Exercício 7 - AC 2 - João Comini César de Andrade

Atividade 1

```
.data
vetor:
    .word 10, 20, 30  # Declara um vetor com 3 palavras (inteiros)
.text
.globl main
main:
    # Carrega o endereço base do vetor em s0.
    # A pseudo-instrução 'la' é convertida em lui + addi pelo montador.
    la s0, vetor

    # Carrega os elementos do vetor usando offsets
    # lw t0, 0(s0) -> carrega a palavra no endereço (s0 + 0)
    lw t0, 0(s0)      # t0 = vetor[0] = 10
    lw t1, 4(s0)      # t1 = vetor[1] = 20 (offset de 4 bytes)
    lw t2, 8(s0)      # t2 = vetor[2] = 30 (offset de 8 bytes)

    # Soma os elementos
    add t3, t0, t1    # t3 = 10 + 20
    add s1, t3, t2    # s1 = 30 + 30 = 60

    # Imprime o resultado final
    li a7, 1
    mv a1, s1
    ecall

    # Fim do programa
    li a7, 10
    ecall
```

Atividade 2

```
.data
vetor:
    .word 10, 20, 30

.text
.globl main
main:
    # Carrega o endereço base do vetor em s0
    la s0, vetor
```

```
    # --- Incrementa cada elemento ---
    # Elemento 0
    lw t0, 0(s0)      # Carrega vetor[0]
    addi t0, t0, 1    # Incrementa
    sw t0, 0(s0)      # Salva de volta em vetor[0]

    # Elemento 1
    lw t0, 4(s0)      # Carrega vetor[1]
    addi t0, t0, 1    # Incrementa
    sw t0, 4(s0)      # Salva de volta em vetor[1]

    # Elemento 2
    lw t0, 8(s0)      # Carrega vetor[2]
    addi t0, t0, 1    # Incrementa
    sw t0, 8(s0)      # Salva de volta em vetor[2]


    # --- Imprime o novo vetor ---
    # Usa um laço para imprimir os 3 elementos
    li s1, 3          # Contador de elementos
    li s2, 0          # Offset inicial (em bytes)

loop_impressao:
    # Condição de parada do loop
    beqz s1, fim

    # Carrega o elemento atual do vetor
    add t0, s0, s2    # Calcula o endereço do elemento atual
    lw a1, 0(t0)      # Carrega o valor para a impressão
    li a7, 1
    ecall

    # Imprime um espaço para separar os números
    li a7, 11
    li a1, ' '
    ecall

    # Atualiza para o próximo elemento
    addi s2, s2, 4    # Próximo offset (4 bytes)
    addi s1, s1, -1   # Decrementa o contador
    j loop_impressao

fim:
```

```
    # Fim do programa
    li a7, 10
    ecall
```

Atividade 3

```
.rodata # Região de memória para constantes (read-only)
vetor_a:
    .word 1, 2, 3, 4, 5
vetor_b:
    .word 10, 20, 30, 40, 50

.data # Região para variáveis
vetor_c:
    .space 20 # Reserva 20 bytes (5 palavras de 4 bytes) para o vetor resultado

.text
.globl main
main:
    # Carrega os endereços base dos vetores
    la s0, vetor_a
    la s1, vetor_b
    la s2, vetor_c

    # Configuração do laço
    li s3, 5          # Contador (5 elementos)
    li t0, 0          # Offset inicial (em bytes)

loop_soma_vetores:
    # Condição de parada
    beqz s3, soma_final

    # Calcula endereço do elemento atual de cada vetor
    add t1, s0, t0    # Endereço de A[i]
    add t2, s1, t0    # Endereço de B[i]
    add t3, s2, t0    # Endereço de C[i]

    # Carrega os valores de A[i] e B[i]
    lw t4, 0(t1)
    lw t5, 0(t2)

    # Soma e armazena em C[i]
    add t6, t4, t5
```

```
    sw t6, 0(t3)

    # Atualiza para a próxima iteração
    addi t0, t0, 4    # Incrementa o offset em 4 bytes
    addi s3, s3, -1   # Decrementa o contador
    j loop_soma_vetores

soma_final:
    # Agora, soma todos os elementos do vetor_c
    la s2, vetor_c    # Recarrega endereço base de C
    li s3, 5          # Reinicia contador
    li t0, 0          # Reinicia offset
    li s4, 0          # s4 guardará a soma total

loop_soma_c:
    beqz s3, imprime_soma_final

    add t1, s2, t0    # Endereço C[i]
    lw t2, 0(t1)      # Carrega C[i]
    add s4, s4, t2    # Acumula a soma em s4

    addi t0, t0, 4
    addi s3, s3, -1
    j loop_soma_c

imprime_soma_final:
    li a7, 1
    mv a1, s4
    ecall

    li a7, 10
    ecall
```

