Linguagem de montagem

2. Instruções de acesso à memória e representação de inteiros

Prof. John L. Gardenghi
Adaptado dos slides do livro

Acesso à memória

- A memória principal é usada para armazenar os dados que não cabem nos registradores
  - Arrays, structures, dados dinâmicos
- Para executar operações aritméticas:
  - Carrega o dado em um registrador
  - Armazena o resultado do registrador na memória
- A memória é endereçada por *bytes*
  - Cada endereço é um dado de 8-bits (1 byte)
- Palavras são alinhadas na memória
  - O endereço inicial deve ser múltiplo de 4
  - Restrição de alinhamento

Instruções de acesso à memória

- `lw reg, offset(base)`
  - `lw`: *load word,* `lh`*: load halfword,* `lb`*: load byte*
  - carrega o dado no registrador `reg` do endereço `base + offset`
- `sw reg, offset(base)`
  - `sw`: *store word,* `sh`*: store halfword,* `sb`*: store byte*
  - salva o dado do registrador `reg` no endereço `base + offset`
- *Syscall* código 9: alocação de memória
  - `$a0` recebe o tamanho em bytes
  - `$v0` retorna o endereço base
- **Atenção!**
  - `base` é um registrador
  - `offset` é um número

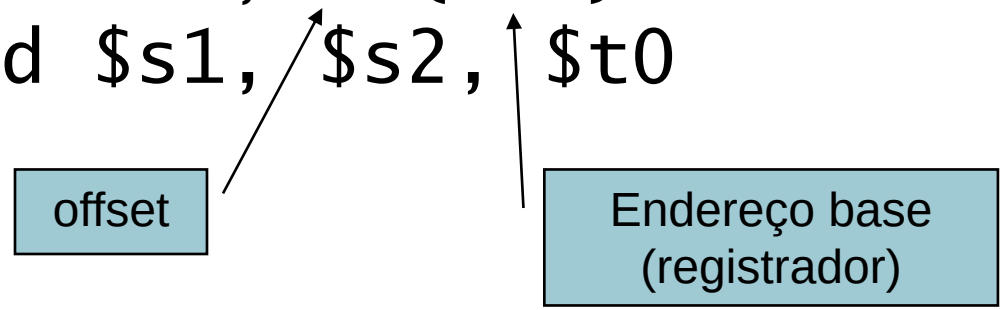
Acesso à memória – Exemplo 1

- Código C:

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

  - g em $s1, h em $s2, endereço base de A em $s3
- Código MIPS compilado:
  - Índice 8 do vetor requer um *offset* de 32 *bytes*
    - 4 bytes por palavra

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



Acesso à memória – Exemplo 2

- Código C:

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

  - h em $s2, endereço base de A em $s3

- Compiled MIPS code:
  - Índice 8 do vetor requer um *offset* de 32

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

- Atenção com o uso de `lw` e `sw`!

Registradores vs. Memória

- Registradores possuem acesso mais rápido que memória
- Operar na memória requer carregar e salvar o dado
  - Mais instruções a serem executadas
- Um compilador deve usar os registradores o máximo possível
  - A memória deve ser acessada apenas para variáveis menos utilizadas
  - Otimização de registradores é importante!

Instruções imediatas

- Dado constante especificado na instrução

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

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

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

- *Princípio de Design 3:* Torne o caso comum mais rápido
  - O uso de constantes pequenas é muito comum
  - Instruções imediatas evitam uma instrução `lw` num registrador

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 … 0010_2
  - –2 = 1111 1111 … 1101_2 + 1
    = 1111 1111 … 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