

Relatório

Mateus Barbosa e Matheus de Oliveira Rocha

Universidade do Vale do Itajaí - UNIVALI
Escola do Mar, Ciência e Tecnologia
Ciência da Computação

{mateus.barbosa, matheus.rocha}@edu.univali.br

Arquitetura e Organização de Processadores

Avaliação 01 – Programação em linguagem de montagem

Thiago Felski Pereira

1. Introdução

Este documento é o relatório descrevendo a implementação de 2 programas usando a Linguagem de Montagem do Risc-V, mostrando os valores usado e as estatísticas referentes a execução das Instruções. Além de recriar o código usando uma linguagem de Alto Nível, sendo ela C/C++.

2. Programa 01

2.1 Enunciado: Implemente um programa que leia dois vetores via console e, após a leitura dos vetores, troque os conteúdos dos dois vetores. Por fim, o programa deve imprimir esse novo vetor na tela.

2.2 Código fonte em Linguagem de Alto Nível C/C++

// Disciplina : Arquitetura e Organização de Computadores
// Atividade : Avaliação 01 – Programação em Linguagem de Alto nível
// Programa 01
// Grupo : - Mateus Barbosa
// - Matheus de Oliveira Rocha

```
using namespace std;

#include <iostream>

#define MAX_TAM 8
#define MIN_TAM 1

int main()
{
    int input_vec_tam;

    // Solicita tamanho dos vetores
    do
    {
        cout << "\nEntre com o tamanho dos vetores (máx = 8): ";
        cin >> input_vec_tam;
        if (input_vec_tam < MIN_TAM || input_vec_tam > MAX_TAM)
            cout << "\nValor inválido!\n";
    } while (input_vec_tam < MIN_TAM || input_vec_tam > MAX_TAM);
```

```

int Vetor_A[input_vec_tam], Vetor_B[input_vec_tam];

// Preenche Vetor_A com os valores digitados pelo usuario
for (int i = 0; i < input_vec_tam; i++)
{
    cout << "\nVetor_A[" << i << "] = ";
    cin >> Vetor_A[i];
}

// Preenche Vetor_B com os valores digitados pelo usuario
for (int i = 0; i < input_vec_tam; i++)
{
    cout << "\nVetor_B[" << i << "] = ";
    cin >> Vetor_B[i];
}

// Troca os valores dos Vetores
int auxiliar; // Usado como variavel temp para armazenar o valor de um dos Vetores
for (int i = 0; i < input_vec_tam; i++)
{
    auxiliar = Vetor_A[i];
    Vetor_A[i] = Vetor_B[i];
    Vetor_B[i] = auxiliar;
}

// Mostra Vetor_A com os valores tracodos com o Vetor_B
for (int i = 0; i < input_vec_tam; i++)
{
    cout << "\nVetor_A[" << i << "] = " << Vetor_A[i];
}

// Mostra Vetor_B com os valores tracodos com o Vetor_A
for (int i = 0; i < input_vec_tam; i++)

```

```

{
    cout << "\nVetor_B[" << i << "] = " << Vetor_B[i];
}

return 0;
}

```

2.3 Código fonte em Linguagem de Montagem do Risc-V

Explicação da Lógica do Código:

O programa solicita ao usuário o tamanho de dois vetores, em seguida, solicita ao usuário os valores que compõem o primeiro vetor e, em seguida, os valores que compõem o segundo vetor. A lógica por trás do programa é simples, mas é um pouco extensa.

Inicialmente, há um segmento de dados onde é definido o valor dos vetores Vetor_A e Vetor_B com o valor 0, além do tamanho máximo e mínimo permitido para o vetor (8 e 1, respectivamente) e as strings que serão usadas no decorrer do programa.

Em seguida, há um segmento de código que começa com a carga dos registradores t0 e t1 com o valor de min_tam e max_tam, respectivamente. Em seguida, são carregados os registradores s10 e s11 com Vetor_A e Vetor_B, respectivamente.

O programa entra em um loop chamado "loop_define_vetor_tam" onde o usuário é solicitado a inserir o tamanho dos vetores. Se o valor inserido pelo usuário for menor que 1, uma mensagem de erro será exibida e o loop será executado novamente. Se o valor inserido pelo usuário for maior que 8, uma mensagem de erro será exibida e o loop será executado novamente. Caso contrário, o programa avança para o loop "define_Vetor_A_contador".

A próxima função "define_Vetor_A_contador" define um contador (t0) como 0 e carrega o texto "texto_input_Vetor_A" para o registrador t1. Em seguida, ele chama a função "loop_preenche_Vetor_A".

O loop "loop_preenche_Vetor_A" solicita que o usuário insira um valor para a posição atual do vetor, exibe o índice da posição atual e armazena o valor digitado pelo usuário no vetor. O loop é executado até que o contador (t0) seja maior ou igual ao tamanho informado pelo usuário. Se o contador for menor que o tamanho informado, o programa solicita o próximo valor do usuário, caso contrário, o programa passa para a próxima etapa, que é a definição do segundo vetor.

A próxima função "define_Vetor_B_contador" é semelhante à função "define_Vetor_A_contador", mas carrega "texto_input_Vetor_B" no registrador t1 e chama a função "loop_preenche_Vetor_B", que solicita os valores do segundo vetor.

O loop "loop_preenche_Vetor_B" é semelhante ao "loop_preenche_Vetor_A", mas é usado para preencher o vetor Vetor_B.

Após os 2 vetores estarem preenchidos, é executada a função chamada "troca_valores_vetor", que é responsável por trocar o conteúdo dos dois vetores "Vetor_A" e "Vetor_B" entre si. O código usa um loop para percorrer os vetores e realiza a troca dos valores de cada posição. O loop começa no rótulo

"loop_troca_valores_vetor" e termina quando o contador t0 é maior ou igual ao tamanho informado pelo usuário, após passa a executar a instrução na label "print_Vetor_A".

Dentro do loop, as instruções calculam as posições do Vetor_A e Vetor_B na memória, carregam os valores nas respectivas posições em registradores auxiliares e, em seguida, trocam os valores entre si usando as instruções "sw" (store word) e "lw" (load word). Por fim, o contador é incrementado e o loop é reiniciado com a instrução "j" (jump) para "loop_troca_valores_vetor".

