INSTITUTO FEDERAL
SANTA CATARINA

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: .asciiz "What is your favorite number?: "

# variable to store response
favnum: .word 0
```


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
```


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
```

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

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 !

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)
```


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

Leitura e Escrita de sub-palavras

- Leitura
  - lb: load byte (sign extend)
  - lh: load halfword (sign extend)
  - 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

Entrada e Saída de Dados

- 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

O Sistema Operacional

- 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)

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

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).

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!
…..
```