Exemplo 3 - Outra forma de trabalhar com a memória

Existe uma outra maneira de se carregar dados em memória sem a utilização de instruções store. Ela é muito útil principalmente para a definição de valores ditos constantes (na verdade eles podem ser alterados futuramente já que temos acesso a sua posição de memória) e em inicialização de dados.

Para se armazenar os dados em memória, basta colocar a diretiva *.data*, que indica ao SPIM que os dados que a seguem devem ser carregados no segmento de dados.

```
.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)
```

Realizaremos algumas operações simples com estes dados para ver como é feito o acesso a eles.

```
.text
.globl main
main:
lb $s0,const1                           # $s0 recebe o valor de const1 (1)
lw $s1,const2                           # $s1 recebe o valor de const2 (4)
```

Observe que o label do dado declarado funciona como um ponteiro de memória. Assim, para acessá-lo basta fazer a instrução de load, passando-o como parâmetro. Com os dois arrays vamos fazer algo para acessar todos os seus elementos, como por exemplo somar todos colocando o resultado em um registrador.

```
add $s2,$zero,$zero                     # Zera o registrador $s2
add $t0,$zero,$zero                     # Zera o registrador $t0
lb $t1,tam1                             # $t1 recebe o tamanho do array1 (5bytes)
soma1:
lb $t2,array1($t0)                      # $t2 recebe o valor da posicao apontada por array1
                                        + $t0
add $s2,$s2,$t2                         # $s2 é somado com o valor carregado do array1
add $t0,$t0,$s0                         # $t0 irá apontar para a próxima posição (atual +1)
bne $t0,$t1,soma1                       # Repete até chegar ao final do array1
```

Para o segundo array basta trocar a instrução de carga de load byte para load word:

```
add $s3,$zero,$zero                     # Zera o registrador $s3
add $t0,$zero,$zero                     # Zera o registrador $t0
lb $t1,tam2                             # $t1 recebe o tamanho do array2 (24bytes)
soma2:
lw $t2,array2($t0)                      # $t2 recebe o valor da posicao apontada por array2
                                        + $t0
add $s3,$s3,$t2                         # $s3 é somado com o valor carregado do array2
add $t0,$t0,$s1                         # $t0 irá apontar para a próxima posição do array
```

```
                                    (atual +4)
bne $t0,$t1,soma2                   # Repete até chegar ao final do array2
```

Salve o arquivo com o nome de "*exercicio3.s*". Abra a Janela do segmento de dados. Observe que os dados definidos já estão carregados em memória. É interessante observar como esses dados são armazenados:

```
      DATA
[0x10000000]...[0x1000fffc]   0x00000000
[0x1000fffc]                    0x00000000
[0x10010000]                    0x00000001  0x00000004  0x04101509  0x00000526
[0x10010010]                    0x000000ce  0x00000607  0x0000015c  0x000002c5
[0x10010020]                    0x00001b58  0x000003e2  0x00000018  0x00000000
[0x10010030]...[0x10040000]   0x00000000
```

A primeira posição armazenada (10011000h) foi ocupada apenas por um byte (valor 1). O próximo valor era uma word (de valor 4). Em seguida vem o primeiro array (5 elementos - bytes). Veja que o SPIM opera a memória em *little-endian*, ou seja, a posição de memória vai sendo ocupada de trás para frente. O último valor do array1 é armazenado em outra posição de memória e após ele há um byte contendo o tamanho do array. Segue-se com o array2, mas desta vez utilizando toda a posição de memória (word) para cada elemento.

Execute o código. Na tela de registradores você vai ver os valores carregados em $s0, $s1, $s2, e $s3 - 1, 4, 88 (soma dos valores do array1) e 10800 (soma dos valores do array2).

Observação: Os dados carregados em memória através da diretiva .data são armazenados a partir da posição 10011000h. Tome cuidado quando estiver usando este tipo de carga em memória para não alcançar esta posição pelo programa, ou seja, não use valores maiores que $gp + 36864. Se necessitar de mais memória verifique até que posição os dados estáticos foram carregados e utilize um valor maior que esse.

Observação: Existem outros tipos além de .word e .byte. Entre eles podemos citar o tipo .double para armazenar valores de ponto flutuante com precisão dupla, o .float para precisão simples, o .half para armazenar meia-palavra (16bits) e outro que vamos fazer muito uso principalmente quando chegarmos à escrita de dados no console do SPIM: o .asciiz que armazena strings.

Curiosidade: O processador MIPS pode operar tanto em big-endian como em little-endian. O SPIM por sua vez, utiliza o padrão da máquina que o simulador está sendo executado. Assim, o SPIM é little-endian no Intel 80x86 e big-endian no Macintosh.