Após isso, vai para a parte final do código em que se mostra os valores no terminal que é feita pela função print_Vetor_A começa inicializando o contador t0 com o valor zero e em seguida chama o loop loop_print_Vetor_A. O loop verifica se o contador é maior ou igual ao tamanho do vetor s0. Se sim, ele sai do loop e chama a função print_Vetor_B. Se não, ele calcula a posição atual do vetor A, imprime a string "A[i] = " e o valor naquela posição do vetor A. Em seguida, ele incrementa o contador t0 e reinicia o loop.

A função print_Vetor_B é similar, com a diferença que ela calcula a posição atual do vetor B e imprime a string "B[i] = " e o valor naquela posição do vetor B.

Ambas as funções usam a syscall PrintString para imprimir as strings na tela e a syscall PrintInt para imprimir os valores dos vetores. Ao final do loop da função print_Vetor_B, o programa chama a função Exit que faz uma syscall Exit para finalizar a execução do programa.

Código do Risc-V:

```
# Disciplina: Arquitetura e Organização de Computadores
# Atividade: Avaliação 01 – Programação em Linguagem de Montagem
# Programa 01
# Grupo: - Mateus Barbosa
#           - Matheus de Oliveira Rocha

.data # Segmento de Dados
    Vetor_A: .word 0, 0, 0, 0, 0, 0, 0, 0 # Inicializa um vetor com 8 posições com o
                                             # valor 0
    Vetor_B: .word 0, 0, 0, 0, 0, 0, 0, 0 # Inicializa um vetor com 8 posições com o
                                             # valor 0
    max_tam: .word 8
    min_tam: .word 1
    texto_input_max_tam: .asciz "\nEntre com o tamanho dos vetores (máx = 8): "
    texto_valor_invalido: .asciz "\nValor inválido!\n"
    texto_input_Vetor_A: .asciz "\nVetor_A["
    texto_input_Vetor_B: .asciz "\nVetor_B["
```

```
    texto_fecha_index: .asciz "] = " # Essa string será usada para evitar ter que fazer
    algo muito trabalhoso para adicionar o número do contador
```

```
.text # Segmento de Código
```

```
    lw t0, min_tam
    lw t1, max_tam
    la s10, Vetor_A # Carrega o Vetor_A no registrador s10
    la s11, Vetor_B # Carrega o Vetor_B no registrador s11
```

```
loop_define_vetor_tam:
```

```
    # Imprime: String texto_input_max_tam
    addi a7, zero, 4 # Adiciona o valor 4 (PrintString) ao registrador de serviço a7
    la a0, texto_input_max_tam # Carrega ao registrador a0 o texto_input_max_tam
    ecall # Chama a syscall
```

```
    # Solicita: Int tamanho do vetor
    addi a7, zero, 5 # Adiciona o valor 5 (ReadInt) ao registrador de serviço a7
    ecall # Chama a syscall
    add s0, zero, a0 # O que foi digitado no console (registrador a0), é salvo no
    registrador s0
```

```
    bge s0, t0, if_maior_que # Se o valor de s0 (sem sinal) for maior que 1, vai para
    a função if_maior_que
```

```
    # Imprime: String valor_invalido
    addi a7, zero, 4 # Adiciona o valor 4 (PrintString) ao registrador de serviço a7
    la a0, texto_valor_invalido # Carrega ao registrador a0 o valor_invalido
    ecall # Chama a syscall
```

```
j loop_define_vetor_tam # "Pula" para a função loop_define_vetor_tam
```

```
if_maior_que: # Função que verifica se o valor informado é menor que tam
```

```
    ble s0, t1, define_Vetor_A_contador # Se o valor de s0 (sem sinal) for menor
    que max_tam, vai para a função loop_preenche_vetor_A
```

```

# Imprime: String valor_invalido
addi a7, zero, 4 # Adiciona o valor 4 (PrintString) ao registrador de serviço a7
la a0, texto_valor_invalido # Carrega ao registrador a0 o valor_invalido
ecall # Chama a syscall

j loop_define_vetor_tam # "Pula" para a função loop_define_vetor_tam

define_Vetor_A_contador:
    li t0, 0 # Define o counter para o valor 0
    la t1, texto_input_Vetor_A # Carrega o texto texto_input_vetor_A no registrador
t1
    j loop_preenche_Vetor_A # Executa a função loop_preenche_Vetor_A

loop_preenche_Vetor_A:
    bge t0, s0, define_Vetor_B_contador # Se o contador foi maior que o tamanho
informado pelo usuário, sai do loop

    # Imprime: String texto_input_vetor_A
    addi a7, zero, 4 # Adiciona o valor 4 (PrintString) ao registrador de serviço a7
    la a0, texto_input_Vetor_A # Carrega ao registrador a0 o texto_input_vetor_A
    ecall # Chama a syscall

    # Imprime: Int contador ou i
    addi a7, zero, 1 # Adiciona o valor 1 (PrintInt) ao registrador de serviço a7
    add a0, zero, t0 # Carrega ao registrador a0 o contador
    ecall # Chama a syscall

    # Imprime: String texto_fecha_index
    addi a7, zero, 4 # Adiciona o valor 4 (PrintString) ao registrador de serviço a7
    la a0, texto_fecha_index # Carrega ao registrador a0 o texto_fecha_index
    ecall # Chama a syscall

    # Solicita: Int valor na posição atual

```

```

addi a7, zero, 5 # Adiciona o valor 5 (ReadInt) ao registrador de serviço a7
ecall # Chama a syscall

add s1, zero, a0 # O que foi digitado no console (registrador a0), é salvo no
registrador s0

slli t1, t0, 2 # Move 2 bits para a esquerda: 4 * i
add t2, s10, t1 # Calcula a posicao no Vetor_A desde o seu começo:
começo_do_Vetor + (4 * i)

# Adiciona o valor na posição calculada do Vetor_A
sw s1, 0(t2) # Guarda o valor informado pelo usuário no s1, com um offset de 0
bits, no Vetor_A (s10)

addi t0, t0, 1 # Incrementa o contador: i = i + 1
j loop_preenche_Vetor_A # "Reinicia" o loop

define_Vetor_B_contador:
li t0, 0 # Define o counter para o valor 0
la t1, texto_input_Vetor_B # Carrega o texto texto_input_vetor_B no registrador
t1
j loop_preenche_Vetor_B # Executa a função loop_preenche_Vetor_B

loop_preenche_Vetor_B:
bge t0, s0, troca_valores_vetor # Se o contador foi maior que o tamanho
informado pelo usuário, sai do loop

# Imprime: String texto_input_vetor_B
addi a7, zero, 4 # Adiciona o valor 4 (PrintString) ao registrador de serviço a7
la a0, texto_input_Vetor_B # Carrega ao registrador a0 o texto_input_vetor_B
ecall # Chama a syscall

# Imprime: Int contador ou i
addi a7, zero, 1 # Adiciona o valor 1 (PrintInt) ao registrador de serviço a7
add a0, zero, t0 # Carrega ao registrador a0 o contador
ecall # Chama a syscall

