Exercício 8 - AC 2 - João Comini César de Andrade

Atividade 1

```
.data
vetor:
    .word 1, 2, 3, 4, 5
msg_resultado:
    .asciz "Vetor modificado: "
espaco:
    .asciz " "

.text
.globl main
main:
    # Parâmetros para MultiplicaVetor(unsigned N, unsigned *v, unsigned fator)
    li   a0, 5          # a0 = N = 5
    la   a1, vetor      # a1 = &vetor
    li   a2, 10         # a2 = fator = 10

    call MultiplicaVetor # Chama a função

    # Opcional: Imprime o vetor modificado para verificar
    li   a7, 4
    la   a1, msg_resultado
    ecall

    la   s0, vetor     # Endereço base
    li   s1, 5         # Contador
loop_print:
    beqz s1, fim_main
    lw   a1, 0(s0)     # Carrega vetor[i]
    li   a7, 1
    ecall

    li   a7, 4
    la   a1, espaco
    ecall

    addi s0, s0, 4
    addi s1, s1, -1
    j    loop_print

fim_main:
    # Fim do programa
```

```
    li   a7, 10
    ecall
MultiplicaVetor:
    # Salva registradores s0, s1, s2 e ra na pilha
    addi sp, sp, -16
    sw   s0, 12(sp)
    sw   s1, 8(sp)
    sw   s2, 4(sp)
    sw   ra, 0(sp)

    # Move parâmetros para registradores salvos
    mv   s0, a0          # s0 = N
    mv   s1, a1          # s1 = *v
    mv   s2, a2          # s2 = fator
for:
    beqz s0, fim_multiplica_vetor # Se N == 0, fim

    # Prepara parâmetros para Multiplica(x, y)
    lw   a0, 0(s1)       # a0 = v[i]
    mv   a1, s2          # a1 = fator
    call Multiplica      # Chama a função Multiplica

    # Salva o resultado (em a0) de volta no vetor
    sw   a0, 0(s1)

    # Próxima iteração
    addi s1, s1, 4       # v++ (próximo endereço)
    addi s0, s0, -1      # N--
    j    for

fim_multiplica_vetor:
    # Restaura registradores da pilha
    lw   ra, 0(sp)
    lw   s2, 4(sp)
    lw   s1, 8(sp)
    lw   s0, 12(sp)
    addi sp, sp, 16
    ret
```

Atividade 2

```
Multiplica:
    mul  a0, a0, a1      # a0 = x * y
    ret                  # Retorna (resultado já está em a0)
```

Atividade 3

```
.data
vetor:
    .word 10, 20, 30, 40, 50
msg_resultado:
    .asciz "A soma total do vetor eh: "

.text
.globl main
main:
    # Parâmetros para SomaVetor(unsigned N, unsigned *v)
    li   a0, 5          # a0 = N = 5
    la   a1, vetor      # a1 = &vetor

    call SomaVetor      # Chama a função

    # O resultado da soma está em a0
    mv   a1, a0         # Move o resultado para a1 para impressão

    # Imprime a mensagem
    li   a7, 4
    la   a1, msg_resultado
    ecall

    # Imprime o resultado
    li   a7, 1
    mv   a1, a0
    ecall

    # Fim do programa
    li   a7, 10
    ecall

SomaVetor:
    # Move parâmetros para registradores temporários
    mv   t0, a0         # t0 = N (contador)
    mv   t1, a1         # t1 = *v (ponteiro para o vetor)

    li   a0, 0          # Zera a0 (será o acumulador da soma e o retorno)

loop_soma:
    beqz t0, fim_soma_vetor # Se N == 0, fim
```

```
    lw   t2, 0(t1)         # t2 = v[i]
    add  a0, a0, t2        # soma = soma + v[i]

    addi t1, t1, 4         # v++ (próximo endereço)
    addi t0, t0, -1        # N--
    j    loop_soma

fim_soma_vetor:
    ret                    # Retorna (soma já está em a0)
```

Atividade 4

```
.data
msg_resultado:
    .asciz "O tamanho da string digitada eh: "

.text
.globl main
main:
    # Chama a função que fará todo o trabalho
    call TamanhoString

    # O resultado (tamanho) está em a0
    mv   a1, a0          # Move para a1 para impressão

    # Imprime a mensagem
    li   a7, 4
    la   a1, msg_resultado
    ecall

    # Imprime o tamanho
    li   a7, 1
    mv   a1, a0
    ecall

    # Fim do programa
    li   a7, 10
    ecall

TamanhoString:
    # Aloca espaço na pilha:
    # 24 bytes para a string (alinhado, para 21 bytes pedidos)
    # 4 bytes para salvar o ra (pois vamos chamar strlen e ecall)
    # Total = 28 bytes
    addi sp, sp, -28
```

```
    sw   ra, 24(sp)      # Salva o endereço de retorno

    # A variável local 'string' está agora em sp+0

    # Lê a string do teclado (syscall 8)
    li   a7, 8
    mv   a1, sp          # a1 = endereço do buffer (nossa variável local)
    li   a2, 21          # a2 = tamanho máximo (20 chars + 1)
    ecall

    # Chama strlen, passando o endereço da nossa string
    mv   a0, sp          # a0 = &string
    call strlen          # Retorno (tamanho) estará em a0

    # Restaura a pilha
    lw   ra, 24(sp)      # Recupera o endereço de retorno
    addi sp, sp, 28      # Libera o espaço da pilha

    ret                  # Retorna (tamanho já está em a0)


strlen:
    li   a1, 0           # a1 = contador (será o valor de retorno)
loop_strlen:
    lb   t0, 0(a0)       # Carrega 1 byte (caractere)

    # Verifica fim da string (syscall 8 do RARS usa '\n')
    beqz t0, fim_strlen  # Fim se for '\0' (nulo)
    li   t1, 10
    beq  t0, t1, fim_strlen  # Fim se for '\n' (nova linha)

    addi a1, a1, 1       # contador++
    addi a0, a0, 1       # ponteiro++
    j    loop_strlen

fim_strlen:
    mv   a0, a1          # Move o resultado (contador) para a0
    ret
```

