

Microprocessadores

Hugo Marcondes
hugo.marcondes@ifsc.edu.br

Aula 04

Instituto Federal de Educação, Ciência e Tecnologia de Santa Catarina

Declarando constantes e variáveis - MARS

- Os dados e constantes devem ser declarados dentro do seu segmento correspondente (`.data` / `.kdata`)
- Partes de uma declaração:
 - **label**: endereço da variável
 - **diretiva**: define o tipo de dado e quantidade de memória alocada
 - **constante**: valor armazenado pela variável

```
.data
# string prompt constant
prompt: .ascii "What is your favorite number?: "
# variable to store response
favnum: .word 0
```

2 IFSC - Departamento Acadêmico de Eletrônica

Tudo que tiver ponto na frente, é um comando para o montador, de como a gente quer que o código seja montado dentro do sistema.

.Data, tá falando pro diretivo que quer que aloque tudo dentro do .Data

.ascii aloca string na memória, sem \0 no final

.asciiz aloca string na memória e bota \0 final

Declarações sequenciais

- Declarações sequenciais, serão organizadas de forma sequencial na memória

```
.data
# Quebrando textos longos em múltiplas linhas!
help: .ascii "The best tool ever. (v.1.0)\n"
      .ascii "Options:\n"
      .asciiz "--h Print this help text.\n"
# Inicialização de vetores
fibs: .word 0, 1, 1, 2, 3, 5, 8, 13, 21, 35, 55, 89, 144
```

3 IFSC - Departamento Acadêmico de Eletrônica

Declarações sequenciais

- Declarações sequenciais, serão organizadas de forma sequencial na memória

```
.data
# Quebrando textos longos em múltiplas linhas!
help: .ascii "The best tool ever. (v.1.0)\n"
      .ascii "Options:\n"
      .asciiz "--h Print this help text.\n"
# Inicialização de vetores
fibs: .word 0, 1, 1, 2, 3, 5, 8, 13, 21, 35, 55, 89, 144
      .word 0x0, 0x4, 0x8, 0x12
      label - label+4 - label+8 etc
```

4 IFSC - Departamento Acadêmico de Eletrônica

Label é um apelido associado a um endereço de memória"

se nao usar asciiz e quiser botar manualmente o \0 fica

.ascii "--h Print this help text.\n\0"

Esses 3 ascii ascii e asciiz ta tudo junto como se fosse um printf direto

o Label --> Fibs neste caso, está associado ao endereço de memória do primeiro numero neste caso "0".

Reservando espaços



- Espaço de memória pode ser reservado utilizando a diretiva `.space`
- argumento da diretiva representa o número de bytes a ser alocado.
- Uso em dados (em especial vetores) não inicializados

```
.data
# Reserva espaço para um vetor de 10 palavras (int)
array: .space 40
```

- `array` é o endereço do elemento de índice 0.
- Demais elementos acessados pelo label
 - `array+4`, `array+8` ... `array+36`

`fibs[0]+4 = fibs 0x100074 +4bytes --> elemento [1] -> 0x100078`

5 IFSC - Departamento Acadêmico de Eletrônica

label etc é usado para como int float etc, (INICIALIZADORES DO CÓDIGO)
Caso eu queira mudar o valor depois eu tenho que usar instruções.

lucas: .word 10

No endereço de memória onde está associado a label lucas tenho valor 10.

para puxar o valor do lucas pro código:

`lw(loadword) $t0, 0x74(endereço)($gp)`

`$gp` = global pointer

`la(load adress) $s0 lucas`

`lw` (acessar dentro da memória)

`la` (carregar endereço de memória da variável)

`sw` (store word) `$s0`, lucas

Leitura e escrita



- Basicamente realizada através das instruções de `lw` [Load Word] e `sw` [Store Word]

```
.text
lw    $t1, 4($t2)    # $t1 = Memory[$t2+4]
addi  $t1, $t1, 12    # $t1 = $t1 + 12
sw    $t1, 4($t2)    # Memory[$t2+4] = $t1
```

- `$t2` é o **registrador base**
- `4` é o **offset**
- Mas como inicializar o valor do registrador base ?
 - Pseudo instrução **la - Load Address**
 - Uso das instruções **lui** e **ori** para formar a palavra de 32 bits !

