

## Exercício 02

### Objetivo:

Aprender a utilizar a interface do RARS executando o segundo exemplo de programação na linguagem de montagem do RISC-V no livro texto.

### Instruções:

1. Inicie o RARS
2. No editor de textos do RARS, transcreva o código abaixo e salve o arquivo com o nome **exercicio\_02**.

```
#####
# Exercício 02 - Patterson pags. 54/55/56
# Mostra a compilação de um comando de atribuição em C usando Array
#####
# Trecho em C:
#
# A[12] = h + A[8]

.data    # segmento de dados

# definição do array A. Coloca os valores de A[0]=0 até A[15]=150 na memória
Array_A: .word 0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, 120, 130, 140, 150

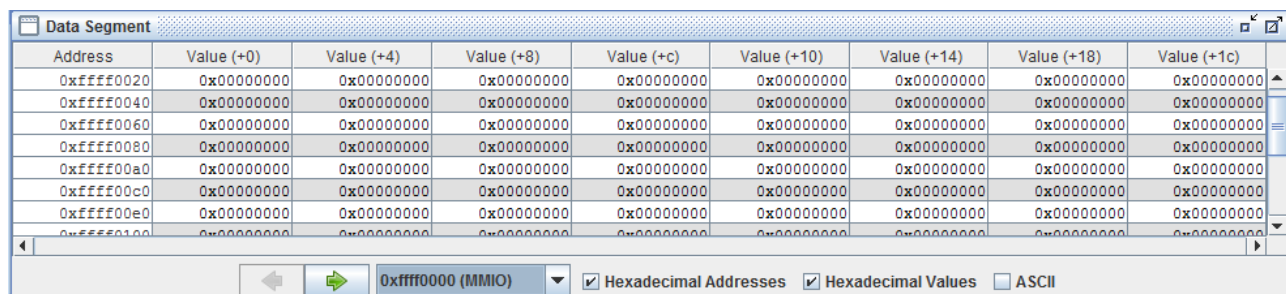
.text    # segmento de código (programa)
main:

    addi s2, zero, 1    # Inicializa s2 em 1
    la    s3, Array_A    # como o exercício assume que o endereço-base de A[]
                        # está em s3, foi incluída esta instrução

    lw     t0, 32(s3)    # t0 = A[8]
    add    t0, s2, t0    # t0 = t0 + h
    sw     t0, 48(s3)    # A[12] = t0
```

3. Para iniciar a montagem do código vá ao menu **Run** e selecione a opção **Assemble** ou pressione **F3**.
4. Na janela de segmento de dados se pode optar por mostrar, por exemplo, o conteúdo da região de memória que armazena os dados do programa (0x10010000 (.data)), da pilha (current sp) e do sistema operacional (0xffff0000 (.MMIO)), conforme é mostrado abaixo.

### Área de dados do sistema operacional



Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0xffff0020	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0xffff0040	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0xffff0060	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0xffff0080	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0xffff00a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0xffff00c0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0xffff00e0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0xffff0100	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

### Área de dados do programa do usuário

Data Segment								
Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0x00000000	0x00000000	0x0000000a	0x00000000	0x00000014	0x00000000	0x0000001e	0x00000000
0x10010020	0x00000028	0x00000000	0x00000032	0x00000000	0x0000003c	0x00000000	0x00000046	0x00000000
0x10010040	0x00000050	0x00000000	0x0000005a	0x00000000	0x00000064	0x00000000	0x0000006e	0x00000000
0x10010060	0x00000078	0x00000000	0x00000082	0x00000000	0x0000008c	0x00000000	0x00000096	0x00000000
0x10010080	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100c0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100e0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

0x10010000 (.data) ☒ Hexadecimal Addresses ☒ Hexadecimal Values ☐ ASCII

## Área de dados da pilha

Data Segment								
Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x7fffffe0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x7fffff00	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x7fffff20	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x7fffff40	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x7fffff60	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x7fffff80	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x7fffffa0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x7fffffc0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

current sp ☒ Hexadecimal Addresses ☒ Hexadecimal Values ☐ ASCII

5. Na seção DATA, os elementos do vetor são armazenados em células de memória organizadas de forma matricial, sendo que o endereço inicial do vetor é igual a **0x10010000** conforme a tabela a seguir:

Endereço da linha	Deslocamento							
	(+0)	(+4)	(+8)	(+c)	(+10)	(+14)	(+18)	(+1C)
0x10010000	Array_A[0]	Array_A[1]	Array_A[2]	Array_A[3]	Array_A[4]	Array_A[5]	Array_A[6]	Array_A[7]
0x10010020	Array_A[8]	Array_A[9]	Array_A[10]	Array_A[11]	Array_A[12]	Array_A[13]	Array_A[14]	Array_A[15]
0x10010040								
0x10010060								

O endereço de cada elemento na matriz é dado pelo endereço da linha somado ao deslocamento associado à coluna. Por exemplo, o endereço do elemento 11 é dado por  $0x10010020 + 0xC = 0x1001002C$ .

