Atividades - Assembly RISC-V

- Autor: Rafael Grossi

Atividade 1

```
# string to int :)
# int LeNumero()
# Returns: a0 (integer read)
LeNumero:
    addi sp, sp, -8
    sw ra, 4(sp)
    sw s0, 0(sp)

    # Initialize accumulator = 0
    li s0, 0

    li a0, 0x130
    ecall

LeNumero_poll:
    li a0, 0x131
    ecall

    beq a0, zero, LeNumero_fim

    li t0, 1
    beq a0, t0, LeNumero_poll

    # Check for \n (10) or \r (13)
    li t1, 10
    beq a1, t1, LeNumero_fim
    li t1, 13
    beq a1, t1, LeNumero_fim

    # '0' => 48 ... '9' => 57
    li t2, 48
    blt a1, t2, LeNumero_poll
    li t2, 57
    bgt a1, t2, LeNumero_poll
    addi a1, a1, -48      # Convert ASCII to Int

    # make the number go to the correct place (sum * 10 + digit)
    li t2, 10
    mul s0, s0, t2
    add s0, s0, a1

    j LeNumero_poll

LeNumero_fim:
    # return to a0
    mv a0, s0
    lw s0, 0(sp)
    lw ra, 4(sp)
    addi sp, sp, 8
    ret
```

Atividade 2

```
# void LeVetor(int *v, int N)
# a0 = vector address, a1 = N
LeVetor:
    addi sp, sp, -16
    sw s0, 12(sp)   # Vector pointer
    sw s1, 8(sp)    # Counter
    sw s2, 4(sp)    # N (limit)
    sw ra, 0(sp)

    mv s0, a0
    li s1, 0
    mv s2, a1

LeVetor_loop:
    beq s1, s2, LeVetor_end

    call LeNumero

    # Store in vector
    sw a0, 0(s0)

    addi s0, s0, 4
    addi s1, s1, 1
    j LeVetor_loop

LeVetor_end:
    lw ra, 0(sp)
    lw s2, 4(sp)
    lw s1, 8(sp)
    lw s0, 12(sp)
    addi sp, sp, 16
    ret
```

Atividade 3

```
# int Media(int *v, int N)
# a0 = vector address, a1 = N
# Returns: a0 (average)
Media:
    mv t0, a0       # t0 = vector ptr
    mv t1, a1       # t1 = N
    li t2, 0        # t2 = sum
    li t3, 0        # t3 = iterator

Media_loop:
    beq t3, t1, Media_calc
    lw t4, 0(t0)
    add t2, t2, t4
    addi t0, t0, 4
    addi t3, t3, 1
    j Media_loop

Media_calc:
    beq t1, zero, Media_zero

    # Average = sum / N
    div a0, t2, t1
    ret
Media_zero:
    li a0, 0
    ret
```