Atividade 4

```
.data
matriz:
    .space 24 # Reserva 24 bytes (2 linhas * 3 colunas * 4 bytes/elemento)

.text
.globl main
main:
    # Registradores:
    # s0: endereço base da matriz
```

```
    # s1: número de colunas (3)
    # t0: contador de linha 'i'
    # t1: contador de coluna 'j'

    la s0, matriz
    li s1, 3          # Número de colunas = 3

    li t0, 0          # i = 0
loop_i:
    li t2, 2          # i < 2
    bge t0, t2, fim   # Se i >= 2, termina

    li t1, 0          # j = 0
loop_j:
    li t3, 3          # j < 3
    bge t1, t3, proxima_linha # Se j >= 3, vai para i++

    # Calcula o endereço do elemento M[i][j]
    # offset_em_bytes = (i * num_cols + j) * 4
    mul t4, t0, s1    # t4 = i * num_cols
    add t4, t4, t1    # t4 = i * num_cols + j
    slli t4, t4, 2    # t4 = (i * num_cols + j) * 4 (multiplica por 4)
    add t5, s0, t4    # t5 = endereço_base + offset

    # Calcula o valor a ser armazenado (i + j)
    add t6, t0, t1

    # Armazena o valor na matriz
    sw t6, 0(t5)

    # j++
    addi t1, t1, 1
    j loop_j

proxima_linha:
    # i++
    addi t0, t0, 1
    j loop_i

fim:
    li a7, 10
    ecall
```

Atividade 5

```
.data
```

```
minha_string:
    .space 21 # Reserva 20 bytes para caracteres + 1 para o nulo ('\0')

.text
.globl main
main:
    # Prepara a chamada de sistema para ler string (syscall 8)
    li a7, 8
    la a1, minha_string # Passa o endereço do buffer
    li a2, 20           # Passa o tamanho máximo

    # Lê a string do teclado
    ecall

    # --- Conta os caracteres lidos ---
    # s0: endereço atual do caractere
    # s1: contador de tamanho
    la s0, minha_string
    li s1, 0

loop_contagem:
    # Carrega um byte (caractere) da string
    lb t0, 0(s0)

    # Se o caractere for nulo ('\0') ou nova linha ('\n'), fim da contagem
    # Syscall 8 do RARS termina com '\n' e '\0'
    beqz t0, fim_contagem # Se for 0, fim
    li t1, 10             # Código ASCII para '\n'
    beq t0, t1, fim_contagem # Se for '\n', fim

    # Se não for fim, incrementa contador e ponteiro
    addi s1, s1, 1      # tamanho++
    addi s0, s0, 1      # próximo caractere
    j loop_contagem

fim_contagem:
    # Imprime o tamanho da string
    li a7, 1
    mv a1, s1
    ecall

    li a7, 10
    ecall
```

Atividade 6

```
.data
# declaração do vetor de pontos com 5 posições de 2 inteiros cada
vetor_pontos:
    # (x, y)
    .word 1, 2
    .word 5, 1
    .word 3, 4
    .word 5, 3  # Este será o resultado (empata com (5,1) em X, mas Y é maior)
    .word 2, 5

.text
.globl main
main:
    # s0: endereço base do vetor
    # s1: coordenada X do ponto mais à direita/acima encontrado
    # s2: coordenada Y do ponto mais à direita/acima encontrado
    # t0: ponteiro para o ponto atual no loop
    # t1: contador do loop (de 1 a 4)
    # t2: coordenada X do ponto atual
    # t3: coordenada Y do ponto atual
    la s0, vetor_pontos

    # copia o primeiro ponto para os registradores de ponto superior direito
    lw s1, 0(s0)      # max_x = vetor_pontos[0].x
    lw s2, 4(s0)      # max_y = vetor_pontos[0].y

    # loop para percorrer os demais pontos do vetor (do 2º ao 5º)
    li t1, 4          # Contador para 4 iterações restantes
    add t0, s0, 8     # Ponteiro para o segundo ponto (offset 8)

loop_pontos:
    beqz t1, fim_loop # Se o contador chegar a zero, fim

    # Carrega o ponto atual
    lw t2, 0(t0)      # current_x
    lw t3, 4(t0)      # current_y

    # compara a coordenada x do ponto atual com a coordenada x do ponto superior
direito
    blt t2, s1, proximo_ponto # if (current_x < max_x) -> ignora
    bgt t2, s1, novo_max      # if (current_x > max_x) -> novo máximo
```

```
    # Se chegou aqui, current_x == max_x, então...
    # compara a coordenada y do ponto atual com a coordenada y do ponto superior
direito
    ble t3, s2, proximo_ponto # if (current_y <= max_y) -> ignora

novo_max:
    # Encontrou um ponto melhor, atualiza s1 e s2
    mv s1, t2
    mv s2, t3

proximo_ponto:
    # Avança para o próximo ponto
    addi t0, t0, 8      # Próximo ponto (offset de 8 bytes = 2 words)
    addi t1, t1, -1     # Decrementa o contador
    j loop_pontos

fim_loop:
    # imprime o ponto superior direito na tela
    # Imprime X
    li a7, 1
    mv a1, s1
    ecall

    # Imprime uma vírgula e espaço
    li a7, 11
    li a1, ','
    ecall
    li a1, ' '
    ecall

    # Imprime Y
    li a7, 1
    mv a1, s2
    ecall

    li a7, 10
    ecall
```