```

```

# Imprime: String texto_fecha_index
addi a7, zero, 4 # Adiciona o valor 4 (PrintString) ao registrador de serviço a7
la a0, texto_fecha_index # Carrega ao registrador a0 o texto_fecha_index
ecall # Chama a syscall

# Solicita: Int valor na posição atual
addi a7, zero, 5 # Adiciona o valor 5 (ReadInt) ao registrador de serviço a7
ecall # Chama a syscall
add s1, zero, a0 # O que foi digitado no console (registrador a0), é salvo no
registrador s0

slli t1, t0, 2 # Move 2 bits para a esquerda: 4 * i
add t2, s11, t1 # Calcula a posicao no Vetor_A desde o seu começo:
começo_do_Vetor + 4 * i

# Adiciona o valor na posição calculada do Vetor_A
sw s1, 0(t2) # Guarda o valor informado pelo usuário no s1, com um offset de 0
bits, no Vetor_B (s11)

addi t0, t0, 1 # Incrementa o contador: i = i + 1
j loop_preenche_Vetor_B # "Reinicia" o loop

troca_valores_vetor:
li t0, 0
j loop_troca_valores_vetor

loop_troca_valores_vetor:
bge t0, s0, print_Vetor_A # Se o contador foi maior que o tamanho informado
pelo usuário, sai do loop

# Calcula a posição no Vetor_A (s10)
slli t1, t0, 2 # Move 2 bits para a esquerda: 4 * i
add t2, s10, t1 # Calcula a posição no Vetor_A desde o seu começo:
começo_do_Vetor + 4 * i

```

```
lw t3, 0(t2) # Carrega no registrador t3 (auxiliar), sem offset de bits, o conteudo  
do Vetor_A na posicao atual
```

```
# Calcula a posicao no Vetor_B (s11)  
slli t1, t0, 2 # Move 2 bits para a esquerda: 4 * i  
add t4, s11, t1 # Calcula a posicao no Vetor_B desde o seu começo:  
começo_do_Vetor + 4 * i
```

```
lw t5, 0(t4) # Carrega no registrador t4, sem offset de bits, o conteudo do  
Vetor_B na posicao atual
```

```
sw t5, 0(t2) # Vetor_B[i] = Vetor_A[i]  
sw t3, 0(t4) # Vetor_A[i] = Vetor_B[i]
```

```
addi t0, t0, 1 # Incrementa o contador: i = i + 1  
j loop_troca_valores_vetor # "Reinicia" o loop
```

```
print_Vetor_A:
```

```
li t0, 0  
j loop_print_Vetor_A
```

```
loop_print_Vetor_A:
```

```
bge t0, s0, print_Vetor_B # Se o contador foi maior que o tamanho informado  
pelo usuário, sai do loop
```

```
slli t1, t0, 2 # Move 2 bits para a esquerda: 4 * i  
add t2, s10, t1 # Calcula a posicao no Vetor_A desde o seu começo:  
começo_do_Vetor + 4 * i
```

```
# Imprime: String texto_input_vetor_A  
addi a7, zero, 4 # Adiciona o valor 4 (PrintString) ao registrador de serviço a7  
la a0, texto_input_Vetor_A # Carrega ao registrador a0 o texto_input_vetor_A  
ecall # Chama a syscall
```

```
# Imprime: Int contador ou i
```

```
addi a7, zero, 1 # Adiciona o valor 1 (PrintInt) ao registrador de serviço a7  
add a0, zero, t0 # Carrega ao registrador a0 o contador  
ecall # Chama a syscall
```

```
# Imprime: String texto_fecha_index  
addi a7, zero, 4 # Adiciona o valor 4 (PrintString) ao registrador de serviço a7  
la a0, texto_fecha_index # Carrega ao registrador a0 o texto_fecha_index  
ecall # Chama a syscall
```

```
# Imprime: Int valor na posição atual  
addi a7, zero, 1 # Adiciona o valor 1 (PrintInt) ao registrador de serviço a7  
lw a0, 0(t2) # Adciona ao registrador a0, o valor daquela posicao do vetor  
ecall # Chama a syscall
```

```
addi t0, t0, 1 # Incrementa o contador: i = i + 1  
j loop_print_Vetor_A # "Reinicia" o loop
```

```
print_Vetor_B:  
    li t0, 0  
    j loop_print_Vetor_B
```

```
loop_print_Vetor_B:  
    bge t0, s0, exit # Se o contador foi maior que o tamanho informado pelo usuario,  
    sai do loop
```

```
slli t1, t0, 2 # Move 2 bits para a esquerda: 4 * i  
add t2, s11, t1 # Calcula a posicao no Vetor_A desde o seu começo:  
começo_do_Vetor + 4 * i
```

```
# Imprime: String texto_input_vetor_A  
addi a7, zero, 4 # Adiciona o valor 4 (PrintString) ao registrador de serviço a7  
la a0, texto_input_Vetor_B # Carrega ao registrador a0 o texto_input_vetor_B  
ecall # Chama a syscall
```

```

# Imprime: Int contador ou i
addi a7, zero, 1 # Adiciona o valor 1 (PrintInt) ao registrador de serviço a7
add a0, zero, t0 # Carrega ao registrador a0 o contador
ecall # Chama a syscall

# Imprime: String texto_fecha_index
addi a7, zero, 4 # Adiciona o valor 4 (PrintString) ao registrador de serviço a7
la a0, texto_fecha_index # Carrega ao registrador a0 o texto_fecha_index
ecall # Chama a syscall

# Imprime: Int valor na posição atual
addi a7, zero, 1 # Adiciona o valor 1 (PrintInt) ao registrador de serviço a7
lw a0, 0(t2) # Adciona ao registrador a0, o valor daquela posicao do vetor
ecall # Chama a syscall

addi t0, t0, 1 # Incrementa o contador: i = i + 1
j loop_print_Vetor_B # "Reinicia" o loop