Lembre que, se o cálculo for feito em relação ao endereço base do vetor, deve-se fazer:  
 $0x10010000 + 4 \times 11 = 0x10010000 + 44 = 0x10010000 + 0x2C = 0x1001002C$ .

(end. base em hexa)+(4 x posArray dec.) = (end. base em hexa) + ( desl. decimal) = **(end. base hexa) + (desl. hexa)** = (end. desejado).

6. Observe que os valores armazenados no vetor são expressos em hexadecimal, embora no código eles sejam definidos em decimal. Então, por exemplo, o elemento `Array_A[8] = 0x00000050 = 80` (em decimal).
7. Conforme especificado no código do programa, o que se espera, após a sua execução, é que o elemento `Array_A[12]` receba a soma do conteúdo do elemento `Array_A[8]` com o conteúdo do registrador `s2`. Se este registrador for igual a 0, então, após a execução, `Array_A[12]` será igual a `Array_A[8]`.
8. Inicie a execução passo-a-passo, pressionando **F7** até chegar à segunda instrução do programa (endereço `0x00400004`).
9. Abaixo, observe, na quarta coluna, que a instrução a ser executada é **`auipc 19, 0xfc10 [Array_A]`**, mas que originalmente foi especificada como **`la s3, Array_A`**. A instrução **`la`** (load array) é na verdade uma pseudo-instrução que o montador traduz para uma sequência das instruções **`auipc`** (add upper immediate to pc) e **`addi`** (add immediate) do RISC-V (pseudo-instruções tornam a programação mais facilitada). Essa sequência permite carregar o endereço-base do vetor (`Array_A`) para um registrador base (`s3`) para que se possa acessar qualquer elemento do vetor por meio de deslocamentos em relação ao endereço-base.

Text Segment				
Bkpt	Address	Code	Basic	Source
<input type="checkbox"/>	0x00400000	0x00100913	<code>addi x18,x0,1</code>	17: <code>addi s2, zero, 1</code> # Inicializa s2 ...
<input type="checkbox"/>	0x00400004	0x0fc10997	<code>auipc x19,0x0000fc10</code>	18: <code>la s3, Array_A</code> # como o exerci...
<input type="checkbox"/>	0x00400008	0xffc98993	<code>addi x19,x19,0xffffffffc</code>	
<input type="checkbox"/>	0x0040000c	0x0409b283	<code>ld x5,0x00000040(x19)</code>	20: <code>ld t0, 64(s3)</code> # t0 = A[8]
<input type="checkbox"/>	0x00400010	0x005902b3	<code>add x5,x18,x5</code>	21: <code>add t0, s2, t0</code> # t0 = t0 + h
<input type="checkbox"/>	0x00400014	0x0659b023	<code>sd x5,0x00000060(x19)</code>	22: <code>sd t0, 96(s3)</code> # A[12] = t0

10. Faça a execução passo-a-passo do programa e, a cada instrução, preencha a tabela abaixo cada vez que o valor de um registrador ou posição da memória de dados for modificado.

Antes da execução da instrução		Depois da execução da instrução							
		Registradores			Segmento de Dados				
PC	Instrução	R8 (\$t0)	R18 (\$s2)	R19 (\$s3)	10010000 Array_A[0] 1a linha coluna (+0)	...	10010040 Array_A[8] 3a linha coluna (+0)	...	10010060 Array_A[12] 4a linha coluna (+0) ..
		00000000	00000000	00000000	00000000		00000050		00000078
00400000	addi s2, zero, 1		00000001						
00400004	auipc 19, Array_A								
00400008	addi 19,19, 0xffffffffc			1001000					

**Obs:** A segunda (**auipc**) e a terceira (**addi**) instruções apresentadas acima são inseridas pelo montador em substituição à pseudo-instrução **la s3, Array\_A** utilizada originalmente no código fonte. Essa pseudo-instrução tem a função de carregar o endereço base do vetor (**Array\_A**) para o registrador destino (**s3**). A instrução **auipc** primeiramente carrega o endereço do vetor para o registrador **19** e a segunda instrução soma o conteúdo de **19** para **s3**.

NOTA: Se for necessário reiniciar o programa, faça: **Run > Reset**