

Luciano José Senger
Departamento de Informática
UEPG

1 Segmento de dados e E/S com *assembly* do MIPS

A programação em linguagem *assembly* para o MIPS apresenta algumas facilidades para o desenvolvimento de software. Um aspecto importante é a possibilidade de utilização de variáveis em memória, ao invés da utilização de operações de *load/store* utilizando diretamente endereços de memória. Outro aspecto é a utilização de serviços do sistema para realizar operações de E/S através do console. Tais aspectos são descritos nas seções a seguir.

1.1 Utilização de variáveis

A diretiva `.data` em linguagem *assembly* do MIPS permite informar ao montador o início do segmento de dados (`data segment`). É nessa porção de memória que podem ser utilizadas variáveis e seus identificadores.

O primeiro exemplo mostra algumas maneiras de utilizar o segmento de dados, através da declaração de variáveis dos tipos `.byte` e `.word`. Na seção do código de programa (`.text`), pode-se observar a utilização das variáveis `const1` e `const2`. Apesar das variáveis serem declaradas aparentemente como constantes, seus valores podem ser alterados durante a execução do programa, pois os nomes `const1` e `const2` são apenas facilidades disponibilizadas pelo montador para o desenvolvimento de código.

```
Primeiro exemplo
.data          # indica ao SPIM que
              # as próximas linhas são dados
const1: .byte 1 # const1 declarado como
              # byte com valor 1
const2: .word 4 # const2 declarado como
              # word com valor 4
array1: .byte 9, 21, 16, 18, 38
              # array1 declarado com
              # 5 elementos (bytes)
tam1 : .byte 5 # tam1 (tamanho do array1
              # em bytes)
array2: .word 206, 1543, 348, 709, 7000, 994
              # array2 declarado com
              # 6 elementos (words)
tam2 : .byte 24 # tam2 (tamanho do
              # array2 em bytes)

.text

.globl main
main:
lb $s0,const1
lw $s1,const2
```

A execução deste código no XSPIM produz a seguinte saída para o segmento de dados (os endereços de memória e as in-

formações adicionais foram suprimidas para melhorar a apresentação):

Segmento de dados			
DATA			
0x00000001	0x00000004	0x12101509	0x00000526
0x000000ce	0x00000607	0x0000015c	0x000002c5
0x00001b58	0x000003e2	0x00000018	0x00000000

Pode-se observar que o vetor `9,21,16,18,38` é representado pela palavra `0x12101509` e pelo primeiro byte da palavra `0x00000526`. Como os dados são gravados sequencialmente na memória, a variável `tam1` é armazenada nessa mesma palavra, no segundo byte. Assim, pode-se observar que o XSPIM interpreta a ordem dos bytes em memória da forma *little-endian*, de forma que as palavras em memória são preenchidas de trás para frente ¹. O último valor do `array1` é armazenado em outra posição de memória e após ele há um `byte` contendo o tamanho do `array`. O `array2` utiliza toda a posição de memória (`word`) para cada elemento.

1.1.1 Verifique você mesmo

Modifique o programa de forma que a variável `const2` armazene o valor igual a `-4`, ao invés de `4`, carregue o novo programa no XPSIM e verifique a representação de valores negativos na memória, utilizando complemento de 2.

A linguagem *assembly* do MIPS fornece outros tipos além de `.word` e `.byte`. Exemplos são os tipos `.double` para armazenar valores de ponto flutuante com precisão dupla, o `.float` para precisão simples, o `.half` para armazenar meias-palavras (16 bits) e o tipo `.asciiz` que armazena `strings`.

1.2 Serviços do sistema

O XSPIM implementa um lista de serviços que servem para facilitar principalmente na interface com o usuário por meio de um console. Para utilizar um desses serviços, basta colocar em `$v0` o código do serviço, definir os parâmetros (se houverem) e em seguida utilizar a instrução `syscall`.

Tal forma de utilização de serviços, com pequenas diferenças, é utilizada também em outras arquiteturas para a comunicação com o código do sistema operacional ou de uma seção de programas residentes em memórias não voláteis (p.e. BIOS). Os serviços implementados pelo XSPIM estão descritos em sua documentação [1] e na Tabela 1.

A seguir é apresentado um exemplo de programa em *assembly* do MIPS para o cálculo de médias de notas.

¹O processador MIPS pode operar tanto em *big-endian* como em *little-endian*; o XSPIM/SPIM utiliza o padrão da máquina que o simulador está sendo executado. Dessa forma, o XSPIM/SPIM utiliza a forma de ordenação *little-endian* no Intel 80x86 e *big-endian* no Macintosh.

Tabela 1: Serviços implementados pelo XPSIM

Serviço	Cód.	Parâmetros	Resultados
print_int	1	\$a0 = integer	
print_float	2	\$f12 = float	
print_double	3	\$f12 = double	
print_string	4	\$a0 = string	
read_int	5		integer (em \$v0)
read_float	6		float (em \$f0)
read_double	7		double (em \$f0)
read_string	8		\$a0 = buffer, \$a1 = length
sbrk	9	\$a0 = quantidade, endereço (em \$v0)	
exit	10		Encerra a execução do programa

Segundo exemplo

```
.data
msg1: .asciiz "\nEntre o numero de avaliações:"
msg2: .asciiz "\nEntre um valor para a nota "
msg3: .asciiz ": "
msg4: .asciiz "\nA média das notas é: "
.text
.globl main
main:
    add $t0, $zero, $zero
    add $t1, $zero, $zero
numnotas:
    li $v0, 4
    la $a0, msg1
    syscall
    li $v0, 5
    syscall
    add $s0, $v0, $zero
loopnotas:
    addi $t0, $t0, 1
    li $v0, 4
    la $a0, msg2
    syscall
    li $v0, 1
    add $a0, $zero, $t0
    syscall
    li $v0, 4
    la $a0, msg3
    syscall
    li $v0, 5
    syscall
    add $t1, $t1, $v0
    bne $t0, $s0, loopnotas
Calcula:
    div $t1, $s0
    mflo $t2
    li $v0, 4
    la $a0, msg4
    syscall
    li $v0, 1
    add $a0, $zero, $t2
    syscall
    li $v0, 5
    syscall
```

2 Exercícios

- Para o programa abaixo, identifique quais ações estão sendo realizadas, através da simulação da execução no simulador XSPIM

Primeiro exercício

```
.data
const1: .byte 1
const2: .word 4
array1: .byte 9, 21, 16, 4, 38
tam1 : .byte 5
array2: .word 206, 1543, 348, 709, 7000, 994
tam2 : .byte 24

.text
.globl main

main:
    lb $s0, const1
    lw $s1, const2
    add $s2, $zero, $zero
    add $t0, $zero, $zero
    lb $t1, tam1

soma1:
    lb $t2, array1($t0)
    add $s2, $s2, $t2
    add $t0, $t0, $s0
    bne $t0, $t1, soma1
    add $s3, $zero, $zero
    add $t0, $zero, $zero
    lb $t1, tam2

soma2:
    lw $t2, array2($t0)
    add $s3, $s3, $t2
    add $t0, $t0, $s1
    bne $t0, $t1, soma2
```

Referências

- [1] John L. Hennessy and David A. Patterson. *Computer Organization and Design: The Hardware/Software Interface, 2nd edition*. Morgan Kaufmann, 1994.

Anotações