```

exit:

```

addi a7, zero, 10 # Usa a diretir Exit (10)
ecall

```

2.4 Resultados

Informações da execução:

- Tamanho dos Vetores: 4
- Elementos do Vetor_A em ordem de inserção: 1, 2, 3, 4
- Elementos do Vetor_B em ordem de inserção: 5, 6, 7, 8
- Resultado Final da Operação de troca:
 - Vetor_A: 5, 6, 7, 8
 - Vetor_B: 1, 2, 3, 4

Abaixo são as telas de capturas (Prints) da execução do Programa inserindo os valores informados acima:

The screenshot shows the RARS 1.6 assembly debugger interface with several windows open:

- Registers**: Shows registers r0 to r31, each with a value column.
- Data Segment**: Shows memory starting at address 0x00000000. The first few bytes are 0x48 00 00 00, followed by 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000.
- Text Segment**: Shows assembly code. Key instructions include:
 - lw t0, min_tam
 - lw t1, max_tam
 - la s10, Vetur_A # Carrega o Vetor_A no registrador s10
 - la s11, Vetur_B # Carrega o Vetor_B no registrador s11
 - addi a7, zero, 4 # Adiciona o valor 4 (Print)
 - la a8, texto_input_max_tam # Carrega ao registrador a8
 - ecall # Chama a syscall
 - addi a7, zero, 5 # Adiciona o valor 5 (Read)
 - ecall # Chama a syscall

The screenshot shows the QEMU debugger interface with the following sections:

- Registers**: Shows various registers (r0-r31, sp, fp) with their values.
- Floating Point**: Shows floating-point registers (fp0-fp7) with their values.
- Control and Status**: Shows control and status registers with their values.
- Text Segment**: Shows assembly code with columns for Address, Code, and Bkpt. The code includes instructions like lw, la, addi, and ecall.
- Data Segment**: Shows memory dump with columns for Address, Value (+0), Value (+4), Value (+8), Value (+12), Value (+16), Value (+20), Value (+24), and Value (+28).
- Messages**: Displays the message "Run I/O".
- Registers**: Shows the state of registers r0 through r31.
- Text Segment**: Shows the assembly code for the current segment.
- Data Segment**: Shows the memory dump for the current segment.
- Messages**: Displays the message "Enter com o tamanho dos vetores (max = 8): 4".
- Registers**: Shows the state of registers r0 through r31.
- Text Segment**: Shows the assembly code for the current segment.
- Data Segment**: Shows the memory dump for the current segment.
- Messages**: Displays the message "Vetor_A[0] = 1".
- Registers**: Shows the state of registers r0 through r31.
- Text Segment**: Shows the assembly code for the current segment.
- Data Segment**: Shows the memory dump for the current segment.
- Messages**: Displays the message "Vetor_A[1] = 2".
- Registers**: Shows the state of registers r0 through r31.
- Text Segment**: Shows the assembly code for the current segment.
- Data Segment**: Shows the memory dump for the current segment.
- Messages**: Displays the message "Vetor_A[2] = 3".
- Registers**: Shows the state of registers r0 through r31.
- Text Segment**: Shows the assembly code for the current segment.
- Data Segment**: Shows the memory dump for the current segment.
- Messages**: Displays the message "Vetor_A[3] = 4".
- Registers**: Shows the state of registers r0 through r31.
- Text Segment**: Shows the assembly code for the current segment.
- Data Segment**: Shows the memory dump for the current segment.
- Messages**: Displays the message "Vetor_B[0] = 4".

Abaixo é o resultado da operação de trocar os valores dos vetores um com o outro.

Name	Number	Value
zero	0	0x0000000000000000
r0	1	0x0000000000000008
r1	2	0x0000000000000000
r2	3	0x0000000000000000
r3	4	0x0000000000000000
r4	5	0x0000000000000000
r5	6	0x0000000000000000
r6	7	0x0000000000000000
r7	8	0x0000000000000000
r8	9	0x0000000000000000
r9	10	0x0000000000000000
r10	11	0x0000000000000000
r11	12	0x0000000000000000
r12	13	0x0000000000000000
r13	14	0x0000000000000000
r14	15	0x0000000000000000
r15	16	0x0000000000000000
r16	17	0x0000000000000000
r17	18	0x0000000000000000
r18	19	0x0000000000000000
r19	20	0x0000000000000000
r20	21	0x0000000000000000
r21	22	0x0000000000000000
r22	23	0x0000000000000000
r23	24	0x0000000000000000
r24	25	0x0000000000000000
r25	26	0x0000000000000000
r26	27	0x0000000000000000
r27	28	0x0000000000000000
r28	29	0x0000000000000000
r29	30	0x0000000000000000
r30	31	0x0000000000000000
pc	32	0x0000000000000000

Abaixo é a inclusão do Instruction Statistics no Programa e seus resultados de cada instrução realizada:

Name	Number	Value
zero	0	0x0000000000000000
r0	1	0x0000000000000008
r1	2	0x0000000000000000
r2	3	0x0000000000000000
r3	4	0x0000000000000000
r4	5	0x0000000000000000
r5	6	0x0000000000000000
r6	7	0x0000000000000000
r7	8	0x0000000000000000
r8	9	0x0000000000000000
r9	10	0x0000000000000000
r10	11	0x0000000000000000
r11	12	0x0000000000000000
r12	13	0x0000000000000000
r13	14	0x0000000000000000
r14	15	0x0000000000000000
r15	16	0x0000000000000000
r16	17	0x0000000000000000
r17	18	0x0000000000000000
r18	19	0x0000000000000000
r19	20	0x0000000000000000
r20	21	0x0000000000000000
r21	22	0x0000000000000000
r22	23	0x0000000000000000
r23	24	0x0000000000000000
r24	25	0x0000000000000000
r25	26	0x0000000000000000
r26	27	0x0000000000000000
r27	28	0x0000000000000000
r28	29	0x0000000000000000
r29	30	0x0000000000000000
r30	31	0x0000000000000000
pc	32	0x0000000000000000

3. Programa 02

3.1 Enunciado: Considere uma disciplina com 16 dias de aulas e que é necessário registrar a presença de cada aluno em cada dia, consumindo o mínimo de variáveis em memória. Assumindo que a turma tenha no máximo 32 alunos, implemente um programa que utilize um vetor de 16 elementos inteiros (um elemento para cada dia) para registrar a frequência dos alunos de cada disciplina e que ofereça ao professor funções para registrar a presença e ausência de qualquer aluno em qualquer dia.

