Lista 2

Nome: Juliana Aparecida Borges

Questão 1

I)

```
    li $t0, 0          # i = 0
loop:
    lb $t1, 0($a1)     # y[i]
    sb $t1, 0($a0)     # x[i] = y[i]
    beq $t1, $zero, end  # (x[i] = y[i] == '\0', termina
    addi $a0, $a0, 1   # x[i++]
    addi $a1, $a1, 1   # y[i++]
    j loop             # segue no loop
end:
    jr $ra             # retorna
```

Cálculo:

cada iteração:
lb: 1 ciclo (CPI = 1)
sb: CPI = 3
beq: CPI = 1
addi: CPI = 1 (mudando 2x)

ciclos por iteração: 1+3+1+1+1 = 7 ciclos
N iterações, total: 7N
Tempo de CPU = $\frac{7N}{100 \cdot 10^6}$ segundos //

II)

```
swap:
    sll $t2, $a2, 3
    add $t0, $a0, $t2   # v[k]
    ld $t1, 0($t0)
    ld $t3, 8($t0)
    sd $t1, 8($t0)      # v[k] em v[k+1]
    sd $t3, 0($t0)      # temp em v[k]
    jr $ra
```

Cálculo
cada iteração
2 ld: 6 ciclos (2.3)
2 sub: 6 ciclos (2.3)
slt: CPI=1
add: CPI=1

ciclos = 6+6+1+1 = 14 ciclos
Tempo CPU = $\frac{14}{100.10^6}$ (I) segundos

III) BinarySearch:

```
    li $t0, 0          # low = 0
    li $t1, -1         # high = N-1
loop:
    bgt $t0, $t1, check   # enquanto low <= high
    li $v0, -1            # retorna -1 se não encontrado
    jr $ra
check:
    add $t2, $t0, $t1     # mid = (low + high)/2
    srl $t2, $t2, 1
    lw $t3, 0($a0)        # A[mid]
    beq $t3, $a1, found   # se A[mid] == X, encontrado
    blt $t3, $a1, update_high
    addi $t1, $t2, -1     # high = mid - 1
    j loop
update_high:
    addi $t0, $t2, 1      # low = mid + 1
    j loop
found:
    move $v0, $t2         # retorna o índice encontrado
    jr $ra
```

Cálculo
cada iteração
1 add, sra, addi: 1 ciclo cada
ld: 3 ciclos
beq, blt: 2 ciclos cada

ciclos = 4+3+4 = 11 ciclos
Tempo CPU = $\frac{11N}{100.10^6}$ segundos

DOM SEG TER QUA QUI SEX SÁB

Questão 2 a) Bubble Sort em C

```
#include <stdio.h>

void bubbleSort(char arr[], int n){
    char temp;
    for(int i=0; i<n-1; i++){
        for(int j=0; j<n-1-i; j++){
            if(arr[j] > arr[j+1]){
                temp = arr[j];
                arr[j] = arr[j+1];
                arr[j+1] = temp;
            }
        }
    }
}
int main(){
    char arr[10] = {'z', 'x', 's', 'b', 'a', 'e', 't', 'w', 'm', 'o'};
    int n = 10;
    bubbleSort(arr, n);
    for(int i=0; i<n; i++){
        printf("%c ", arr[i]);
    }
    return 0;
}
```

Em MIPS

```
.data
arr: .byte 'z', 'x', 's', 'b', 'a', 'e', 't', 'w', 'm', 'o'

.text
.globl main
main:
    li $t0, 9    # i = 9 (n-1)
```

outer_loop:
beq $t0, $zero, end_outer_loop # i == 0, termino
li $t1, 0   # j = 0
inner_loop:
beq $t1, $t0, end_inner_loop   # j == i, termino
lw $t2, arr($t1)   # arr[j] em $t2
lw $t3, 1($t1)   # arr[j+1]
ble $t2, $t3, skip_swap   # arr[j] <= arr[j+1], pula
sw $t2, 1($t1)   # arr[j+1] = arr[j]
sw $t3, arr($t1)   # arr[j] = arr[j+1]
skip_swap:
addi $t1, $t1, 1   # j += 1
j inner_loop
end_inner_loop:
addi $t0, $t0, -1   # i -= 1
j outer_loop
end_outer_loop:
jr $ra # impressão e finalização

b) Selection Sort em C

```
#include <stdio.h>

void selectionSort(char arr[], int n){
    int i, j, min_idx;
    char temp;
    for(i=0; i<n-1; i++){
        min_idx = i;
        for(j=i+1; j<n; j++){
            if(arr[j] < arr[min_idx]){
                min_idx = j;
            }
        }
        temp = arr[min_idx];
        arr[min_idx] = arr[i];
        arr[i] = temp;
```

```
int main(){
  char aux[10]={'z','a','h','b','c','x','y','k','l','m'};
  int n=10;
  selectionSort(aux, n);
  for(int i=0; i<n; i++){
    printf("%c ", aux[i]);
  }
  return 0;
}
```