Atividade 5

```
.data
prompt_n1:   .asciz "Digite o primeiro numero: "
prompt_op:   .asciz "Digite a operacao (+ ou -): "
```

```
prompt_n2:   .asciz "Digite o segundo numero: "
prompt_res:  .asciz "Resultado: "
msg_erro:    .asciz "Operacao invalida!\n"

# Estrutura: { char (4 bytes), *funcao (4 bytes) }
operacoes:
    .word '+'          # Caractere '+'
    .word soma         # Endereço da função 'soma'
    .word '-'          # Caractere '-'
    .word subtracao    # Endereço da função 'subtracao'
fim_operacoes:
    .word 0

.text
.globl main
main:
    # Lê num1
    li a7, 4
    la a1, prompt_n1
    ecall
    li a7, 5
    ecall
    mv s0, a0           # s0 = num1

    # Lê operação
    li a7, 4
    la a1, prompt_op
    ecall
    li a7, 12
    ecall
    mv s1, a0           # s1 = char

    # Lê num2
    li a7, 4
    la a1, prompt_n2
    ecall
    li a7, 5
    ecall
    mv s2, a0           # s2 = num2

    # Procura a função na nossa "struct"
    la t0, operacoes     # t0 = ponteiro para a struct
loop_busca:
    lw t1, 0(t0)         # t1 = operacoes[i].caracter
```

```
    beqz t1, erro_op     # Se carregar 0 (fim_operacoes), não achou

    beq t1, s1, op_encontrada # Se t1 == s1 (char), achou

    addi t0, t0, 8       # Próxima struct (char 4 bytes + func 4 bytes)
    j loop_busca

op_encontrada:
    lw t2, 4(t0)         # t2 = operacoes[i].op (endereço da função)

    # Prepara argumentos
    mv a0, s0            # a0 = num1
    mv a1, s2            # a1 = num2

    # Chama a função no endereço carregado
    jalr t2              # Salva ra e pula para endereço em t2

    # O resultado está em a0
    j imprime_resultado

erro_op:
    li a7, 4
    la a1, msg_erro
    ecall
    j fim_calculadora

imprime_resultado:
    mv a1, a0
    li a7, 4
    la a1, prompt_res
    ecall

    li a7, 1
    mv a1, a0
    ecall

fim_calculadora:
    li a7, 10
    ecall

soma:
    add a0, a0, a1
    ret
```

```
subtracao:
    sub a0, a0, a1
    ret
```

Atividade 6

```
.data
prompt_n:    .asciz "Digite um numero para o fatorial: "
prompt_res:  .asciz "Resultado: "

.text
.globl main
main:
    # Lê N
    li a7, 4
    la a1, prompt_n
    ecall
    li a7, 5
    ecall
    # a0 já contém N, pronto para a chamada

    call fatorial

    # Imprime o resultado (está em a0)
    mv a1, a0
    li a7, 4
    la a1, prompt_res
    ecall

    li a7, 1
    mv a1, a0
    ecall

    # Fim
    li a7, 10
    ecall

fatorial:
    # Salva registradores na pilha
    # 8 bytes: 4 para ra, 4 para s0 (onde salvaremos n)
    addi sp, sp, -8
    sw   ra, 4(sp)       # Salva endereço de retorno
    sw   s0, 0(sp)       # Salva s0 (que usaremos para guardar n)
```

```
    mv   s0, a0          # s0 = n

    # Caso Base: n <= 1 ?
    li   t0, 1
    bgt  s0, t0, recursivo # Se n > 1, pula para o passo recursivo

    # Caso base: n <= 1
    li   a0, 1           # Retorna 1
    j    fim_fatorial

recursivo:
    # Passo recursivo: n * fatorial(n-1)
    addi a0, s0, -1     # a0 = n - 1 (parâmetro para chamada recursiva)
    call fatorial       # Chama fatorial(n-1). Resultado está em a0

    # Agora, calcula n * resultado_da_chamada
    mv   a1, a0          # a1 = fatorial(n-1)
    mv   a0, s0          # a0 = n

    call Multiplica     # Chama Multiplica(n, fatorial(n-1))
    # Resultado final está em a0

fim_fatorial:
    # Restaura registradores da pilha
    lw   s0, 0(sp)
    lw   ra, 4(sp)
    addi sp, sp, 8

    ret

Multiplica:
    mul  a0, a0, a1        # a0 = x * y
    ret                  # Retorna
```