3.2 Código fonte em Linguagem de Alto Nível C/C++

// Disciplina : Arquitetura e Organização de Computadores
// Atividade : Avaliação 01 – Programação em Linguagem de Alto nível
// Programa 02
// Grupo : - Mateus Barbosa
// - Matheus de Oliveira Rocha

```
using namespace std;
```

```
#include <iostream>
```

```
#include <string>
```

```
#define MAX_AULA 15
```

```
#define MAX_ALUNO 31
```

```
#define MAX_REGISTRO 1
```

```
int main()
```

```
{
```

```
    string Vetor_Dias[MAX_AULA]; // Por simplicidade, utilizamos uma string  
contendo os 32 bits dos alunos
```

```
// Preenche todos os dias com todos os alunos presentes
```

```
for (int i = 0; i < MAX_AULA; i++)
```

```
{
```

```
    Vetor_Dias[i] = "11111111111111111111111111111111";
```

```
}
```

```
while (true)
```

```
{
```

```
    // Solicita o número da aula
```

```
    int input_aula;
```

```
    do
```

```
{
```

```
    cout << "\nEntre com o número da aula (de 0 a 15): ";
```

```

    cin >> input_aula;
    if (!cin.good()) // Sai deste loop se o input não for número
        break;
    else if (input_aula < 0 || input_aula > MAX_AULA)
        cout
            << "\nValor inválido";
    } while (input_aula < 0 || input_aula > MAX_AULA);

    if (!cin.good()) // Sai do loop principal se o input não for número
        break;

    // Solicita o número do aluno
    int input_aluno;
    do
    {
        cout << "\nEnter com o número do aluno (de 0 a 31): ";
        cin >> input_aluno;
        if (!cin.good()) // Sai deste loop se o input não for número
            break;
        else if (input_aluno < 0 || input_aluno > MAX_ALUNO)
            cout << "\nValor inválido";
    } while (input_aluno < 0 || input_aluno > MAX_ALUNO);

    if (!cin.good()) // Sai do loop principal se o input não for número
        break;

    // Solicita o tipo do registro
    int input_registro;
    do
    {
        cout << "\nEnter com o tipo do registro (presença = 1; ausência = 0): ";
        cin >> input_registro;
        if (!cin.good()) // Sai deste loop se o input não for número
            break;

```

```

    else if (input_registro < 0 || input_registro > MAX_REGISTRO)
        cout
            << "\nValor inválido";
    } while (input_registro < 0 || input_registro > MAX_REGISTRO);

    if (!cin.good()) // Sai do loop principal se o input não for número
        break;

    Vetor_Dias[input_aula][(MAX_ALUNO - 1) - input_aluno] =
        to_string(input_registro)[0]; // Para simular que as string são bits, inserimos da direita
        para a esquerda, isso é alcançado usando a formula (MAX_ALUNO - 1) - input_aluno
    }

// O código abaixo simularia o campo Data Segment na aba Execute do RARS
cout << "\nData Segment:\n";
for (int i = 0; i < MAX_AULA; i++)
{
    cout << i << "\t=> " << Vetor_Dias[i] << endl;
}

return 0;
}

```

3.3 Código fonte em Linguagem de Montagem do Risc-V

Explicação da Lógica do Código:

O programa utiliza uma sequência de strings formatadas e chamadas de sistema para receber as entradas do usuário (aula, aluno e tipo de registro) e validar as entradas fornecidas.

Na seção ".data", são definidos os dados, incluindo o vetor de dias (Vetor_Dias) que guarda o registro de presença/falta de cada aluno em cada aula, uma máscara de bits (mask) que será utilizada posteriormente e várias strings formatadas que serão utilizadas nas chamadas de sistema.

Na seção ".text", a função "main" é definida como ponto de partida para o programa, que então chama a função "loop_define_aula". Dentro dessa função, o programa exibe a string "texto_input_aula" e recebe um valor inteiro do usuário que corresponde ao número da aula. Em seguida, o programa verifica se o valor fornecido é maior ou igual a zero e menor ou igual a "max_aula". Se for, ele passa para a função

"loop_define_aluno", caso contrário, exibe a string "texto_valor_invalido" e chama a função "loop_define_aula" novamente.

A função "loop_define_aluno" é semelhante à função "loop_define_aula", exibindo a string "texto_input_aluno" e recebendo um valor inteiro do usuário que corresponde ao número do aluno. Em seguida, o programa verifica se o valor fornecido é maior ou igual a zero e menor ou igual a "max_aluno". Se for, ele passa para a função "loop_define_registro", caso contrário, exibe a string "texto_valor_invalido" e chama a função "loop_define_aluno" novamente.

A função "loop_define_registro" é novamente semelhante às funções anteriores, exibindo a string "texto_input_registro" e recebendo um valor inteiro do usuário que corresponde ao tipo de registro (1 para presença e 0 para falta). Em seguida, o programa verifica se o valor fornecido é maior ou igual a zero e menor ou igual a "max_registro". Se for, ele usa a máscara de bits (mask) para definir o registro de presença/falta do aluno na aula correspondente, atualizando o vetor de dias (Vetor_Dias), e em seguida retorna à função "loop_define_aula", possibilitando a re-execução do programa, mas mantendo os valores informados anteriormente. Caso contrário, exibe a string "texto_valor_invalido" e chama a função "loop_define_registro" novamente.