Atividade 4

```
.data
msg_n: .string "Digite N (tamanho do vetor): "
msg_v: .string "Digite um numero: "
msg_res: .string "\nQuantidade num maiores que a media: "
msg_med: .string "Media = "

.text
```

```
main:
    # 1. Ask for N
    li a0, 4
    la a1, msg_n
    ecall

    # Read N
    call LeNumero
    mv s0, a0

    # Size = N * 4 bytes
    slli t0, s0, 2
    sub sp, sp, t0
    mv s1, sp

    mv a0, s1
    mv a1, s0
    call LeVetor

    mv a0, s1
    mv a1, s0
    call Media
    mv s2, a0       # s2 = Average

    li a0, 4
    la a1, msg_med
    ecall

    mv a1, s2
    li a0, 1
    ecall

    # 5. Count numbers > Average
    mv a0, s1
    mv a1, s0
    mv a2, s2
    call CountBiggerAvg
    mv s3, a0       # s3 = Count

    # 6. Print Result
    li a0, 4
    la a1, msg_res
    ecall

    li a0, 1
    mv a1, s3
    ecall

    # Newline
    li a0, 11
    li a1, 13
    ecall

    # 7. Deallocate stack and Exit
    slli t0, s0, 2
    add sp, sp, t0

    li a0, 10
    ecall


# int CountBiggerAvg(int *v, int N, int media)
CountBiggerAvg:
    mv t0, a0       # ptr
    mv t1, a1       # N
    mv t2, a2       # media
    li t3, 0        # count
    li t4, 0        # i

Count_loop:
    beq t4, t1, Count_end
    lw t5, 0(t0)

    # if val (t5) <= media (t2), skip increment
    ble t5, t2, Count_next
    addi t3, t3, 1

Count_next:
    addi t0, t0, 4
    addi t4, t4, 1
    j Count_loop

Count_end:
    mv a0, t3
    ret

# Ex1 -> Ex3

LeNumero:
    addi sp, sp, -8
    sw ra, 4(sp)
    sw s0, 0(sp)
    li s0, 0
    li a0, 0x130
    ecall
LeNumero_poll:
    li a0, 0x131
    ecall
    beq a0, zero, LeNumero_fim # Break on EOF/Empty
    li t0, 1
    beq a0, t0, LeNumero_poll
    li t1, 10
    beq a1, t1, LeNumero_fim   # Break on \n
    li t1, 13
    beq a1, t1, LeNumero_fim   # Break on \r
    li t2, 48
    blt a1, t2, LeNumero_poll  # Ignore < '0'
    li t2, 57
    bgt a1, t2, LeNumero_poll  # Ignore > '9'
    addi a1, a1, -48
    li t2, 10
    mul s0, s0, t2
    add s0, s0, a1
    j LeNumero_poll
LeNumero_fim:
    mv a0, s0
    lw s0, 0(sp)
    lw ra, 4(sp)
    addi sp, sp, 8
    ret

LeVetor:
    addi sp, sp, -16
    sw s0, 12(sp)
    sw s1, 8(sp)
    sw s2, 4(sp)
    sw ra, 0(sp)
    mv s0, a0
    li s1, 0
    mv s2, a1
LeVetor_loop:
    beq s1, s2, LeVetor_end
    li a0, 4
    la a1, msg_v
    ecall

    call LeNumero
    sw a0, 0(s0)
    addi s0, s0, 4
    addi s1, s1, 1
    j LeVetor_loop
LeVetor_end:
    lw ra, 0(sp)
    lw s2, 4(sp)
    lw s1, 8(sp)
    lw s0, 12(sp)
    addi sp, sp, 16
    ret

Media:
    mv t0, a0
    mv t1, a1
    li t2, 0
    li t3, 0
Media_loop:
    beq t3, t1, Media_calc
    lw t4, 0(t0)
    add t2, t2, t4
    addi t0, t0, 4
    addi t3, t3, 1
    j Media_loop
Media_calc:
    beq t1, zero, Media_zero
    div a0, t2, t1
    ret
Media_zero:
    li a0, 0
    ret
```

Atividade 5

O problema do código fornecido no enunciado está na função **LeVetor**.

A função `LeVetor` salva os registradores no topo da pilha (`sp`). No entanto, ao chamar a função `LeString`, ela passa o próprio registrador `sp` (`mv a0, sp`) como endereço do buffer para armazenar a string lida.

Como a função `LeString` começa a escrever a partir do endereço passado (`0(sp)`), qualquer entrada do usuário que ocupe bytes suficientes irá sobrescrever os valores salvos na pilha. Isso corrompe o programa quando `LeVetor` tenta retornar (Buffer Overflow).

Atividade 6

```
LeVetor:
    addi  sp, sp, -56

    # Save at top
    sw    s2, 52(sp)
    sw    s1, 48(sp)
    sw    s0, 44(sp)
    sw    ra, 40(sp)

    mv    s0, a0
    mv    s1, a1

l2:
    blt   s1, zero, fiml2
    li    a0, 4
    la    a1, mensagem1
    ecall

    # Pass buffer address
    mv    a0, sp
    call  LeString

    # Convert string at buffer to number
    mv    a0, sp
    call  ConverteNumero

    sw    a0, 0(s0)
    addi  s0, s0, 4
    addi  s1, s1, -1
    j     l2

fiml2:
    # Restore registers from the top of the frame
    lw    ra, 40(sp)
    lw    s0, 44(sp)
    lw    s1, 48(sp)
    lw    s2, 52(sp)
    addi  sp, sp, 56
    ret
```

