  - $t0 – $t7 → 8 – 15; $t8 – $t9 → 24 – 25
  - $s0 – $s7 → 16 – 23
- O processo de montagem é facilmente reversível, diferentemente da compilação.

Formato do tipo-R



- Campos
  - op: código da operação (opcode)
  - rs: número do primeiro registrador de origem
  - rt: número do segundo registrador de origem
  - rd: número do registrador de destino
  - shamt: quantidade de shift (00000 for now)
  - funct: código da função (complementar a opcode)

Exemplo de formato tipo-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

$00000010001100100100000000100000_2 = 02324020_{16}$

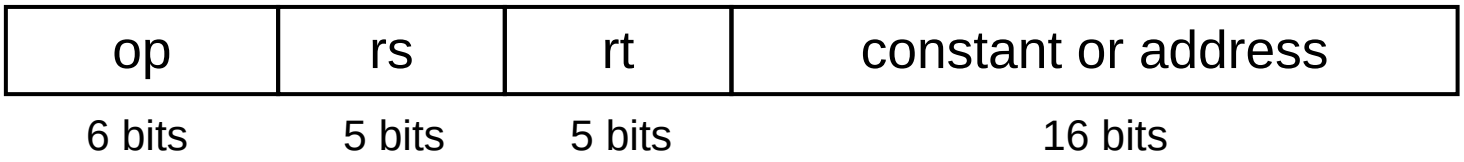
Hexadecimal

- Base 16
  - Representação compacta para binários
  - 4 bits por dígito hexadecimal

0	0000	4	0100	8	1000	c	1100
1	0001	5	0101	9	1001	d	1101
2	0010	6	0110	a	1010	e	1110
3	0011	7	0111	b	1011	f	1111

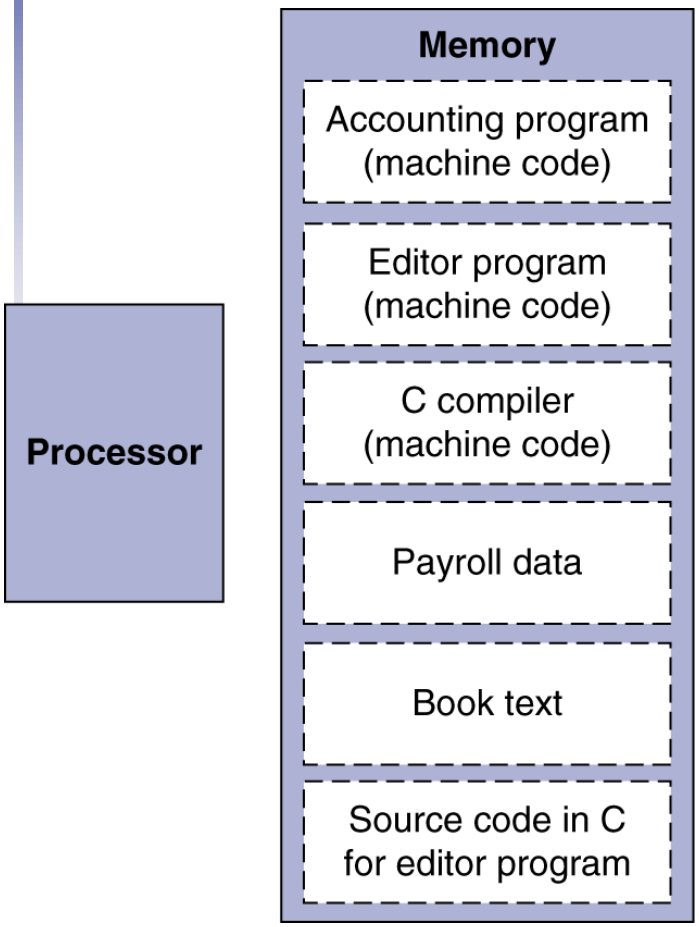
- Exemplo: eca8 6420
  - 1110 1100 1010 1000 0110 0100 0010 0000

Formato do tipo-I



- Instruções imediatas e de acesso à memória
  - rt: registrador de origem (sw) ou destino (lw)
  - constante: -2^{15} to $+2^{15} - 1$
  - endereço: offset added to base address in rs

O programa armazenado



- Instruções são representadas em binários, assim como os dados
- Instruções e dados são armazenados juntos na memória
- Programas podem operar com programas
  - e.g., compiladores, linkers, …

