Aula 5

(25/nov)

- Instruções de acesso à memória
- Instruções imediatas
- Representação de inteiros com e sem sinal

Instruções de 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

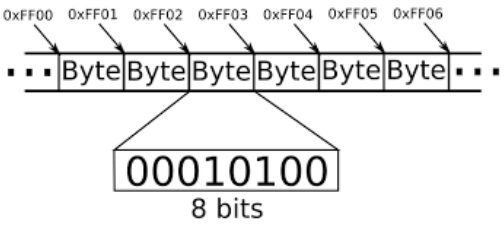
- Cada endereço é um dado de 8-bits (1 byte)

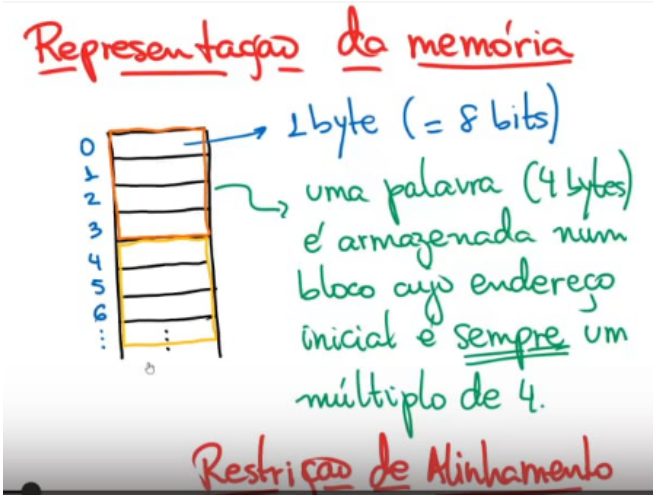
Palavras são alinhadas na memória

- O endereço inicial deve ser múltiplo de 4
- **Restrição de alinhamento** - Facilita a aritmética de endereços (4 é potência de 2)

Representação da memória

- Dado de 4 bytes = palavra
- **Restrição de alinhamento** PROVA!!!!
  - A palavra nunca vai ser armazenada sem ser um múltiplo de 4 (registrador)
  - Para não ficar espaços livres inutilizáveis (de 3 bytes, por exemplo)





Instruções de acesso a memória

1. lw reg, offset(base)
   - **lw**: load word - le na memória e salva no registrador
   - **lh**: load halfword
   - **lb**: load byte
   - carrega o dado no registrador reg do endereço base + offset
2. sw reg, offset(base)
   - **sw**: store word - le dados do registrador e salva na memória
   - **sh**: store halfword
   - **sb**: store byte
   - salva o dado do registrador reg no endereço base + offset
3. Syscall código 9: alocação de memória
   - $a0 recebe o tamanho em bytes
   - $v0 retorna o endereço base

Numa arquitetura, a quantidade de memória (de registradores) é limitado, e isso não é suficiente para lidar com programas complexos que contêm milhares de variáveis. Por isso, um computador possui unidades de memória de maior capacidade de armazenamento (como a memória RAM, o HD, etc). À memória de maior capacidade chamaremos de memória principal. A memória princiapl é vista como um grande vetor unidimensional.

Assim sendo, o MIPS possui instruções de transferência de dados que recuperam ou armazenam dados da/para a memória principal. São elas:

- \b (bad byte) e lw (load word) carregam um byte de dado e uma palavra inteira da memória principal para o resgitrador, respectivamente.
- sb (store byte) e sw (store word) salvam um byte e uma palavra do registrador para a memória, respectivamente.

Exemplo:

```
lw $t0, 20($a0) carrega uma palavra inteira do endereço de memória [$a0+20] no r$gp            |   28         |
Ponteiro Global
```

$sp | 29 | Ponteiro da pilha $fp egistrador $t0

Essas instruções utilizam um endereçamento indexado, que é feito da seguinte forma: offset(reg-base), onde

- reg-base é o registrador que contém o endereço base
- offset é o deslocamento necessário a partir do endereço base

Importante: como cada palavra possui 4 bytes (= 32bits), o endereçamento da memória é feito de 4 em 4 unidades

Memória princial:

Endereço	Dados	Posição
16		4
12		3
8		2
4		1
0		0

```
Endereço = 4 * Posição
```

A sintaxe das instruções de acesso à memória é:

```
op reg_operando, offset(reg_base)
                          |
                          V
                      endereço
```

Exemplo:

Suponhamos que A seja um vetor de 100 palavras e as variáveis g e h estejam associadas aos registradores $s1 e $s2. Suponha ainda que o endereço base de A esteja em $s3. Como compilar a instrução:

```
g = h + A[8]

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

e quanto à instruções:

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

lw $t0, 32($s3)     !=  $t0=A[8]
add $t0, $s2, $t0   !=  $t0=g + A[8]
sw $t0, 48($s3)     !=  A[12]=$t0

register int i;
```

Obs: No MIPS, endereço bases devem ser multiplos de 4. Isso é chamado restrição de alinhamento

Exercício:

Consideremos o código a seguir. Suponha que o endereço base de A está em $s3 e result, em $s4

```
int A[4] = {1, 2, 3, 4};
int result=A[0]+A[1]+A[2]+A[3]

lw $t0, 0($s3)
add $s4, $t0, $zero
lw $t0, 4($s3)
add $s4, $s4, $t0
lw $t0, 8($s3)
add $s4, $s4, $t0
lw $t0, 12($s3)
add $s4, $s4, $t0
```

Atenção!