Atividade 7

```
.data
buffer_temp: .space 100
msg_input: .string "Digite uma string: "
```

```
msg_out: .string "String alocada no Heap: "

.text
main:
    li a0, 4
    la a1, msg_input
    ecall

    # newline
    li a0, 11
    li a1, 13
    ecall

    call LeString
    mv s0, a0

    li a0, 4
    la a1, msg_out
    ecall

    # Print string from heap
    li a0, 4
    mv a1, s0
    ecall

    # newline
    li a0, 11
    li a1, 13
    ecall

    li a0, 10
    ecall

# char * LeString()
# Allocates exact memory on Heap and returns pointer
LeString:
    addi sp, sp, -16
    sw ra, 12(sp)
    sw s0, 8(sp)

    # Read into temp
    la a0, buffer_temp
    li a1, 100       # Max size
    call fgets

    la a0, buffer_temp
    call strlen
    mv s1, a0        # s1 = length

    # proper strlen -> len + \0
    addi a0, s1, 1
    li a7, 9
    ecall
    mv s0, a0        # s0 = new heap address

    # copy to heap
    mv a0, s0
    la a1, buffer_temp
    call strcpy

    # Return new address
    mv a0, s0

    lw s0, 8(sp)
    lw ra, 12(sp)
    addi sp, sp, 16
    ret

# From last week exercise.
fgets:
    mv t0, a0
    addi t3, a1, -1
    li t4, 0
    li a0, 0x130
    ecall
fgets_poll:
    bge t4, t3, fgets_end
    li a0, 0x131
    ecall
    li t1, 2
    beq a0, t1, fgets_read
    beq a0, zero, fgets_end
    j fgets_poll
fgets_read:
    li t2, 10
    beq a1, t2, fgets_end
    sb a1, 0(t0)
    addi t0, t0, 1
    addi t4, t4, 1
    j fgets_poll
fgets_end:
    sb zero, 0(t0)
    ret

# int strlen(char *s)
strlen:
    li t0, 0
strlen_loop:
    add t1, a0, t0
    lbu t2, 0(t1)
    beq t2, zero, strlen_end
    addi t0, t0, 1
    j strlen_loop
strlen_end:
    mv a0, t0
    ret

# char * strcpy(char *dest, char *src)
strcpy:
    mv t0, a0 # dest
    mv t1, a1 # src
strcpy_loop:
    lbu t2, 0(t1)
    sb t2, 0(t0)
    beq t2, zero, strcpy_done
    addi t0, t0, 1
    addi t1, t1, 1
    j strcpy_loop
strcpy_done:
    ret
```

Atividade 8

```
# Pessoa* LePessoa()
# Allocates a Person struct and allocates memory for the name.
LePessoa:
    addi sp, sp, -16
    sw ra, 12(sp)
    sw s0, 8(sp)

    # struct (12 bytes)
    # [0: name*] [4: age] [8: next*]

    # FIX: ID goes to a0, Argument goes to a1
    li a0, 9      # sbrk ID
    li a1, 12     # Size (12 bytes)
    ecall
    mv s0, a0     # s0 = struct address

    # 2. Read Name
    li a0, 4
    la a1, msg_nome
    ecall

    # newline
    li a0, 11
    li a1, 13
    ecall

    # read string and store in struct
    call LeString
    sw a0, 0(s0)

    # read age
    li a0, 4
    la a1, msg_idade
    ecall

    # store age
    call LeNumero
    sw a0, 4(s0)

    # pointer start as null
    sw zero, 8(s0)

    # return struct address
    mv a0, s0

    lw s0, 8(sp)
    lw ra, 12(sp)
    addi sp, sp, 16
    ret
```

Atividade 9

```
# void ImprimePessoa(Pessoa *p)
ImprimePessoa:
    mv t0, a0

    # print name
    li a0, 4
    lw a1, 0(t0)
    ecall

    # Print Space
    li a0, 11
    li a1, 32
    ecall

    # print age
    li a0, 1
    lw a1, 4(t0)
    ecall

    # newline
    li a0, 11
    li a1, 13
    ecall
    ret
```