6 IFSC - Departamento Acadêmico de Eletrônica

Exemplo



```
.data
# Variáveis
nums: .word -7, 20, -5
result: .word 0

.text
la    $t1, nums
lw    $s1, 0($t1)
lw    $s2, 4($t1)
lw    $s3, 8($t1)
add   $s1, $s1, $s2
add   $s1, $s1, $s3
la    $t1, result
sw    $s1, 0($t1)
```

7 IFSC - Departamento Acadêmico de Eletrônica

Exemplo



```
.data
# Variáveis
nums: .word -7, 20, -5
result: .word 0

.text
lw    $s1, nums
lw    $s2, nums+4
lw    $s3, nums+8
add   $s1, $s1, $s2
add   $s1, $s1, $s3
sw    $s1, result
```

Pseudo Instrução

8 IFSC - Departamento Acadêmico de Eletrônica

- Leitura
 - **lb**: load byte (sign extend) **8bits**
 - **lh**: load halfword (sign extend) **16bits**
 - **lbu**: load byte unsigned (zero extend)
 - **lhu**: load halfword unsigned (zero extend)
- Escrita
 - **sb**: store byte (low order)
 - **sh**: store halfword (low order)
- Lembrar:
 - MARS -> Little-endian
 - Dados devem ser alinhados em seus respectivos tamanhos!

Chamada de Sistemas Syscalls

- **A interface com o mundo externo deve**
 - Ser genérica (dispositivos heterogêneos)
 - Ser simples ! (Principalmente em RISC)
- Memória e E/S compartilham o mesmo espaço de endereçamento
 - Uma faixa do endereçamento é destinado a E/S
 - Entrada: Carga (lw) deste endereços
 - Saída: Armazenamento (sw) nestes endereços

- Muitas vezes, não conhecemos (ou não desejamos conhecer) o endereçamento de E/S
- Programas de usuário não possuem permissão para utiliza-los diretamente
- **Sistema Operacional - (kernel)**
 - **Conhece os endereços e possui acesso** aos mesmos
 - **Provê serviços** para a **interação** dos programas
 - **Tais serviços** são **requisitados através** de chamadas de sistemas (**Syscall**)

Chamada de Sistema



- Interface para requisitar que SO faça alguma coisa.
- Na perspectiva do MIPS, as chamadas de sistemas funcionam da seguinte forma:
 - **syscall** - instrução para invocar o SO
 - SO verifica o registrador \$v0 para saber o que vc quer
 - SO acessa possíveis argumentos nos registradores \$a0 - \$a3
 - Rotina do SO é executada
 - SO coloca o resultado em registradores (se há retorno)
- MARS oferece um série de chamadas de sistema para interagir com o SO nativo. [F1]

13 IFSC - Departamento Acadêmico de Eletrônica

\$a0 - \$a3 --> caso queira enviar parâmetros para SO

Caso haja retorno, ele vai voltar para registrador \$v0

Principais Chamadas no MARS



Serviço	\$v0	Parâmetros	Retorno
Imprime Inteiro	1	\$a0 = valor	-
Imprime String	4	\$a0 = ptr string	-
Lê Inteiro	5		\$v0
Lê String	8	\$a0 = ptr buffer \$a1 = tamanho	
Encerra	10	-	-

Documentação completa no menu Help (F1) do Mars

14 IFSC - Departamento Acadêmico de Eletrônica

chamada de sistema

Quando eu chamar a Syscall, estou buscando em \$v0 qual a instrução que eu desejo fazer.

chamei a função exemplo 1, que é inteiro, quando eu sair da função ele volta para \$v0 o resultado que eu digitei.

Macros do Montador



- O montador possui mecanismos para permitir a construção de macros para facilitar a construção de código recorrentes.
- Para a definição de macros, as seguintes diretivas do montador são utilizadas:
 - .macro
 - .end_macro
- Macros também podem ter parâmetros que são substituídos conforme os exemplos a seguir (MARS).

15 IFSC - Departamento Acadêmico de Eletrônica

.macro exit

li \$v0, 10
syscall

.end macro

Exercício



- Parrot.asm
- Faça um programa no MARS, utilizando as chamadas de sistema que implementa um papagaio :)
- O programa simplesmente imprime no terminal a mesma frase que foi digitada.

```
#Diga alguma coisa que irei dizer também!  
#Você diz: ....  
# Eu digo: ....
```

```
#Diga alguma coisa que irei dizer também!  
.....
```

16 IFSC - Departamento Acadêmico de Eletrônica