Código do Risc-V:

```
# Disciplina: Arquitetura e Organização de Computadores
# Atividade: Avaliação 01 – Programação em Linguagem de Montagem
# Programa 02
# Grupo: - Mateus Barbosa
#           - Matheus de Oliveira Rocha

.data # Segmento de Dados
    Vetor_Dias: .word 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF,
0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF,
0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF,
0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF
    mask: .word 0x00000001
    texto_input_aula: .asciz "\nEntre com o número da aula (de 0 a 15): "
    texto_input_aluno: .asciz "\nEntre com o número do aluno (de 0 a 31):"
    texto_input_registro: .asciz "\nEntre com o tipo do registro (presença = 1;
ausência = 0): "
    texto_valor_invalido: .asciz "\nValor Inválido"
    max_aula: .word 15
    max_aluno: .word 31
    max_registro: .word 1
```

```
.text # Segmento de Código
```

```
main: # Função usada como ponto de partida do código, e retorno de loops
```

```
    lw t0, mask # Carrega a máscara de bits no registrador t0
```

```
    la s0, Vetor_Dias # Carrega o Vetor_dias no registrador s0
```

```
    j loop_define_aula
```

```
loop_define_aula:
```

```
    # Imprime: String texto_input_aula
```

```
    addi a7, zero, 4 # Adiciona o valor 4 (PrintString) ao registrador de serviço a7
```

```
    la a0, texto_input_aula # Carrega ao registrador a0 o texto_input_max_tam
```

```
    ecall # Chama a syscall
```

```
# Solicita: Int numero da aula
```

```
    addi a7, zero, 5 # Adiciona o valor 5 (ReadInt) ao registrador de serviço a7
```

```
    ecall # Chama a syscall
```

```
    add s1, zero, a0 # O que foi digitado no console (registrador a0), é salvo no registrador s1
```

```
    bge s1, zero, if_aula_maior_que # Se o valor de s1 for maior ou igual que 0, vai para a função if_maior_que
```

```
    j loop_define_aula # "Pula" para a função loop_define_aula
```

```
if_aula_maior_que:
```

```
    lw t1, max_aula
```

```
    ble s1, t1, loop_define_aluno # Se o valor de s1 for menor ou igual que max_aluno, vai para a função loop_define_aluno
```

```
# Imprime: String valor_invalido
```

```
    addi a7, zero, 4 # Adiciona o valor 4 (PrintString) ao registrador de serviço a7
```

```
    la a0, texto_valor_invalido # Carrega ao registrador a0 o valor_invalido
```

```
    ecall # Chama a syscall
```

```
j loop_define_aula # "Pula" para a função loop_define_aluno

loop_define_aluno:
    # Imprime: String texto_input_aluno
    addi a7, zero, 4 # Adiciona o valor 4 (PrintString) ao registrador de serviço a7
    la a0, texto_input_aluno # Carrega ao registrador a0 o texto_input_max_tam
    ecall # Chama a syscall

    # Solicita: Int numero da aluno
    addi a7, zero, 5 # Adiciona o valor 5 (ReadInt) ao registrador de serviço a7
    ecall # Chama a syscall
    add s2, zero, a0 # O que foi digitado no console (registrador a0), é salvo no
registrador s2
```

bge s2, zero, if_aluno_maior_que # Se o valor de s2 for maior ou igual que 0,
vai para a função if_aluno_maior_que

```
j loop_define_aluno # "Pula" para a função loop_define_aluno

if_aluno_maior_que:
    lw t1, max_aluno
    ble s2, t1, loop_define_registro # Se o valor de s2 for menor ou igual que
max_aluno, vai para a função loop_define_registro

    # Imprime: String valor_invalido
    addi a7, zero, 4 # Adiciona o valor 4 (PrintString) ao registrador de serviço a7
    la a0, texto_valor_invalido # Carrega ao registrador a0 o valor_invalido
    ecall # Chama a syscall
```

```
j loop_define_aluno # "Pula" para a função loop_define_aluno

loop_define_registro:
    # Imprime: String texto_input_aluno
    addi a7, zero, 4 # Adiciona o valor 4 (PrintString) ao registrador de serviço a7
    la a0, texto_input_registro # Carrega ao registrador a0 o texto_input_registro
```

```

        ecall # Chama a syscall

# Solicita: Int numero da aluno
addi a7, zero, 5 # Adiciona o valor 5 (ReadInt) ao registrador de serviço a7
ecall # Chama a syscall
add s3, zero, a0 # O que foi digitado no console (registrador a0), é salvo no
registrador s3

bge s3, zero, if_registro_maior_que # Se o valor de s3 for maior ou igual que 0,
vai para a função if_registro_maior_que

j loop_define_registro # "Pula" para a função loop_define_presenca

if_registro_maior_que:
    lw t1, max_registro
    ble s3, t1, presenca_ou_ausencia # Se o valor de s3 for menor ou igual que
max_registro, vai para a função presenca_ou_ausencia

# Imprime: String valor_invalido
addi a7, zero, 4 # Adiciona o valor 4 (PrintString) ao registrador de serviço a7
la a0, texto_valor_invalido # Carrega ao registrador a0 o valor_invalido
ecall # Chama a syscall

j loop_define_registro # "Pula" para a função loop_define_presenca

presenca_ou_ausencia:
    sll t2, t0, s2 # Desloca para a mascara (t0) esquera de acordo com o numero do
aluno (s2)

    slli t3, s1, 2 # Move 2 bits para a esquerda de acordo com o dia: 4 * dia
    add t4, s0, t3 # Calcula a posicao no Vetor_Dias desde o seu começo:
comeco_do_Vetor + (4 * dia)

    lw t5, 0(t4) # Armazena em t5 o valor dos bits do dia informado (t4) com offset
de 0 bits

```

```
beq s3, zero, ausencia # Se for 0, esta ausente  
beq s3, t1, presenca # Se for 1 (max_registro), esta presente  
j exit # Tratamento de error, nunca devera cair nesse jump, devido as validacoes  
anteriores
```

ausencia:

```
not t2, t2 #Inverte a mascara  
and t6, t5, t2 # Altera o bit do aluno usando a mascara (t2) usando and  
sw t6, 0(t4) # Armazena os bites com a alteracao da mascara na posicao do  
Vetor_Dias
```

```
j main
```

presenca:

```
or t6, t5, t2 # Altera o bit do aluno usando a mascara (t2) usando or  
sw t6, 0(t4) # Armazena os bites com a alteracao da mascara na posicao do  
Vetor_Dias
```

```
j main
```

exit:

```
addi a7, zero, 10 # Usa a diretriz Exit (10)  
ecall
```

2.4 resultados

Informações da execução:

- Apenas após sair do programa que será possível visualizar as informações armazenadas no Vetor
- Na primeira execução, informamos que os seguintes alunos (5, 12, 24 e 31) não estão presentes no dia 8.
- Na segunda execução, informamos que no dia 1, o aluno 0 será inicialmente informado como ausente e depois retonará como presente. E o aluno 1 estará ausente.
- Para sair do programa, digite um valor diferente de um inteiro, uma letra por exemplo.