Atividade 10

```
.data
buffer_temp: .space 100
msg_nome: .string "Digite o nome: "
msg_idade: .string "Digite a idade: "
msg_res: .string "\nDados da Pessoa (Struct com nome dinamico):\n"

.text
main:
    # read person struct
    call LePessoa
    mv s0, a0        # s0 = struct pointer

    # print msg
    li a0, 4
    la a1, msg_res
    ecall

    # print person
    mv a0, s0
    call ImprimePessoa

    li a0, 10
    ecall

# Pessoa* LePessoa()
```

```
LePessoa:
    addi sp, sp, -16
    sw ra, 12(sp)
    sw s0, 8(sp)

    # struct (12 bytes)
    # [0: name*] [4: age] [8: next*]

    # FIX: ID goes to a0, Argument goes to a1
    li a0, 9       # sbrk ID
    li a1, 12      # Size (12 bytes)
    ecall
    mv s0, a0      # s0 = struct address

    # 2. Read Name
    li a0, 4
    la a1, msg_nome
    ecall

    # newline
    li a0, 11
    li a1, 13
    ecall

    # read string and store in struct
    call LeString
    sw a0, 0(s0)

    # read age
    li a0, 4
    la a1, msg_idade
    ecall

    # store age
    call LeNumero
    sw a0, 4(s0)

    # pointer start as null
    sw zero, 8(s0)

    # return struct address
    mv a0, s0

    lw s0, 8(sp)
    lw ra, 12(sp)
    addi sp, sp, 16
    ret

# void ImprimePessoa(Pessoa *p)
ImprimePessoa:
    mv t0, a0

    # print name
    li a0, 4
    lw a1, 0(t0)
    ecall

    # Print Space
    li a0, 11
    li a1, 32
    ecall

    # print age
    li a0, 1
    lw a1, 4(t0)
    ecall

    # newline
    li a0, 11
    li a1, 13
    ecall
    ret


# HELPERS
LeString:
    addi sp, sp, -16
    sw ra, 12(sp)
    sw s0, 8(sp)

    # Read into temp
    la a0, buffer_temp
    li a1, 100       # Max size
    call fgets

    la a0, buffer_temp
    call strlen
    mv s1, a0        # s1 = length

    addi a1, s1, 1   # Size = len + 1 (\0)
    li a0, 9         # sbrk ID
    ecall
    mv s0, a0        # s0 = new heap address

    # copy to heap
    mv a0, s0
    la a1, buffer_temp
    call strcpy

    # Return new address
    mv a0, s0

    lw s0, 8(sp)
    lw ra, 12(sp)
    addi sp, sp, 16
    ret

LeNumero:
    addi sp, sp, -8
    sw ra, 4(sp)
    sw s0, 0(sp)
    li s0, 0
    li a0, 0x130
    ecall
LeNumero_poll:
    li a0, 0x131
    ecall
    beq a0, zero, LeNumero_fim # Break on EOF/Empty
    li t0, 1
    beq a0, t0, LeNumero_poll
    li t1, 10
    beq a1, t1, LeNumero_fim   # Break on \n
    li t1, 13
    beq a1, t1, LeNumero_fim   # Break on \r
    li t2, 48
    blt a1, t2, LeNumero_poll  # Ignore < '0'
    li t2, 57
    bgt a1, t2, LeNumero_poll  # Ignore > '9'
    addi a1, a1, -48
    li t2, 10
    mul s0, s0, t2
    add s0, s0, a1
    j LeNumero_poll
LeNumero_fim:
    mv a0, s0
    lw s0, 0(sp)
    lw ra, 4(sp)
    addi sp, sp, 8
    ret

fgets:
    mv t0, a0
    addi t3, a1, -1
    li t4, 0
    li a0, 0x130
    ecall
fgets_poll:
    bge t4, t3, fgets_end
    li a0, 0x131
    ecall
    li t1, 2
    beq a0, t1, fgets_read
    beq a0, zero, fgets_end
    j fgets_poll
fgets_read:
    li t2, 10
    beq a1, t2, fgets_end
    sb a1, 0(t0)
    addi t0, t0, 1
    addi t4, t4, 1
    j fgets_poll
fgets_end:
    sb zero, 0(t0)
    ret

# int strlen(char *s)
strlen:
    li t0, 0
strlen_loop:
    add t1, a0, t0
    lbu t2, 0(t1)
    beq t2, zero, strlen_end
    addi t0, t0, 1
    j strlen_loop
strlen_end:
    mv a0, t0
    ret

# char * strcpy(char *dest, char *src)
strcpy:
    mv t0, a0 # dest
    mv t1, a1 # src
strcpy_loop:
    lbu t2, 0(t1)
    sb t2, 0(t0)
    beq t2, zero, strcpy_done
    addi t0, t0, 1
    addi t1, t1, 1
    j strcpy_loop
strcpy_done:
    ret
```