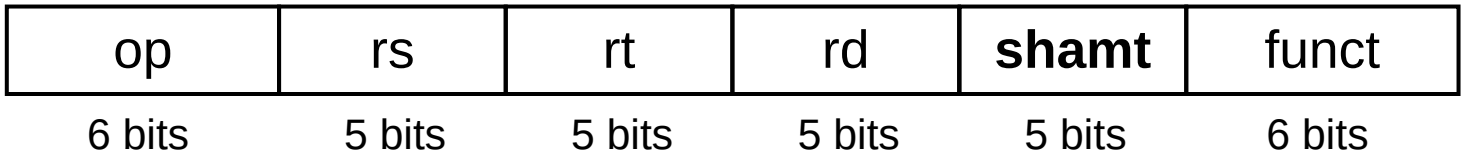
Operações lógicas

- Instruções para manipulação de bits

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

- Úteis para inserir ou extrair bits numa palavra

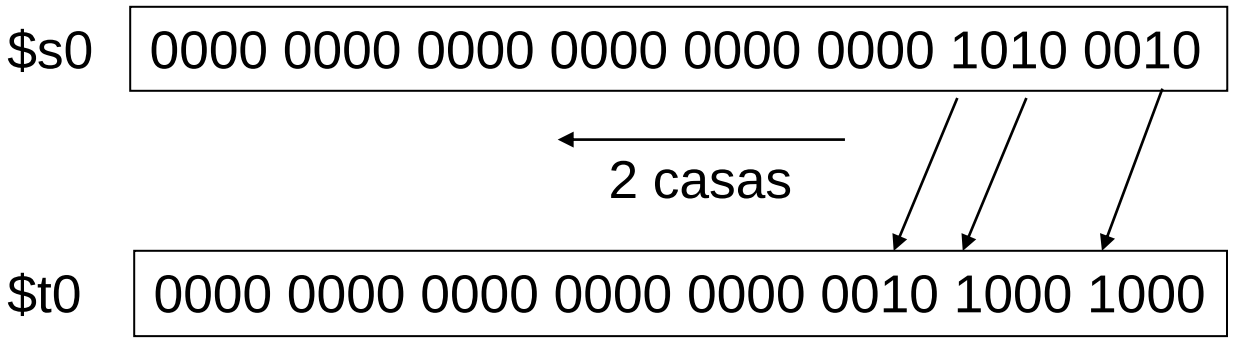
Operações de shift



- shamt: quantas posições deslocar
- Shift à esquerda (`sll` – *shift left logical*)
  - Desloca à esquerda e preenche com zero à direita
  - `sll` *i* bits multiplica por 2^i
- Shift à direita (`srl` – *shift right logical*)
  - Desloca à direita e preenche com zero à esquerda
  - `srl` *i* bits divide por 2^i (apenas sem sinal)

Operações de shift

- `sll $t0, $s0, 2`

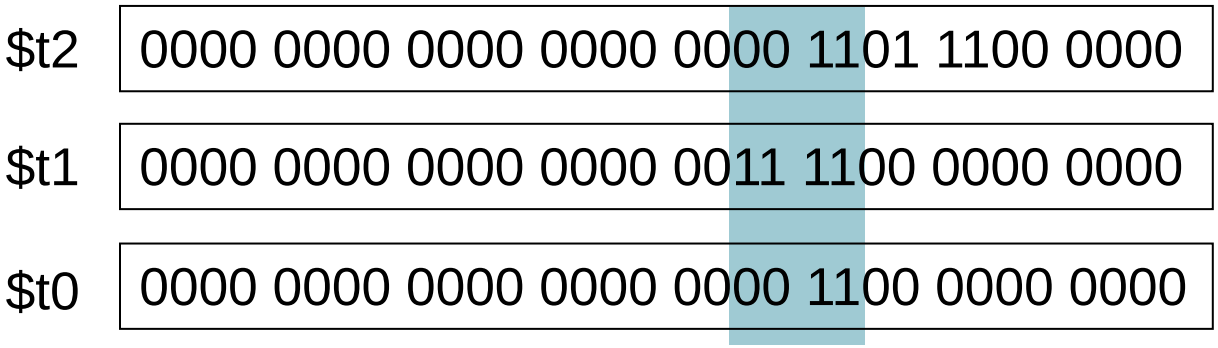


- É uma exceção ao formato do tipo R
  - Embora envolva dois registradores e uma constante (parece do tipo I), usa o campo shamt do formato tipo-R

Operações AND

- Útil para usar como máscara
  - Seleciona alguns bits, define os demais como zero

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



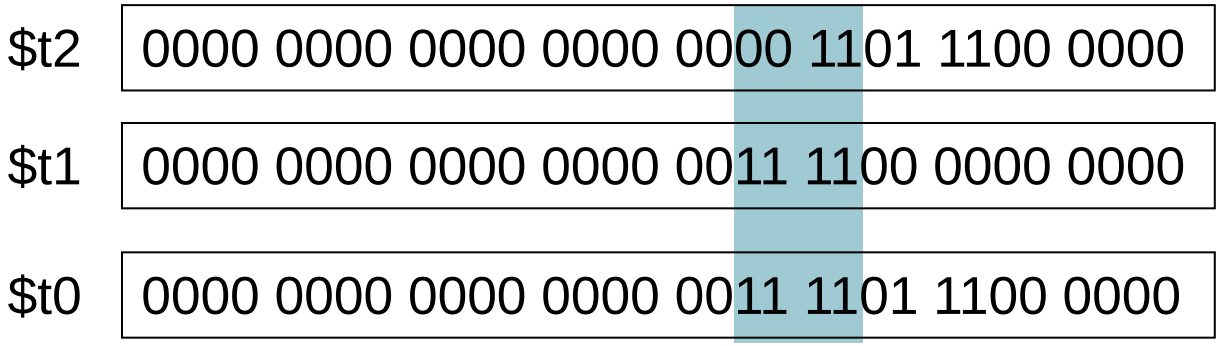

```
andi $t0, $t1, 60 (imediata)
```

- *Exemplo*: extrair o bit mais significativo de uma palavra

Operações OR

- Útil para incluir bits numa palavra
  - Define alguns bits como 1, e outros não modifica

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




```
ori $t0, $t1, 32 (imediata)
```

- *Exemplo*: definir como 1 um bit zero de uma palavra.

Operações NOT

- Útil para inverter os bits numa palavra
  - Muda 0 para 1 e 1 para 0
- MIPS possui a instrução tipo R NOR
  - a NOR b == NOT (a OR b)

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

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