Inserção de informação do aluno 5, 12 e 24:

The screenshot shows the RARS 1.6 debugger interface with the assembly code for inserting student information. The code includes several `addi` and `ecall` instructions to manipulate memory and perform system calls.

```

    .text
    .segment Text Segment
    .bupt
    .address 0x00000000
    .code
    .basic
    .source
    .text
        .bupt
        .address 0x00000000
        .code
        .basic
        .source
        .text
            .bupt
            .address 0x00000000
            .code
            .basic
            .source
            .text
                .bupt
                .address 0x00000000
                .code
                .basic
                .source
                .text
                    .bupt
                    .address 0x00000000
                    .code
                    .basic
                    .source
                    .text
                        .bupt
                        .address 0x00000000
                        .code
                        .basic
                        .source
                        .text
                            .bupt
                            .address 0x00000000
                            .code
                            .basic
                            .source
                            .text
                                .bupt
                                .address 0x00000000
                                .code
                                .basic
                                .source
                                .text
                                    .bupt
                                    .address 0x00000000
                                    .code
                                    .basic
                                    .source
                                    .text
                                        .bupt
                                        .address 0x00000000
                                        .code
                                        .basic
                                        .source
                                        .text
                                            .bupt
                                            .address 0x00000000
                                            .code
                                            .basic
                                            .source
                                            .text
                                                .bupt
                                                .address 0x00000000
                                                .code
                                                .basic
                                                .source
                                                .text
                                                    .bupt
                                                    .address 0x00000000
                                                    .code
                                                    .basic
                                                    .source
                                                    .text
                                                        .bupt
                                                        .address 0x00000000
                                                        .code
                                                        .basic
                                                        .source
                                                        .text
                                                            .bupt
                                                            .address 0x00000000
                                                            .code
                                                            .basic
                                                            .source
                                                            .text
                                                                .bupt
                                                                .address 0x00000000
                                                                .code
                                                                .basic
                                                                .source
                                                                .text
                                                                    .bupt
                                                                    .address 0x00000000
                                                                    .code
                                                                    .basic
                                                                    .source
                                                                    .text
                                                                        .bupt
                                                                        .address 0x00000000
                                                                        .code
                                                                        .basic
                                                                        .source
                                                                        .text
                                                                            .bupt
                                                                            .address 0x00000000
                                                                            .code
                                                                            .basic
                                                                            .source
                                                                            .text
                                                                                .bupt
                                                                                .address 0x00000000
                                                                                .code
                                                                                .basic
                                                                                .source
                                                                                .text
                                                                                    .bupt
                                                                                    .address 0x00000000
                                                                                    .code
                                                                                    .basic
                                                                                    .source
                                                                                    .text
                                                                                        .bupt
                                                                                        .address 0x00000000
                                                                                        .code
                                                                                        .basic
                                                                                        .source
                                                                                        .text
                                                                                            .bupt
                                                                                            .address 0x00000000
                                                                                            .code
                                                                                            .basic
                                                                                            .source
                                                                                            .text
                                                                                                .bupt
                                                                                                .address 0x00000000
                                                                                                .code
                                                                                                .basic
                                                                                                .source
                                                                                                .text
                                                                ................................................................

```

The assembly code includes several `addi` and `ecall` instructions to manipulate memory and perform system calls. The `addi` instructions are used to add values to memory locations, while the `ecall` instructions are used to call system functions. The code is designed to insert information for students 5, 12, and 24.

Inserção de informação do aluno 31:

The screenshot shows the RARS 1.6 debugger interface with the assembly code for inserting student information for student 31. The code follows a similar pattern to the previous one, using `addi` and `ecall` instructions to manipulate memory and perform system calls.

```

    .text
    .segment Text Segment
    .bupt
    .address 0x00000000
    .code
    .basic
    .source
    .text
        .bupt
        .address 0x00000000
        .code
        .basic
        .source
        .text
            .bupt
            .address 0x00000000
            .code
            .basic
            .source
            .text
                .bupt
                .address 0x00000000
                .code
                .basic
                .source
                .text
                    .bupt
                    .address 0x00000000
                    .code
                    .basic
                    .source
                    .text
                        .bupt
                        .address 0x00000000
                        .code
                        .basic
                        .source
                        .text
                            .bupt
                            .address 0x00000000
                            .code
                            .basic
                            .source
                            .text
                                .bupt
                                .address 0x00000000
                                .code
                                .basic
                                .source
                                .text
                                    .bupt
                                    .address 0x00000000
                                    .code
                                    .basic
                                    .source
                                    .text
                                        .bupt
                                        .address 0x00000000
                                        .code
                                        .basic
                                        .source
                                        .text
                                            .bupt
                                            .address 0x00000000
                                            .code
                                            .basic
                                            .source
                                            .text
                                                .bupt
                                                .address 0x00000000
                                                .code
                                                .basic
                                                .source
                                                .text
                                                    .bupt
                                                    .address 0x00000000
                                                    .code
                                                    .basic
                                                    .source
                                                    .text
                                                        .bupt
                                                        .address 0x00000000
                                                        .code
                                                        .basic
                                                        .source
                                                        .text
                                                            .bupt
                                                            .address 0x00000000
                                                            .code
                                                            .basic
                                                            .source
                                                            .text
                                                                .bupt
                                                                .address 0x00000000
                                                                .code
                                                                .basic
                                                                .source
                                                                .text
                                                                    .bupt
                                                                    .address 0x00000000
                                                                    .code
                                                                    .basic
                                                                    .source
                                                                    .text
................................................................

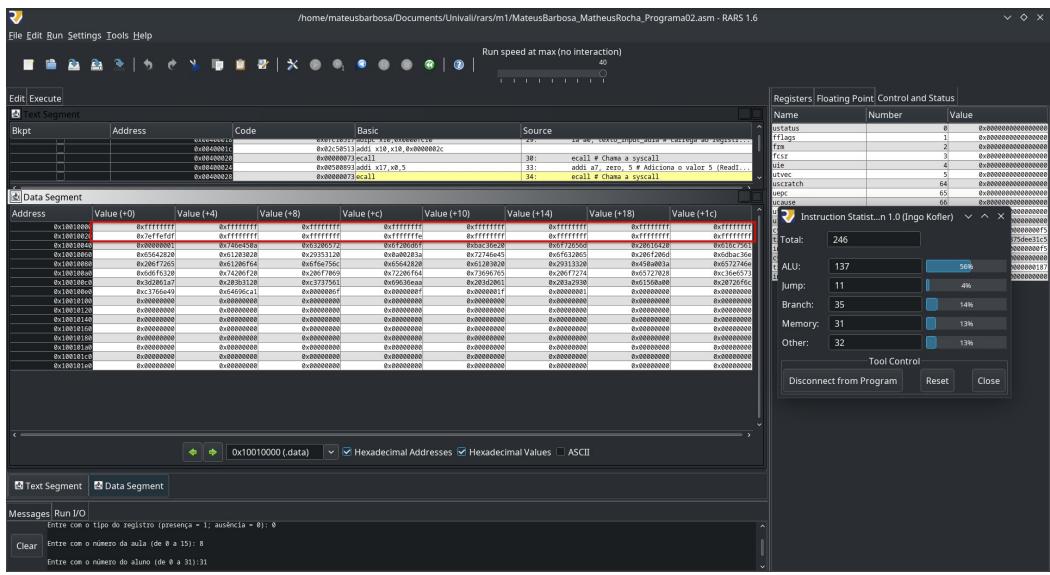
```

The assembly code uses `addi` and `ecall` instructions to insert information for student 31. The `addi` instructions add values to memory locations, and the `ecall` instructions call system functions.