Atividade 11

```
```

Atividade 12

```
.data
heap_head: .word 0
msg_alloc1: .string "1. Alocando 10 bytes (A)...\n"
msg_addrA:  .string "   -> Endereco de A: "
msg_free1:  .string "2. Liberando bloco A...\n"
msg_alloc2: .string "3. Alocando 10 bytes novamente (B)...\n"
msg_addrB:  .string "   -> Endereco de B: "

msg_success: .string "\n[SUCESSO] O endereco B eh igual ao A.\n"
msg_fail:    .string "\n[FALHA] O endereco B eh diferente de A.\n"

.text
main:
    # malloc
    li a0, 4
    la a1, msg_alloc1
    ecall

    li a0, 10
    call my_malloc
    mv s0, a0       # s0 = address of A

    # write to A
    li t0, 99
    sb t0, 0(s0)

    # print Address A
    li a0, 4
    la a1, msg_addrA
    ecall

    # print address in hex
    mv a1, s0
    li a0, 34
    ecall

    # newline
    li a0, 11
    li a1, 13
    ecall

    # free
    li a0, 4
    la a1, msg_free1
    ecall

    mv a0, s0
    call my_free

    # malloc
    li a0, 4
    la a1, msg_alloc2
    ecall

    li a0, 10
    call my_malloc
    mv s1, a0

    # print address b
    li a0, 4
    la a1, msg_addrB
    ecall

    mv a1, s1
    li a0, 34
    ecall

    # newline
    li a0, 11
    li a1, 13
    ecall

    # check equal
    beq s0, s1, print_success

    # if not
    li a0, 4
    la a1, msg_fail
    ecall
    j end_program

print_success:
    li a0, 4
    la a1, msg_success
    ecall

end_program:
    li a0, 10
    ecall

# void * my_malloc(int size)
my_malloc:
    addi sp, sp, -16
    sw s0, 12(sp) # requested size
    sw s1, 8(sp)  # current block ptr
    sw ra, 0(sp)

    mv s0, a0     # Size needed

    # Load head
    la t0, heap_head
    lw s1, 0(t0)

malloc_search:
    # end of list -> create new
    beq s1, zero, malloc_new

    # check if free (offset 4)
    lw t1, 4(s1)
    bne t1, zero, malloc_next

    # check size (offset 0)
    lw t2, 0(s1)

    # if block too small
    blt t2, s0, malloc_next

    # mark -> free and useable block
    li t3, 1
    sw t3, 4(s1)

    # return pointer to area
    addi a0, s1, 16
    j malloc_ret
```

```
# goes next block
malloc_next:
    lw s1, 12(s1)
    j malloc_search

malloc_new:
    # alloc => Size + 16 bytes header

    addi a1, s0, 16
    li a0, 9
    ecall
    mv s1, a0

    # Setup Header
    sw s0, 0(s1)  # size
    li t0, 1
    sw t0, 4(s1)  # busy
    # Offset 8 is 'data' ptr
    addi t1, s1, 16
    sw t1, 8(s1)
    sw zero, 12(s1) # next = null

    # insert into list
    la t0, heap_head
    lw t1, 0(t0)
    beq t1, zero, malloc_first

    # Traverse to end
    mv t2, t1
find_tail:
    lw t3, 12(t2)
    beq t3, zero, append_block
    mv t2, t3
    j find_tail

append_block:
    sw s1, 12(t2)
    j malloc_finish

# update heap
malloc_first:
    sw s1, 0(t0)

# return data address
malloc_finish:
    addi a0, s1, 16

malloc_ret:
    lw ra, 0(sp)
    lw s1, 8(sp)
    lw s0, 12(sp)
    addi sp, sp, 16
    ret

# void my_free(void *ptr)
my_free:
    # ptr points to data, header is 16 bytes back
    addi t0, a0, -16

    # mark set to false (0)
    sw zero, 4(t0)
    ret
```