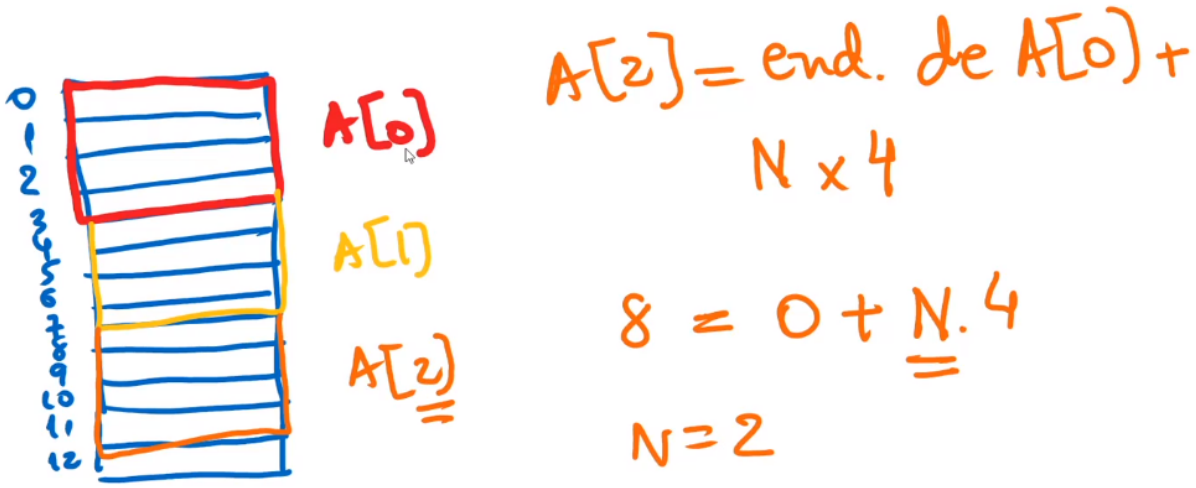
- base é um registrador
- offset é um número inteiro

Por que fazer base + offset?

1. Acesso natural para vetores

```
A[10] => endereço de A[0] + 4x10

10 = deslocamento
4 = tamanho de 1 inteiro
```



2. Se fosse o endereço direto, não seria possível representar valores muito altos (16GB)

Exemplo 1

- Códido 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 (4bytes por palabra)

lw      $t0, 32($s3)
add     $s1, $s2, $st0

#32 é o offset
#$s3 endereço base(registrador)
```

Exemplo 2

- Códido c

```
A[12] = h + A[8];
// h em $s2, endereço base de A em $s3
```

```
// quero fazer a soma e armazenar na posição 12
// sobrescreve o valor de A[12] -> sw (armazena o valor da soma)
```

- Código MIPS compilado

```
#índice 8 do vetor requer um offset de 32 bytes

lw      $t0, 32($s3)    #load word - carrega o a[8] em to
add     $t0, $s2, $t0
sw      $t0, 48($s3)    #store word (12x4)
```

Registradores x 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

i = imediato

Opera com um registrador e uma constante (ao inves de 2 registradores)

Dado constante especificado na instrução

- Substitui o segundo operando por uma constante

```
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)
```

Representação de inteiros com e sem sinal

Inteiros binários sem sinal

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

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

Varia de 0 a + 2^n – 1

- Com n=32, de 0 to +4,294,967,295
- De 0000...0000 (n vezes) até 1111....1111 (n vezes) = 0 até $2^n - 1$ (PG)

Exemplo:

$$0000:0000:0000:0000:0000:0000:0000:1011_2 = 0 + ... + 1x2^3 + 0x2^2 + 1x2^1 + 1X2^0 = 0 + ... + 8 + 0 + 2 + 1 = 11_{10}$$

Inteiros binários com sinal (complemento a 2)

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

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

Varia de -2^{n-1} a +$2^{n-1} -1$

- Com n=32, de –2,147,483,648 to +2,147,483,647

Exemplo:

$$1111:1111:1111:1111:1111:1111:1111:1100_2 = -1.2^{31} + 1.2^{30} + ... + 1.2^2 + 0.2^1 + 0.2^0 = -2,147,483,648 + 2,147,483,644 = -4_{10}$$