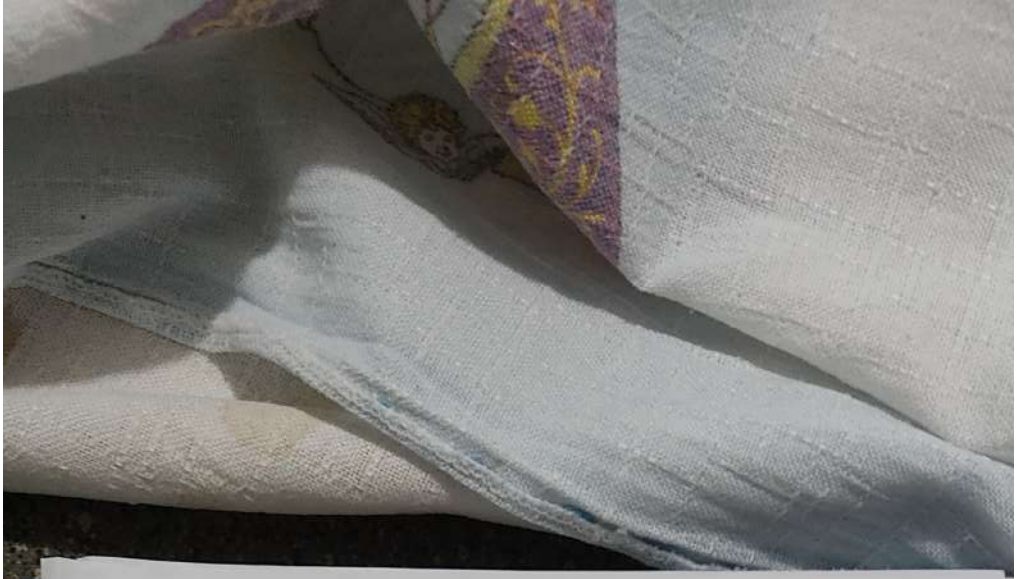
Em MIPS

```
.data
aux: .byte 'z','a','h','b','c','x','y','k','l','m'

.text
.globl main
main:
  li $t0, 0                  # i=0
outer_loop:
  li $t1, $t0                # min_idx = i
  li $t2, $t0                # y=i
  add $t2, $t2, 1            # j=i+1
inner_loop:
  li $t3, 10                 # n=10
  beq $t2, $t3, end_inner_loop  # j==n, termina
  lb $t4, aux($t2)           # aux[j]
  lb $t5, aux($t1)           # aux[min_idx]
  bge $t5, $t4, update_min_idx  # aux[min_idx]>=aux[j], atualiza min_idx
  j skip_update
update_min_idx:
  move $t1, $t2              # min_idx = y
```

```
addi $t2, $t2, 1          # j += 1
j inner_loop
end_inner_loop:
lw $t6, 0($t0)            # aux[i] para troca
lw $t7, 0($t1)            # aux[min_idx]
sw $t6, 0($t1)            # aux[min_idx] = aux[i]
sw $t7, 0($t0)            # aux[i] = aux[min_idx]
addi $t0, $t0, 1          # i += 1
li $t3, 9                 # n-1 = 9
bne $t0, $t3, outer_loop  # i != n-1, continua o loop
jr $ra                    # impressão e finalização
```

Questão 3 Busca da instrução: PC envia o endereço da próxima instrução para a Memória de Instruções, que retorna a instrução.
Decodificação: instrução é decodificada, identificando tanto os registradores $rt e o valor (offset).
Leitura dos Registradores: $rs é lido do Banco de Registradores.
Cálculo do Endereço: ALU soma o $rs com o offset, gerando o endereço efetivo na Memória de dados.
Acesso à Memória de Dados: fornece o valor no endereço calculado
Escrita no Registradores: valor da memória é carregado no $rt
Atualização do PC: o PC é incrementado em 4 para apontar para a próxima instrução.



DOM SEG TER QUA QUI SEX SÁB
DOM LUN MAR MIE JUE VIE SÁB

Quinta 9

module Semaforo(input wire clk, input wire rst, input wire
pedestre, output reg luzVerde, // luzAmarela,
// luzVermelha);

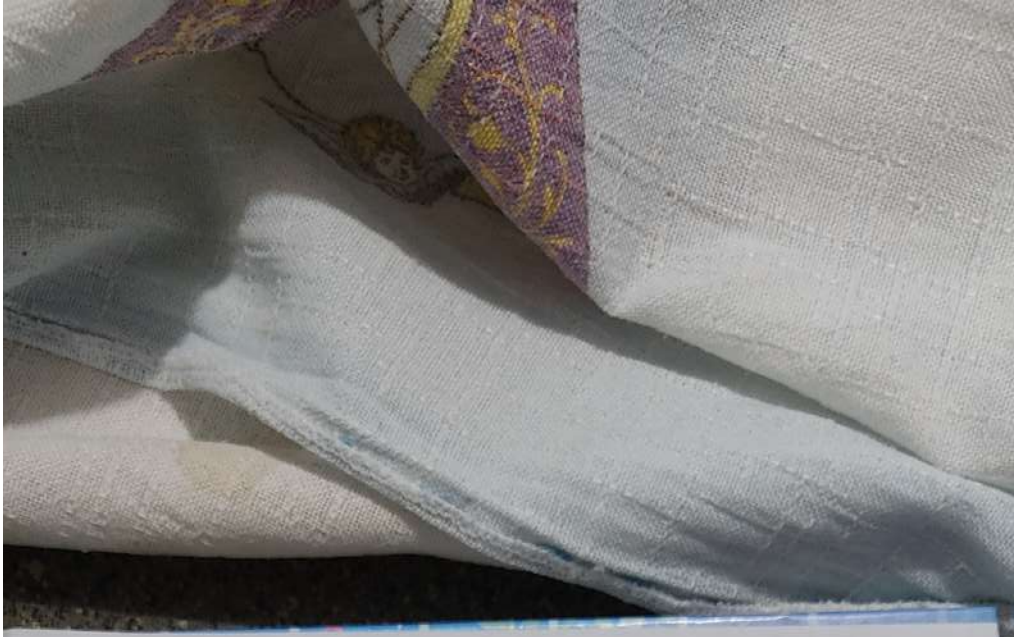
typedef enum logic [1:0]
{VERDE = 2'b00,
AMARELO = 2'b01,
VERMELHO = 2'b10
} state_t;
state_t estadoAtual, estadoProximo;
reg [4:0] contadorTempo;

localparam TEMPO_VERDE = 30;
// TEMPO_AMARELO = 10;
// TEMPO_VERMELHO = 30;

always @(posedge clk, or posedge reset) begin
if(reset) begin
estadoAtual <= VERDE;
contadorTempo <= 0;
end else begin
estadoAtual <= estadoProximo;
if(contadorTempo > 0)
contadorTempo <= contadorTempo - 1;
end
end

always @(*) begin
luzVerde = 0;
luzAmarela = 0;
luzVermelha = 0;
case(estadoAtual)
VERDE: begin

```
        luzVerde = 1;
        if (contadorTempo == 0) begin
          if (piscar)
            estadoProximo = AMARELO;
          else
            estadoProximo = VERMELHO;
        end else begin
          estadoProximo = VERDE;
        end
      end
      AMARELO: begin
        luzAmarelo = 1;
        if (contadorTempo == 0) begin
          estadoProximo = VERMELHO;
        end else begin
          estadoProximo = AMARELO;
        end
      end
      VERMELHO: begin
        luzVermelha = 1;
        if (contadorTempo == 0) begin
          estadoProximo = VERDE;
        end else begin
          estadoProximo = VERMELHO;
        end
      end
      default: estadoProximo = VERDE;
    endcase
  end
  always @(posedge clk or posedge reset) begin
    if (reset) begin
      contadorTempo <= TEMPO_VERDE;
    end else begin
      case (estadoProximo)
```



DOM SEG TER QUA QUI SEX SÁB

VERDE: contadorTempo <= TEMPO_VERDE;
AMARELO: contadorTempo <= TEMPO_AMARELO;
VERMELHO: contadorTempo <= TEMPO_VERMELHO;

endcase
end
end
endmodule