Organização e Arquitetura de Processadores

Assembly do MIPS

Macros/Funções, Índice/Ponteiros

Principais ferramentas: ***MARS***
Apêndice A do livro do Patterson e Hennessy

Prof. Dr. César Marcon

Macro e Função

- **Macros e funções são formas de organizar o código incluindo um grau de abstração e modularidade**

- **Macro define um trecho de código que é inserido dentro de outro trecho de código**
  - Montador realiza pré-processamentos, tais como ajustar rótulos e suportar laços
  - Código da macro é inserido completo dentro de cada referência da macro no programa

- **Função define um trecho de código que é chamado dentro de outro trecho**
  - Código da função é inserido apenas uma vez dentro do programa, independente do número de chamadas à função
  - No MIPS, o programador utiliza a instrução **jal** para chamar uma rotina endereçada por um rótulo (a função) e salvar endereço de retorno da função em **$ra**
  - A instrução **jr $ra** é usada no retorno da função para voltar para a instrução seguinte ao uso da instrução **jal**

- **Em princípio, macro executa mais rápido que função, mas consome mais área de código**

Exemplo de Macro

```
int power(int base, int exp) {
  int pow = 1;
  while(exp >= 1) { // Não trata neg.
    pow = pow * base;
    exp--;
  }
  return pow;
}

int a = 2, b = 3;
int res1, res2;
void main() {
   res1 = power(a, b);
   res2 = power(b, a);
}
```

Cuidado! Se a macro alterar um registrador, ou esta macro deve salvar o registrador, ou este o valor antigo do registrador deve ser reescrito pelo programa que chama a macro

```
.macro power (%base, %expoente)
        li $v0, 1
loop:   ble %expoente, $zero, retorno
        mul $v0, $v0, %base
        subi %expoente, %expoente, 1
        j loop
retorno:
.end_macro
.text
        .globl main
main:   lw $t0, a
        lw $t1, b
        power($t0, $t1)             # |
        sw $v0, res1                # +-> #   res1 = power(a, b);
        lw $t0, b
        lw $t1, a
        power($t0, $t1)             # |
        sw $v0, res2                # +-> #   res2 = power(b, a);
        li $v0, 10                  # | $v0=10 encerra programa
        syscall                     # +-> chamada ao SO }

.data
a: .word 2
b: .word 3
res1: .space 4
res2: .space 4
```

Exemplo de Função

```
int power(int base, int exp) {
  int pow = 1;
  while(exp >= 1) { // Não trata neg.
    pow = pow * base;
    exp--;
  }
  return pow;
}
int a = 2, b = 3;
int res1, res2;
void main() {
   res1 = power(a, b);
   res2 = power(b, a);
}
```

$a0, $a1 são argumentos da função

$v0 é o retorno da função

Problema a ser discutido:

Como fazer para modificações de registradores dentro das funções não alterarem o programa?

```
.text
        .globl main                 # |
main:   lw $a0, a
        lw $a1, b
        jal power                   # |
        sw $v0, res1                # +-> #   res1 = power(a, b);
        lw $a0, b
        lw $a1, a
        jal power                   # |
        sw $v0, res2                # +-> #   res2 = power(b, a);
        li $v0, 10                  # | $v0=10 encerra programa
        syscall                     # +-> chamada ao SO }

power:                              # int power(int base, int exp){
        li $v0, 1                   #   int pow = 1;
loop:   ble $a1, $zero, retorno     #   while(exp >= 1) {
        mul $v0, $v0, $a0           #     pow = pow * base;
        subi $a1, $a1, 1            #     exp--;
        j loop                      #   }
retorno: jr $ra                     #   return pow;
                                    # }

.data
a: .word 2
b: .word 3
res1: .space 4
res2: .space 4
```

Vetores e Matrizes

- Vetores/matrizes são estruturas básicas para agrupar dados relacionados
- Os elementos destas estruturas podem ser acessados por índices que permitem abstrair detalhes de implementação

- Exemplo de acesso à um vetor de caracteres vetChar por um índice i

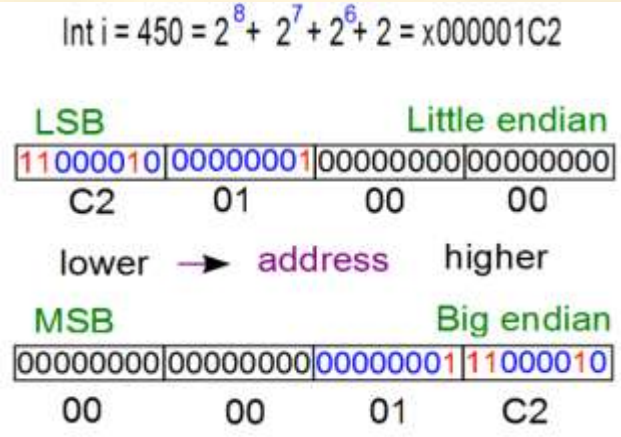
```
char vetChar[8];
char valor = 1;
int i = 0;
vetChar[i] = valor;
i++;
valor++;
vetChar[i] = valor;
```

```
.text
  la    $t0, vetChar    # $t0 = &vetChar[0];
  lb    $t1, valor      # $t1 = valor;
  li    $t2, 0          # int i = 0;
  add   $t3, $t0, $t2   # $t3 = &vetChar[i] = &vetChar + i
  sb    $t1, 0($t3)     # vetChar[i] = valor;
  addi  $t2, $t2, 1     # i++;
  addi  $t1, $t1, 1     # valor++;
  add   $t3, $t0, $t2   # $t3 = &vetChar[i] = &vetChar + i
  sb    $t1, 0($t3)     # vetChar[i] = valor;

.data
  vetChar: .space 8     # char vetChar[8]; Espaço 8 char
  valor:   .byte 1      # char valor = 1;
```

Acesso à Vetor com Índice

- **Exemplo do preenchimento de um vetor de 8 caracteres (1 byte cada caractere) com inteiros, onde cada elemento contém o índice do vetor, a começar de 1**
- ***Verificar no MARS*** **se a memória é organizada como big endian ou *little endian*!!**

```
char vetChar[8], valor = 1;

main() {
        int i = 0;
        while(i < 8) {
                vetChar[i] = valor;
                i++;
                valor++;
        }
}
```




```
.text
  .globl main             # |
main:                     # +-> main() {
  la    $t0, vetChar      # $t0 = &vetChar[0];
  lb    $t1, valor        # $t1 = valor;
  li    $t2, 0            #   int i = 0;
loop:                     # |
  bge   $t2, 8, fim       # +->   while(i < 8) {
  add   $t3, $t0, $t2     #        $t3 = &vetChar + i
  sb    $t1, 0($t3)       #        vetChar[i] = valor;
  addi  $t2, $t2, 1       #        i++;
  addi  $t1, $t1, 1       #        valor++;
  j     loop              # }
fim:                      # |
  li    $v0, 10           # | $v0=10 encerra programa
  syscall                 # +-> chamada ao SO }
.data
  vetChar: .space 8       # char vetChar[8]; Espaço 8 char
  valor:   .byte 1        # char valor = 1;
```

Operação com Vetores/Matrizes – Índices e Ponteiros

- **Acesso a estas estruturas através de índices ou ponteiros, maior ou menor abstração**
  - Algumas linguagens de alto nível suportam os dois tipos de acesso, outras apenas o acesso com índices
- **Matrizes ordenam elementos dentro de um espaço de endereçamento sequencial e contíguo**
  - Implementação com ponteiros se beneficia deste endereçamento

```
char vetChar[8];
char valor = 1;
int i = 0;
vetChar[i] = valor;
```

```
.text
  la      $t0, vetChar          # $t0 = &vetChar[0];
  lb      $t1, valor            # $t1 = valor;
  li      $t2, 0                # int i = 0;
  add     $t3, $t0, $t2         # $t3 = &vetChar[i] = &vetChar + i
  sb      $t1, 0($t3)           # vetChar[i] = valor;
.data
  vetChar: .space 8 # char vetChar[8]; Espaço 8 char
  valor:   .byte 1  # char valor = 1;
```

Índice

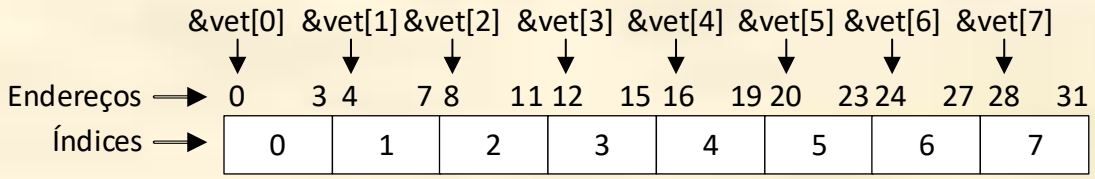
```
char vetChar[8];
char valor = 1;
char *p=&vetChar[8];
*p = valor;
```

```
.text
  la      $t0, vetChar          # $t0 = &vetChar[0]; $t0 já é o próprio *p
  lb      $t1, valor            # $t1 = valor;
  sb      $t1, 0($t0)           # *p = valor;

.data
  vetChar: .space 8 # char vetChar[8]; Espaço 8 char
  valor:   .byte 1  # char valor = 1;
```

Ponteiro

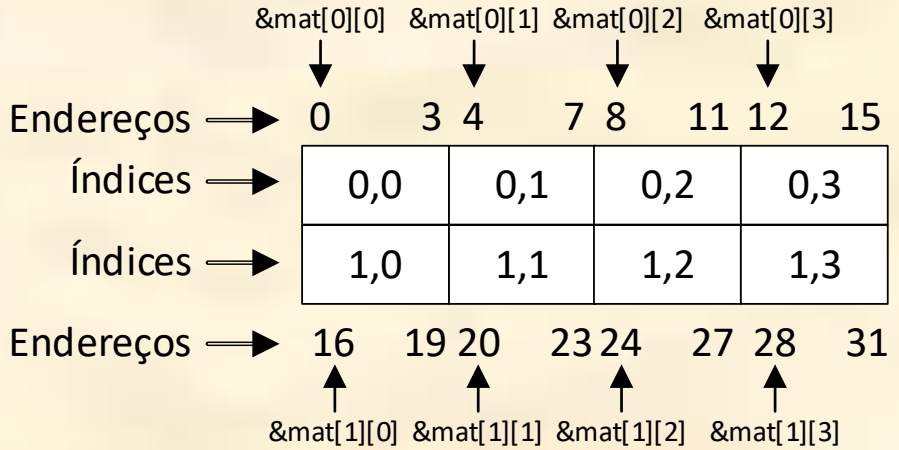
Operação com Vetores/Matrizes – Índices e Ponteiros

- **Tamanho do elemento é uma abstração da linguagem de alto nível**
  - O tamanho de cada elemento implica no incremento do ponteiro
  - O endereço do elemento subsequente é alcançado incrementando o endereço atual com o tamanho de cada elemento do vetor
  - Exemplo de um vetor de 8 inteiros → cada elemento tem 4 bytes



int vet[8];

- **Matriz é uma abstração multidimensional de um determinado dado**
  - Ao incrementar o endereço do último elemento de uma dimensão da matriz, gera o endereço do primeiro elemento desta dimensão em um índice mais alto
  - Exemplo de uma matriz de 2×4 inteiros



int mat[2][4];

Operação com Vetores/Matrizes – Índices e Ponteiros

- Implementar o programa com acesso ao vetor ***através de índices***
- ***Verificar no MARS*** para que serve a diretiva align com o valor 2? Executar sem e ver o que acontece

```
int vet[10];

main () {
  printf("Digite val:");
  int val = getchar();
  int i=0;
  while(i<10) {
    vet[i] = val;
    i++;
  }
}
```

```
.text
  .globl main
main:                           # main() {
  la    $a0, String1            # |
  li    $v0, 4                  # |
  syscall                       # +-> printf("Digite val: ");
  li    $v0, 5                  # | $v0=5 lê inteiro e armazena em $v0
  syscall                       # |
  move  $t1, $v0                # +-> val = getchar();
  la    $t6, vet                #   $t6 = &vet[0];
  move  $t0, $zero              #   i=0;
loop:                           # |
  bge   $t0, 10, fim            # +-> while(i<10) {
  sll   $t2, $t0, 2             # |  $t2 = $t1 * 4 (desloca o índice de palavra)
  add   $t3, $t2, $t6           # |  $t3 = &vet + i (calcula o endereço do elemento)
  sw    $t1, 0($t3)             # +->  vet[i] = val;
  add   $t0, $t0, 1             #   i++;
  j     loop                    # }
fim:                            # |
  li    $v0, 10                 # |
  syscall                       # +-> } chamada ao SO -> $v0=10 encerra o programa
.data
String1:  .asciiz "Digite val: "
.align 2                        # necessário, caso os dados não estejam alinhados
vet:      .space 40             # int vet[10]; Espaço para 10 words
```

Operação com Vetores/Matrizes – Índices e Ponteiros

- Implementar o mesmo programa anterior com acesso ao vetor ***através de ponteiros***
- ***Verificar no MARS*** para que serve a diretiva align com o valor 2? Executar sem e ver o que acontece

```
int vet[10];

main () {
  printf("Digite val:");
  int val = getchar();
  int *p = &vet[0];
  int *pFim = &vet[10];
  while(p < pFim) {
    *p = val;
    p++;
  }
}
```

```
.text
  .globl main
main:
  la    $a0, String1     # | main() {
  li    $v0, 4           # |
  syscall                # +-> printf("Digite val: ");
  li    $v0, 5           # | $v0=5 lê inteiro e armazena em $v0
  syscall                # | chamada ao SO
  move  $t1, $v0         # +-> val = getchar();
  la    $t6, vet         # + int *p = $t6 = &vet[0];
  la    $t5, vet+40      # + int *pFim = $t5 = &vet[10];
loop:                    # |
  bge   $t6, $t5, fim    # +-> while(p < pFim) {
  sw    $t1, 0($t6)      #   *p = val;
  add   $t6, $t6, 4      #   p++;
  j     loop             # }
fim:                     # |
  li    $v0, 10          # | $v0=10 encerra o programa
  syscall                # +-> chamada ao SO }

.data
String1:  .asciiz "Digite val: "
.align 2                 # necessário, caso os dados não estejam alinhados
vet:      .space 40      # int vet[10]; Espaço para 10 words
```

Exercício – Alto nível

- Fazer um programa assembly que reordene, no formato descrito abaixo, um vetor de 10 inteiros preenchido com números de 0 a 9
- O acesso ao vetor deve ser feito com ponteiros
- As trocas devem ser feitas através de uma função swap, que recebe como parâmetro os endereços das posições de memória que devem ser trocadas

```
void swap(int *pa, int* pb) {
	int aux = *pa;
	*pa = *pb;
	*pb = aux;
}
```

```
int vet[] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
void main() {
	int *p = &vet[0];
	int *pFim = &vet[9];
	while(p <= pFim) {
		swap(p, pFim);
		p++;
		pFim--;
	}
}
```

Resposta – Assembly

```
.text
        .globl main
main:                                   # void main() {
        la $a0, vet                     #       int *p = &vet[0];
        la $a1, vet+36                  #       int *pFim = &vet[9];
while:                                  #
        bgt $a0, $a1, fimLoop           #       while(p <= pFim) {
        jal swap                        #               swap(p, pFim);
        addi $a0, $a0, 4                #               p++;
        subi $a1, $a1, 4                #               pFim--;
        j while                         #       }
fimLoop:                                #
        li $v0, 10                      #
        syscall                         # }

swap:                                   # void swap(int *pa, int* pb) {
        lw $t0, 0($a0)                  #       int aux = *pa;
        lw $t1, 0($a1)                  #
        sw $t1, 0($a0)                  #       *pa = *pb;
        sw $t0, 0($a1)                  #       *pb = aux;
        jr $ra                          # }

.data
vet: .word 0, 1, 2, 3, 4, 5, 6, 7, 8, 9        # int vet[] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
```