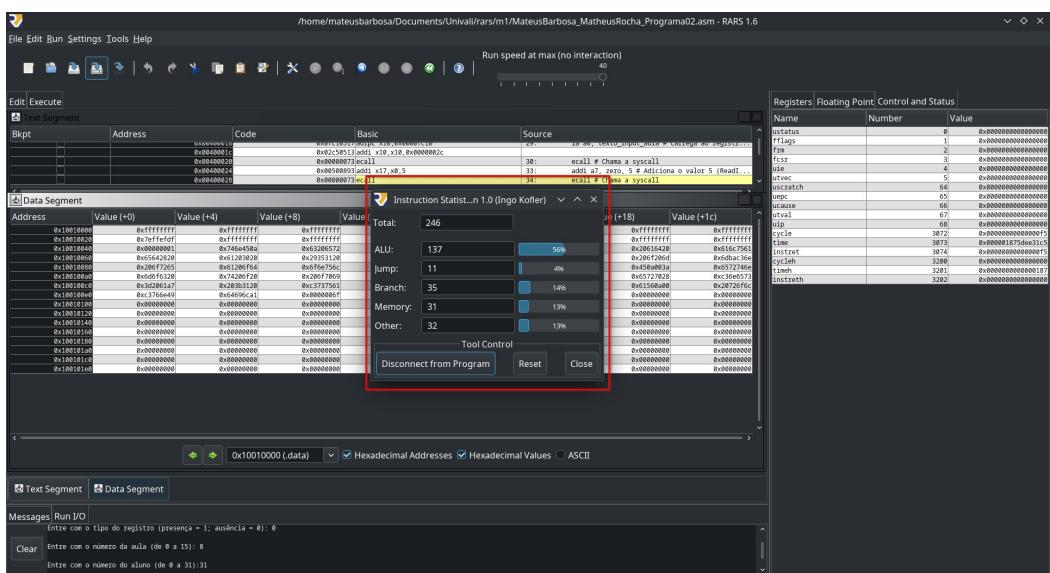
Valor final do Vetor_Dias:

- Valor do Vetor_Dias na posição 8:
 - o Em Hexadecimal: 0x7effefdf
 - o Em Binário: 0111 1110 1111 1111 1110 1111 1101 1111

Como pode-se visualizar nos binários acima, os alunos de número 5, 12, 24 e 31 estão ausentes (com o valor 0).



Abaixo é a inclusão do Instruction Statistics no Programa e seus resultados de cada instrução realizada anteriormente:



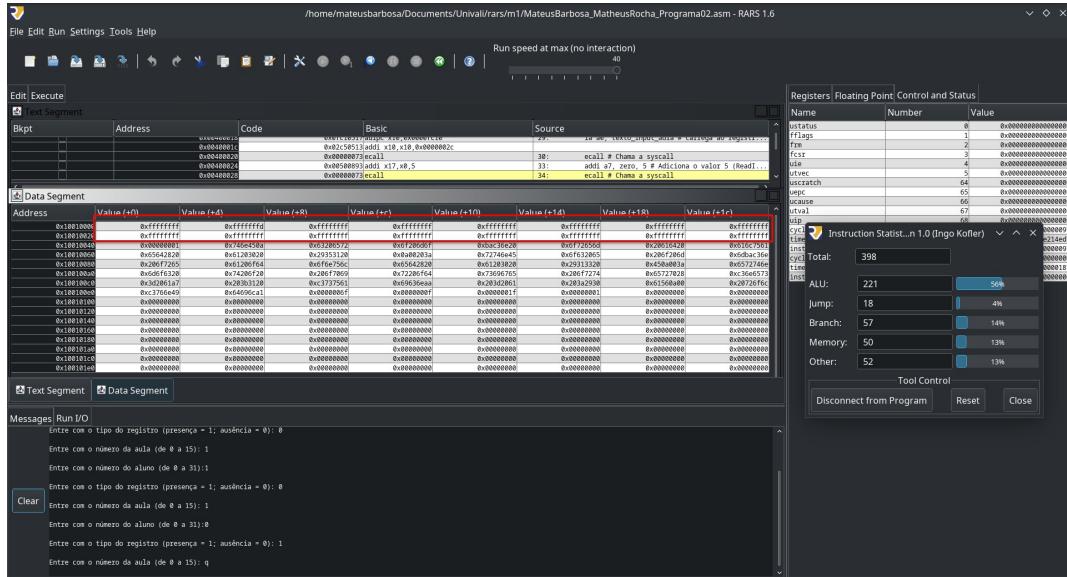
Abaixo será a execução do programa definido no dia 1 que aluno de número 0 estava ausente e se tornou presente, e o aluno 1 se manterá ausente:

The screenshot displays two instances of the RARS 1.6 debugger interface. Both instances show the same assembly code and register values, indicating a step-by-step execution. The assembly code includes instructions for reading from port 0x80000000, performing additions (addl), and calling syscalls. The Registers window shows the state of various CPU registers. The Instruction Statistics window provides a detailed analysis of the instruction types. The Messages window shows the interaction between the user and the program, including input commands and the program's response.

Valor final do Vetor_Dias:

- Valor do Vetor_Dias na posição 1:
 - o Em Hexadecimal: 0xfffffffffd
 - o Em Binário: 11111111 11111111 11111111 11111101

E nesse segundo teste, podemos avaliar que o aluno número zero se tornou presente, e o aluno de número 1 continua ausente nesse dia.



Abaixo é a inclusão do Instruction Statistics no Programa e seus resultados de cada instrução realizada anteriormente:

