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 03 – Programação em linguagem de montagem

Thiago Felski Pereira

29/05/2023

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++. E finalmente fazemos um comparativo mais a fundo dos códigos buscando responder qual será o mais eficiente.

2. Programa 01

2.1 Enunciado: Utilizando a linguagem de montagem do RISC-V, implemente um procedimento que determine a soma dos elementos de um vetor de zero até a posição passada por parâmetro.

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

```
// Disciplina : Arquitetura e Organização de Computadores
// Atividade : Avaliação 03 – Programação em Linguagem de Alto nível
// Programa 01
// Grupo : - Mateus Barbosa
//         - Matheus de Oliveira Rocha

#include <iostream>
using namespace std;

int soma_vet(int vet[], int pos)
{
   int s = 0;
   for (int i = 0; i < pos; i++)
   {
      s = s + vet[i];
   }
   return s;
}

int main()
{
   int pos;
   do
   {
```

```
    cout << "Informe o número de posições do vetor: ";
    cin >> pos;
  } while (pos < 2 || pos > 100);

  int vet[pos];
  for (int i = 0; i < pos; i++)
  {
    vet[i] = i;
  }
  cout << soma_vet(vet, pos);

  return 0;
}
```

2.3 Código fonte em Linguagem de Montagem do Risc-V

Explicação da Lógica do Código:

O programa interage com o usuário para solicitar uma posição do vetor, preencher o vetor com valores crescentes até essa posição e, em seguida, calcular e exibir a soma dos elementos do vetor até essa posição.

O programa começa definindo algumas variáveis e strings. As variáveis "max_tam" e "min_tam" são definidas como palavras (word) e armazenam os valores máximos e mínimos permitidos para a posição do vetor. A variável "vetor" é definida como uma palavra vazia para reservar um endereço de memória para o vetor. As strings "texto_input", "texto_resultado" e "texto_valor_invalido" armazenam mensagens a serem exibidas durante a interação com o usuário.

A função principal do programa é "main". Ela começa chamando a função "solicita_input_tam_vetor", que solicita ao usuário que insira a posição do vetor. O valor de retorno dessa função é armazenado em um registrador (s0).

Em seguida, uma mensagem de resultado é exibida ao usuário usando a syscall "PrintString". A string "texto_resultado" é passada como argumento para a syscall.

Em seguida, o endereço base do vetor é carregado em um registrador (a0) e o número de posições do vetor é definido como o valor obtido anteriormente (s0). Duas funções são chamadas a seguir: "init_vetor" e "soma_vetor".

A função "init_vetor" é responsável por preencher o vetor com valores crescentes até a posição informada pelo usuário. Primeiro, verifica-se se o número de posições está dentro do intervalo permitido. Em seguida, o endereço base do vetor e o número de posições são inicializados em registradores (s0 e s1). Um loop é iniciado para percorrer todas as posições até o valor informado. Dentro do loop, a posição atual

no vetor é calculada multiplicando o índice por 4 (cada elemento ocupa 4 bytes na memória). O valor de índice é então armazenado nessa posição. O índice é incrementado a cada iteração e o loop continua até que o número de posições seja alcançado.

Após a execução da função "init_vetor", a função "soma_vetor" é chamada. Essa função calcula a soma dos elementos do vetor até a posição informada pelo usuário. O endereço base do vetor, o número de posições e um registrador para a soma são inicializados. Um loop é iniciado para percorrer todas as posições até o valor informado. Dentro do loop, a posição atual no vetor é calculada multiplicando o índice por 4. O valor nessa posição é carregado em um registrador e adicionado ao total da soma. O índice é incrementado a cada iteração e o loop continua até que o número de posições seja alcançado.

Finalmente depois de executar a função "soma_vetor", o resultado da soma é exibido na tela usando a syscall "PrintInt".

Código do Risc-V:

```
# Disciplina: Arquitetura e Organização de Computadores
# Atividade: Avaliação 03 – Programação de Procedimentos
# Programa 01
# Grupo:
#       - Mateus Barbosa
#       - Matheus de Oliveira Rocha


.data # Segmento de Dados
        max_tam: .word 100
        min_tam: .word 2
        vetor: .word # Vetor vazio para definir um endereço na memoria
        texto_input: .asciz "\nInforme a posicao do vetor [2-100]: "
        texto_resultado: .asciz "\nResultado da Soma: "
        texto_valor_invalido: .asciz "\nValor inválido!"


.text # Segmento de Código
        jal zero, main # Executa primeiro a funcao main


### Start: Solicita input entre min_tam e max_tam
solicita_input_tam_vetor:
        # Imprime: String texto_input_max_tam
```

```
    addi a7, zero, 4 # Adiciona o valor 4 (PrintString) ao registrador de serviço a7
    la a0, texto_input # Carrega ao registrador a0 o texto_input
    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

    lw t0, min_tam
    bge s0, t0, if_maior_que # Se o valor de s0 for maior que min_tam, 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 solicita_input_tam_vetor # Volta para o inicio de solicita_input_tam_vetor

if_maior_que: # Função que verifica se o valor informado é menor que tam
    lw t1, max_tam
    ble s0, t1, ret_solicita_input_tam_vetor # Se o valor de s0 (sem sinal) for menor que max_tam, vai para a função ret_solicita_input_tam_vetor

    # 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 solicita_input_tam_vetor # Volta para o inicio de solicita_input_tam_vetor

ret_solicita_input_tam_vetor:
    add a0, zero, s0
```

```
        jalr ra # Retorna para o chamador
### End: Solicita input entre min_tam e max_tam

### Start: Cria o vetor ate a posicao informada
init_vetor: # Dinamicamente preenche o vetor ate a posicao informada
        # Salvando registradores na pilha - Push
        addi sp, sp, -12
        sw s0, 0(sp)
        sw s1, 4(sp)
        sw ra, 8(sp)

        lw s0, max_tam # Verificando se o número de posições do vetor é maior que max_tam
        lw s1, min_tam # Verificando se o número de posições do vetor é menor que min_tam
        bge a1, s0, init_vetor_if_maior_que_max_tam
        blt a1, s1, init_vetor_if_menor_que_min_tam
        j posicao_init_vetor_ok

init_vetor_if_maior_que_max_tam:
        lw a1, max_tam # Caso o número de posições seja maior que max_tam, definimos como max_tam
        j posicao_init_vetor_ok

init_vetor_if_menor_que_min_tam:
        lw a1, min_tam # Caso o número de posições seja maior que min_tam, definimos como min_tam
        j posicao_init_vetor_ok

posicao_init_vetor_ok: # Input do usuario OK
        add s0, zero, a0 # Inicializando s0 com o endereço base do vetor
        add s1, zero, a1 # Inicializando s1 com o número de posições do vetor

        li t0, 0 # Inicializando variável i com 0
        j init_vetor_loop # Inicia o loop
```

```
init_vetor_loop:
        bge t0, s1, init_vetor_fim # Verificando se o número de posições do vetor foi alcançado

        slli t1, t0, 2 # Move 2 bits para a esquerda: 4 * i
        add t2, s0, t1 # Calcula a posicao no Vetor_A desde o seu comeco: comeco_do_Vetor + (4 * i)

        sw t0, 0(t2) # Preenchendo a posição atual do vetor com o valor de i

        addi t0, t0, 1 # Incrementando i

        j init_vetor_loop # Retornando ao início do loop

init_vetor_fim:
        # Retirando registradores da Pilha - Pop
        lw s0, 0(sp)
        lw s1, 4(sp)
        lw ra, 8(sp)
        addi sp, sp, 12

        jalr ra # Retorna pro chamador
### End: Cria o vetor ate a posicao informada

### Start: Soma elementos do vetor ate a posicao informada
soma_vetor:
        # PS: Armazenamos os registradores de tipo S pois usamos eles no loop
        # Salvando registradores na pilha - Push
        addi sp, sp, -16
        sw s0, 0(sp)
        sw s1, 4(sp)
        sw s2, 8(sp)
        sw ra, 12(sp)
```

```
    add s0, zero, a0 # Inicializando s0 com o endereço base do vetor
    add s1, zero, a1 # Inicializando s1 com o número de posicoes do vetor
    add s2, zero, zero # Inicializando s2 (total da soma) em zero

    li t0, 0 # Inicializando variável i com 0
    j soma_vetor_loop # Inicia o loop

soma_vetor_loop:
    bge t0, s1, soma_vetor_fim # Verificando se o número de posições do vetor foi alcançado

    slli t1, t0, 2 # Move 2 bits para a esquerda: 4 * i
    add t2, s0, t1 # Calcula a posicao no Vetor_A desde o seu comeco: comeco_do_Vetor + (4 * i)
    lw t2, 0(t2) # Valor do vetor na posicao i

    add s2, s2, t2 # Somando a posição atual do vetor com o valor de i (total += i)

    addi t0, t0, 1 # Incrementando i

    j soma_vetor_loop # Retornando ao início do loop

soma_vetor_fim:
    # Define o valor de retorno
    add a0, zero, s2

    # Retirando registradores da Pilha - Pop
    lw s0, 0(sp)
    lw s1, 4(sp)
    lw s2, 8(sp)
    lw ra, 12(sp)
    addi sp, sp, 16
```

```
        jalr ra, 0 # Retorna pro chamador
### End: Soma elementos do vetor ate a posicao informada

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

main:
        jal ra, solicita_input_tam_vetor # Chama a funcao para solicitar posicao do vetor
        add s0, zero, a0 # O que foi digitado no console (registrador a0), é salvo no registrador s0

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

        # Argumentos:
        #   a0 - endereço base do vetor
        #   a1 - número de posições do vetor
        # O vetor vai de 0 ate o 99 (max_tam - 1) para os valores dos elementos
        la a0, vetor # Carrega o endereco de memoria do vetor em a0
        add a1, zero, s0 # Define o tamanho do vetor e a posicao (index)
        jal ra, init_vetor # Chama init_vetor e define ra para a proxima linha

        # Argumentos:
        #   a0 - endereço base do vetor
        #   a1 - número de posições do vetor
        jal ra, soma_vetor # Chama soma_vetor e define ra para a proxima linha

        # Imprime: Int resultado da soma
        addi a7, zero, 1 # Adiciona o valor 1 (PrintInt) ao registrador de serviço a7
        ecall # Chama a syscall
```

j exit # Sai do programa

2.4 Resultados

Informações da execução:

- Tamanho do Vetor: 52
- Tamanho do Vetor, após informar 5 valores inválidos: 100
- O último valor do vetor será sempre o max_tam - 1
- Resultado Final da Operação:
  - Vetor com tamanho 52: 1326
  - Vetor com tamanho 100: 4950

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

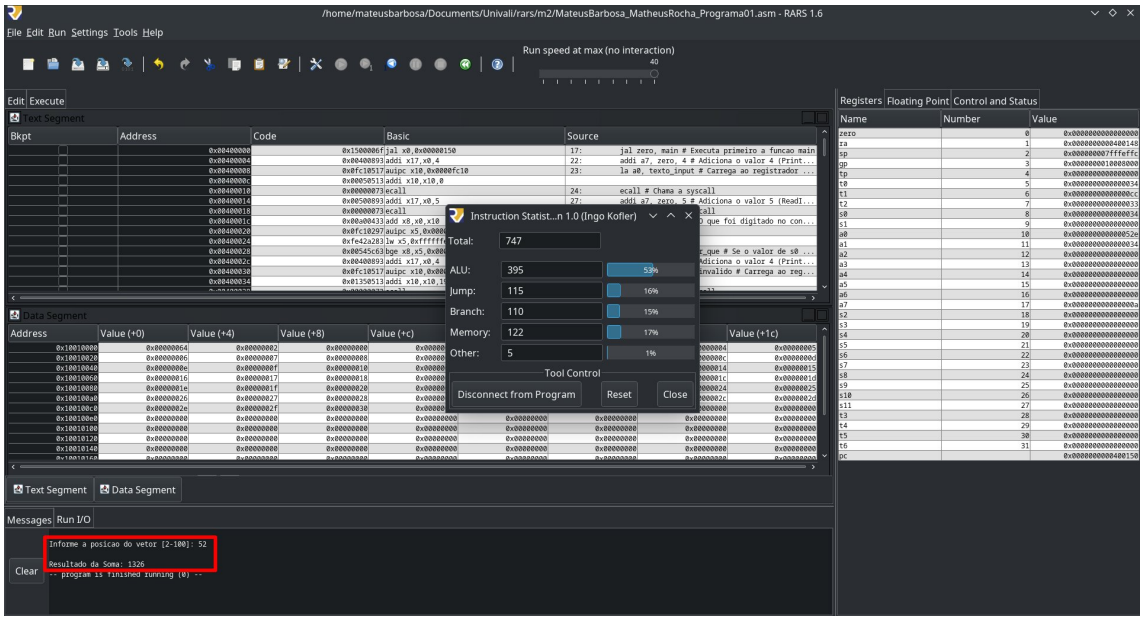

Figura 1: Vetor de Tamanho 52

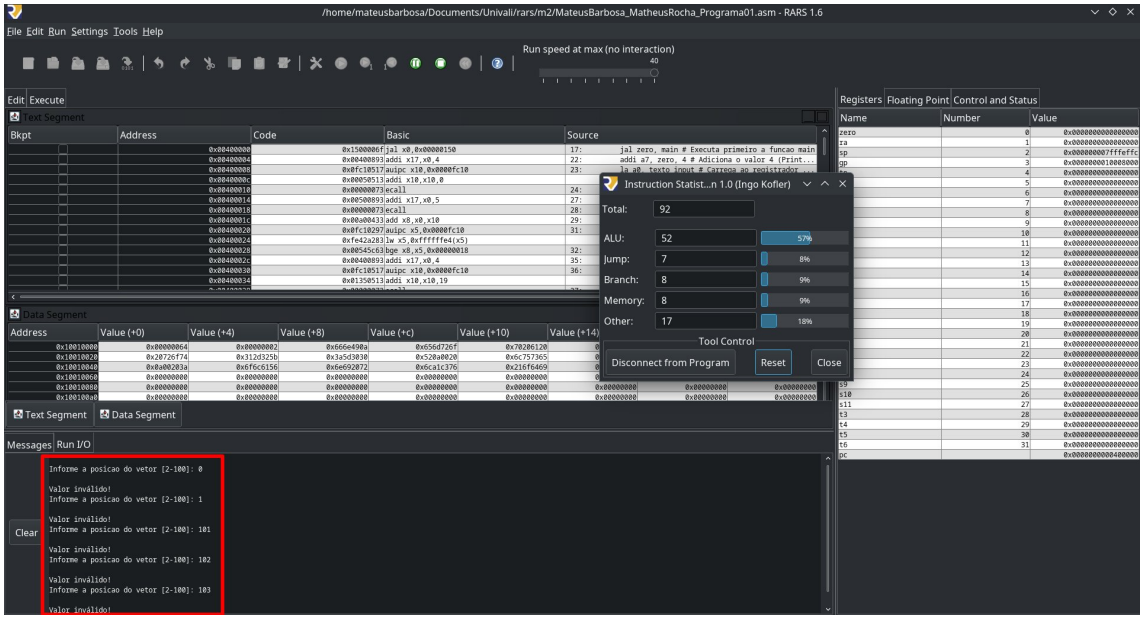

Figura 2.1: Vetor de Tamanho 100, informando 5 tamanhos inválidos

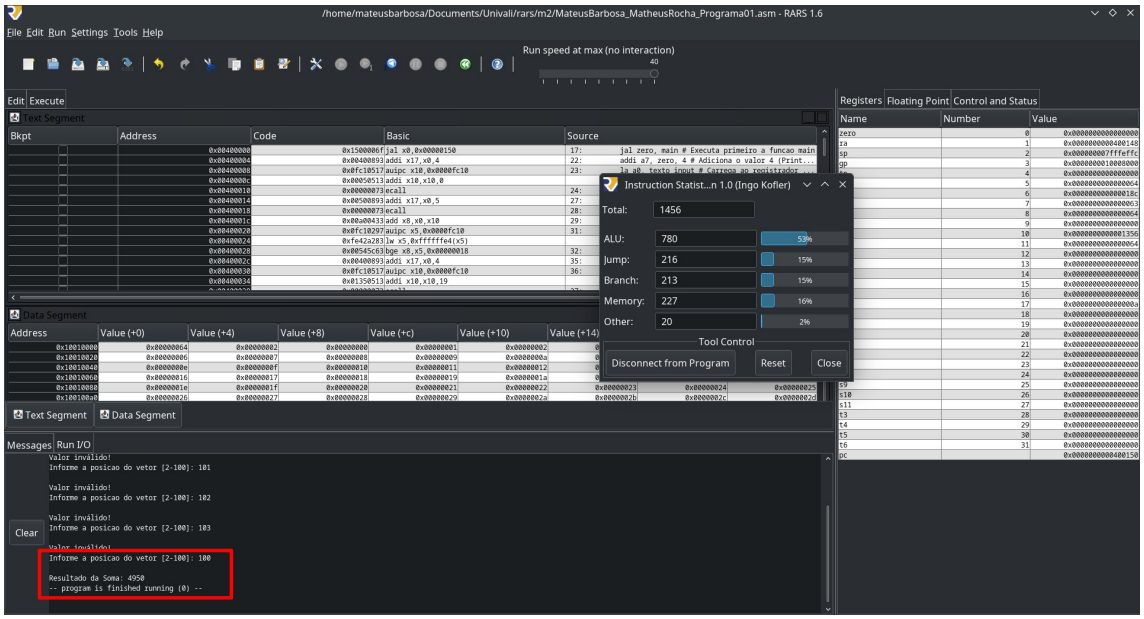

Figura 2.2: Vetor de Tamanho 100, informando o tamanho válido

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

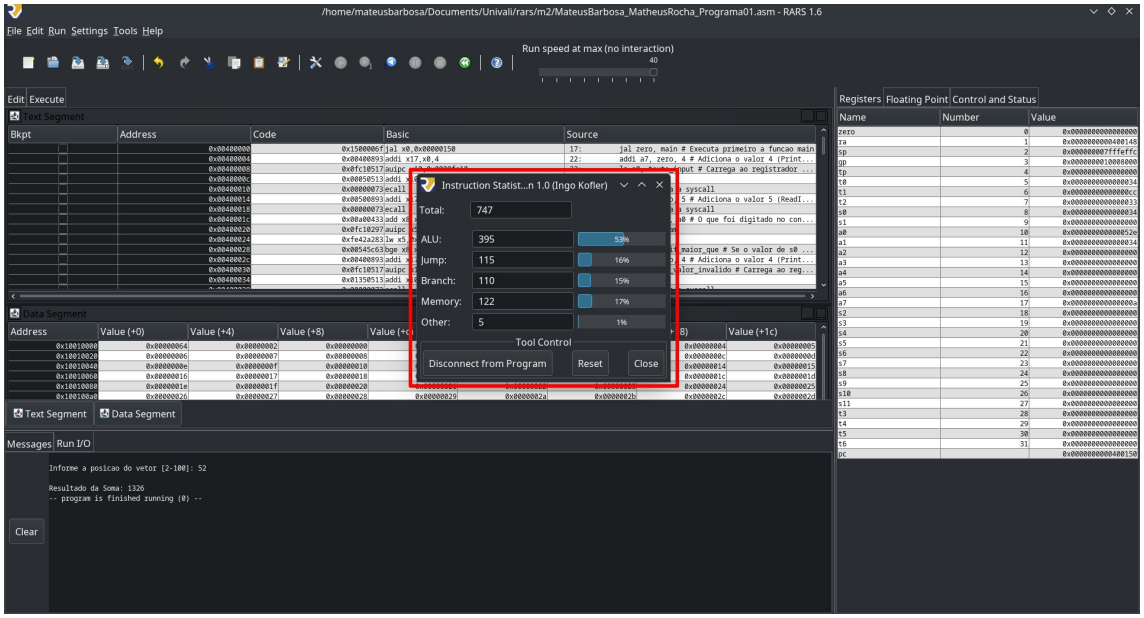

Figura 3: Instruction Statistics do Vetor de Tamanho 52

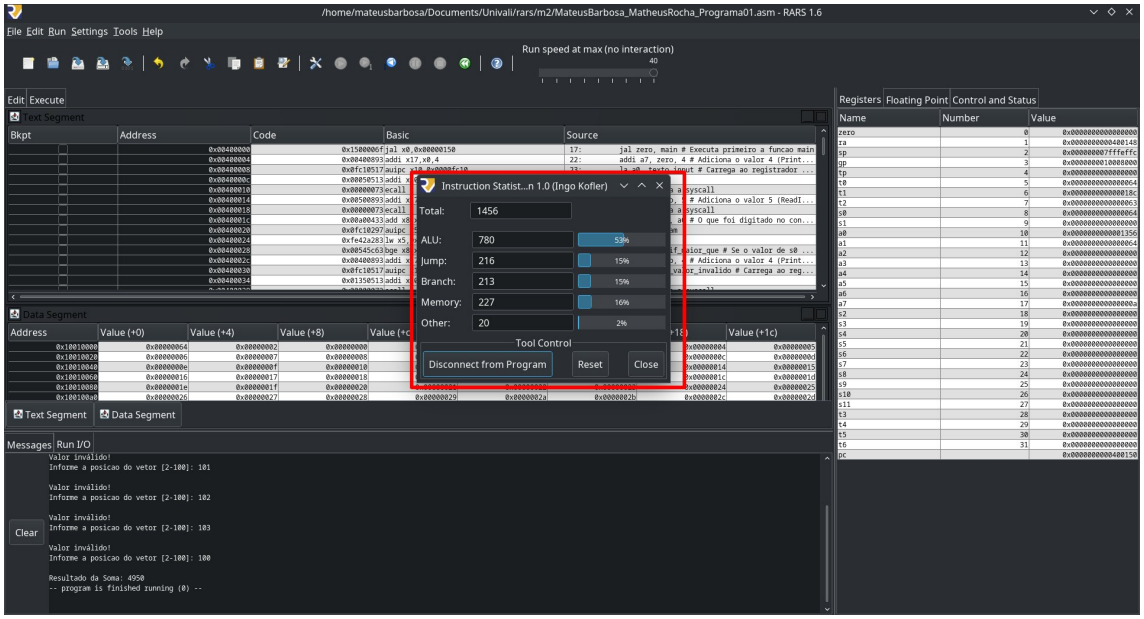

Figura 4: Instruction Statistics do Vetor de Tamanho 100

3. Programa 02

3.1 Enunciado: Utilizando a linguagem de montagem do RISC-V, implemente um procedimento recursivo que determine a soma dos elementos de um vetor de zero até a posição passada por parâmetro.

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

// Disciplina : Arquitetura e Organização de Computadores

// Atividade : Avaliação 03 – Programação em Linguagem de Alto nível

// Programa 02

// Grupo : - Mateus Barbosa

```
//        - Matheus de Oliveira Rocha

#include <iostream>
using namespace std;

int rec_vet_soma(int vet[], int pos)
{
   if (pos < 0)
   {
      return 0;
   }
   return vet[pos] + rec_vet_soma(vet, pos - 1);
}

int main()
{
   int pos;
   do
   {
      cout << "Informe o número de posicoes do vetor: ";
      cin >> pos;
   } while (pos < 2 || pos > 100);

   int vet[pos];
   for (int i = 0; i < pos; i++)
   {
      vet[i] = i;
   }


   cout << "Resultado: " << rec_vet_soma(vet, pos - 1);
   return 0;
}
```

3.3 Código fonte em Linguagem de Montagem do Risc-V

Explicação da Lógica do Código:

O Programa 02 possui o inicio igual o Programa 01, apenas diferindo na soma do vetor, em que se realiza de maneira recursiva ao invés de usar iteração.

"O programa interage com o usuário para solicitar uma posição do vetor, preencher o vetor com valores crescentes até essa posição e, em seguida, calcular e exibir a soma dos elementos do vetor até essa posição.

O programa começa definindo algumas variáveis e strings. As variáveis "max_tam" e "min_tam" são definidas como palavras (word) e armazenam os valores máximos e mínimos permitidos para a posição do vetor. A variável "vetor" é definida como uma palavra vazia para reservar um endereço de memória para o vetor. As strings "texto_input", "texto_resultado" e "texto_valor_invalido" armazenam mensagens a serem exibidas durante a interação com o usuário.

A função principal do programa é "main". Ela começa chamando a função "solicita_input_tam_vetor", que solicita ao usuário que insira a posição do vetor. O valor de retorno dessa função é armazenado em um registrador (s0).

Em seguida, uma mensagem de resultado é exibida ao usuário usando a syscall "PrintString". A string "texto_resultado" é passada como argumento para a syscall.

*Em seguida, o endereço base do vetor é carregado em um registrador (a0) e o número de posições do vetor é definido como o valor obtido anteriormente (s0). Duas funções são chamadas a seguir: "init_vetor" e "**rec_soma_vetor**".*

A função "init_vetor" é responsável por preencher o vetor com valores crescentes até a posição informada pelo usuário. Primeiro, verifica-se se o número de posições está dentro do intervalo permitido. Em seguida, o endereço base do vetor e o número de posições são inicializados em registradores (s0 e s1). Um loop é iniciado para percorrer todas as posições até o valor informado. Dentro do loop, a posição atual no vetor é calculada multiplicando o índice por 4 (cada elemento ocupa 4 bytes na memória). O valor de índice é então armazenado nessa posição. O índice é incrementado a cada iteração e o loop continua até que o número de posições seja alcançado."

A função rec_soma_vetor recebe dois argumentos: o endereço base do vetor (a0), a posição no vetor (a1). A primeira parte do código é responsável por salvar os registradores na pilha, alocando espaço para 4 registradores. Os registradores a serem salvos inicialmente são: ra (registrador de retorno) e a0 (base do vetor). Isso é feito para preservar esses valores durante as chamadas recursivas.

Em seguida, há uma verificação usando a instrução beqz para verificar se a posição (a1) é zero. Se for, a função retorna imediatamente com o valor 0, indicando que não há elementos para somar, e que chegou no final do vetor.

Se a posição não for menor que zero, a função continua executando. O valor da posição (a1) é decrementado em 1 usando a instrução addi e o novo valor é armazenado na pilha.

A próxima instrução é jal rec_soma_vetor, que realiza a chamada recursiva da função rec_soma_vetor. Isso permite que a função seja executada repetidamente até que a posição seja menor que zero.

Após todas as chamadas recursivas, a função começa a retornar os cálculos das posições. Os valores salvos na pilha são recuperados: o endereço base do vetor (a0) e o valor da posição atual (t1).

Em seguida, a função calcula a posição atual no vetor somando o endereço base do vetor com o deslocamento calculado a partir do valor da posição (4 * i). O valor do vetor na posição atual é carregado em t2.

O valor de retorno da função recursiva anterior é carregado em t3. Em seguida, a função soma o valor do retorno da função anterior com o valor da função atual, usando add e armazena o resultado em a0. Isso representa a soma acumulada dos elementos do vetor até a posição atual.

Finalmente, a função salta para o retorno (return_rec_soma_vetor) para sair da função atual. O registrador de retorno (ra) é carregado e os registradores salvos são desempilhados, liberando o espaço na pilha. Em seguida, ocorre uma chamada para retornar ao chamador original usando jalr ra, 0.

Código do Risc-V:

```
# Disciplina: Arquitetura e Organização de Computadores
# Atividade: Avaliação 03 – Programação de Procedimentos
# Programa 02
# Grupo:
#       - Mateus Barbosa
#       - Matheus de Oliveira Rocha


.data # Segmento de Dados
        max_tam: .word 100
        min_tam: .word 2
        vetor: .word # Vetor vazio para definir um endereço na memoria
        texto_input: .asciz "\nInforme a posicao do vetor [2-100]: "
        texto_resultado: .asciz "\nResultado da Soma: "
        texto_valor_invalido: .asciz "\nValor inválido!"


.text
        jal zero, main # Executa primeiro a funcao main
```

```
### Start: Solicita input entre min_tam e max_tam
solicita_input_tam_vetor:
    # Imprime: String texto_input_max_tam
    addi a7, zero, 4 # Adiciona o valor 4 (PrintString) ao registrador de serviço a7
    la a0, texto_input # Carrega ao registrador a0 o texto_input
    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

    lw t0, min_tam
    bge s0, t0, if_maior_que # Se o valor de s0 for maior que min_tam, 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 solicita_input_tam_vetor # Volta para o inicio de solicita_input_tam_vetor

if_maior_que: # Função que verifica se o valor informado é menor que tam
    lw t1, max_tam
    ble s0, t1, ret_solicita_input_tam_vetor # Se o valor de s0 (sem sinal) for menor que max_tam, vai para a função ret_solicita_input_tam_vetor

    # 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 solicita_input_tam_vetor # Volta para o inicio de solicita_input_tam_vetor
```

```
ret_solicita_input_tam_vetor:
        add a0, zero, s0
        jalr ra # Retorna para o chamador
### End: Solicita input entre min_tam e max_tam

### Start: Cria o vetor ate a posicao informada
init_vetor: # Dinamicamente preenche o vetor ate a posicao informada
        # Salvando registradores na pilha - Push
        addi sp, sp, -12
        sw s0, 0(sp)
        sw s1, 4(sp)
        sw ra, 8(sp)

        lw s0, max_tam # Verificando se o número de posições do vetor é maior que max_tam
        lw s1, min_tam # Verificando se o número de posições do vetor é menor que min_tam
        bge a1, s0, init_vetor_if_maior_que_max_tam
        blt a1, s1, init_vetor_if_menor_que_min_tam
        j posicao_init_vetor_ok

init_vetor_if_maior_que_max_tam:
        lw a1, max_tam # Caso o número de posições seja maior que max_tam, definimos como max_tam
        j posicao_init_vetor_ok

init_vetor_if_menor_que_min_tam:
        lw a1, min_tam # Caso o número de posições seja maior que min_tam, definimos como min_tam
        j posicao_init_vetor_ok

posicao_init_vetor_ok: # Input do usuario OK
        add s0, zero, a0 # Inicializando s0 com o endereço base do vetor
        add s1, zero, a1 # Inicializando s1 com o número de posições do vetor
```

```
        li t0, 0 # Inicializando variável i com 0
        j init_vetor_loop # Inicia o loop

init_vetor_loop:
        bge t0, s1, init_vetor_fim # Verificando se o número de posições do vetor foi alcançado

        slli t1, t0, 2 # Move 2 bits para a esquerda: 4 * i
        add t2, s0, t1 # Calcula a posicao no Vetor_A desde o seu comeco: comeco_do_Vetor + (4 * i)

        sw t0, 0(t2) # Preenchendo a posição atual do vetor com o valor de i

        addi t0, t0, 1 # Incrementando i

        j init_vetor_loop # Retornando ao início do loop

init_vetor_fim:
        # Retirando registradores da Pilha - Pop
        lw s0, 0(sp)
        lw s1, 4(sp)
        lw ra, 8(sp)
        addi sp, sp, 12

        jalr ra # Retorna pro chamador
### End: Cria o vetor ate a posicao informada

### Start: Soma elementos do vetor ate a posicao informada de maneira recursiva
rec_soma_vetor:
        # Dados armazenados na pilha (Topo eh o ultimo item):
        # 4) ra -> Registrador de retorno
        # 3) a0 -> base do vetor
        # 2) a1 -> posicao no vetor
```

```
# 1) Retorno recursivo da funcao rec_soma_vetor (valor final apos a execucao da funcao)

# Salvando registradores na pilha - Push
addi sp, sp, -16 # Armazenaremos 4 registradores na pilha
sw ra, 0(sp) # Armazena o registrador de retorno
sw a0, 4(sp) # Armazena o registrador contendo o endereco base do vetor

# beqz -> se igual a zero
beqz a1, return_zero # If (pos < 0) return 0;

addi a1, a1, -1 # Decrementa pos (a1) em 1
sw a1, 8(sp) # Armazena o valor de a1 na pilha

jal rec_soma_vetor # Chamada recursiva de rec_soma_vetor, ate que seja retornado 0

# Depois de todas as chamadas recursivas, inicia-se os retornos com os calculos das posicoes
sw a0, 12(sp) # Armazena o valor recursivo da funcao rec_soma_vetor (valor final apos a execucao da funcao)
lw t0, 4(sp) # Carrega no registrador t0 o endereco base do vetor
lw t1, 8(sp) # Carrega no registrador t1 o valor da posicao atual (i == contador)

# Pega o elemento na posicao atual do vetor
slli t1, t1, 2 # Move 2 bits para a esquerda: 4 * i
add t0, t0, t1 # Calcula a posicao no Vetor_A desde o seu comeco: comeco_do_Vetor + (4 * i)
lw t2, 0(t0) # Valor do vetor na posicao i

lw t3, 12(sp) # Carrega o valor recursivo da funcao rec_soma_vetor (valor final apos a execucao da funcao)
add a0, t3, t2 # Soma o valor do retorno da funcao anterior com o valor da funcao atual ==soma(vet, pos-1 ) + vet[pos]
j return_rec_soma_vetor # Pula para o retorno que sai da funcao atual (Nao se faz o jump link pois precisamos saber o ra da funcao pai)
```

```
return_zero:
        add  a0, zero, zero # Retorna 0
        jalr ra, 0 # Retorna para o chamador

return_rec_soma_vetor:
        add a0, a0, zero # Copia o valor de retorno da função atual para a0
        lw ra, 0(sp) # Carrega o registrador de retorno
        addi sp, sp, 16  # Remvoe o espaço na pilha usado pelos 4 registradores
        jalr ra, 0 # Retorna para o chamador
### End: Soma elementos do vetor ate a posicao informada de maneira recursiva

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

main:
        jal ra, solicita_input_tam_vetor # Chama a funcao para solicitar posicao do vetor
        add s0, zero, a0 # O que foi digitado no console (registrador a0), é salvo no registrador s0

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

        # Argumentos:
        #   a0 - endereço base do vetor
        #   a1 - número de posições do vetor
        # O vetor vai de 0 ate o 99 (max_tam - 1) para os valores dos elementos
        la a0, vetor # Carrega o endereco de memoria do vetor em a0
        add a1, zero, s0 # Define o tamanho do vetor e a posicao (index)
        jal ra, init_vetor # Chama init_vetor e define ra para a proxima linha
```

```
# Argumentos:
#   a0 - endereço base do vetor
#   a1 - número de posições do vetor
jal ra, rec_soma_vetor # Chama soma_vetor e define ra para a proxima linha


# Imprime: Int resultado da soma
addi a7, zero, 1 # Adiciona o valor 1 (PrintInt) ao registrador de serviço a7
ecall # Chama a syscall


j exit # Sai do programa
```

3.4 resultados

Informações da execução:

- Tamanho do Vetor: 52
- Tamanho do Vetor, após informar 5 valores inválidos: 100
- O último valor do vetor será sempre o max_tam - 1
- Resultado Final da Operação:
  - Vetor com tamanho 52: 1326
  - Vetor com tamanho 100: 4950

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

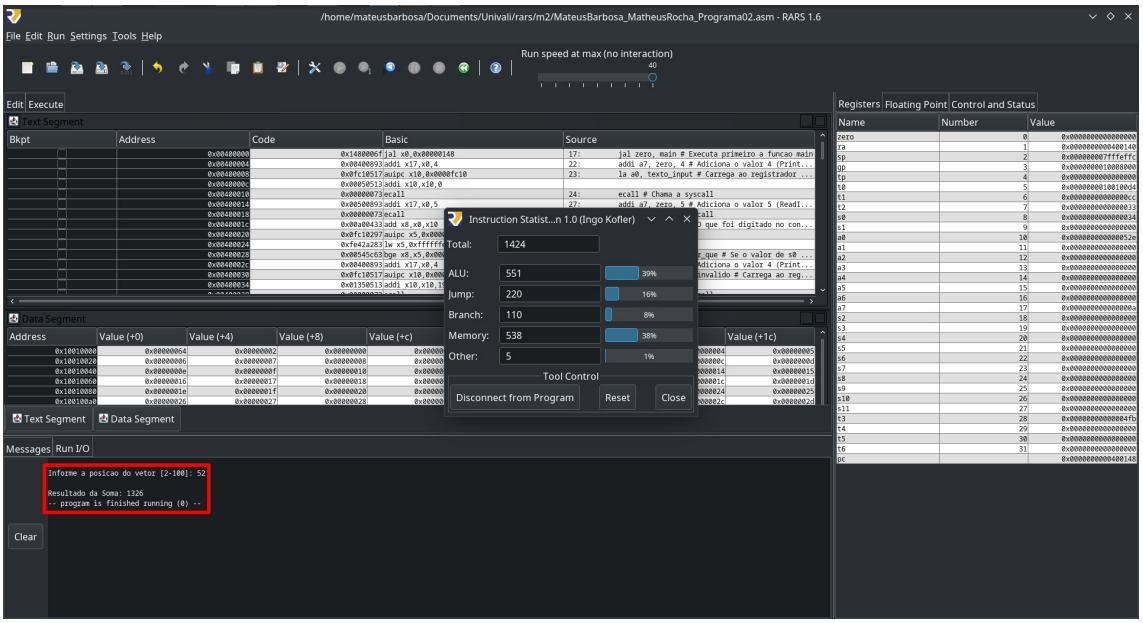

Figura 5: Vetor de Tamanho 52

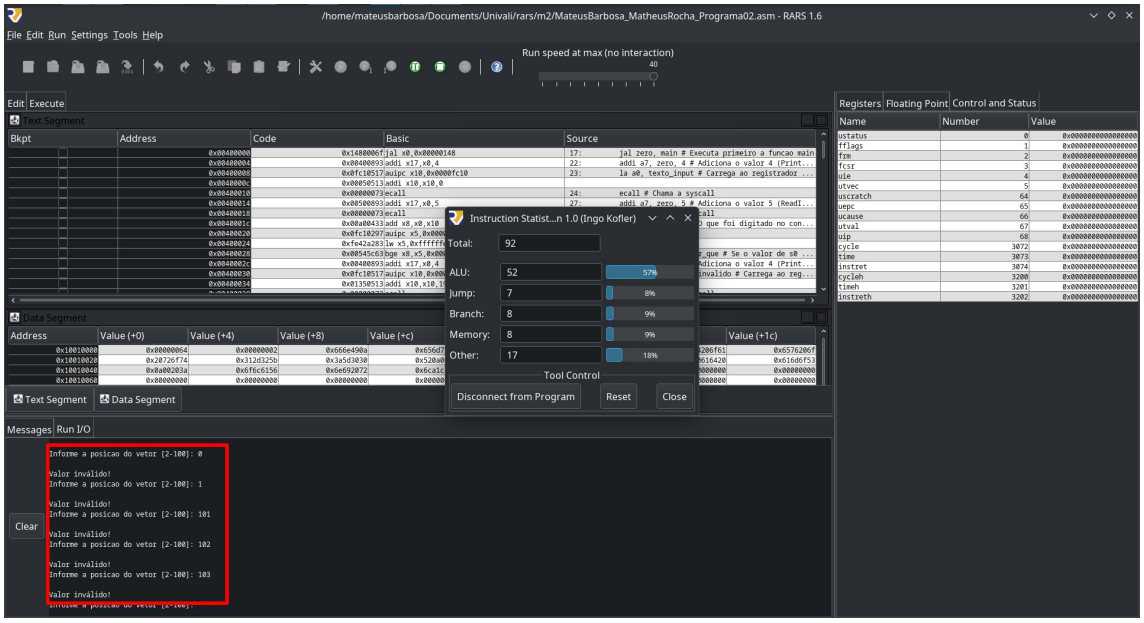

Figura 6.1: Vetor de Tamanho 100, informando 5 tamanhos inválidos

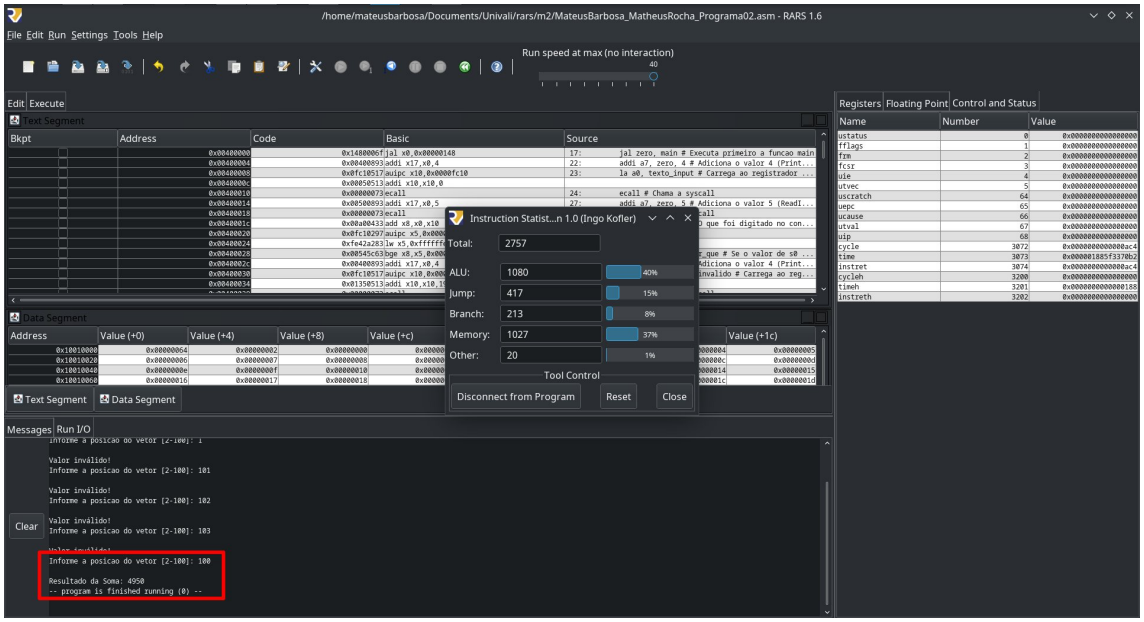

Figura 6.2: Vetor de Tamanho 100, informando o tamanho válido

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

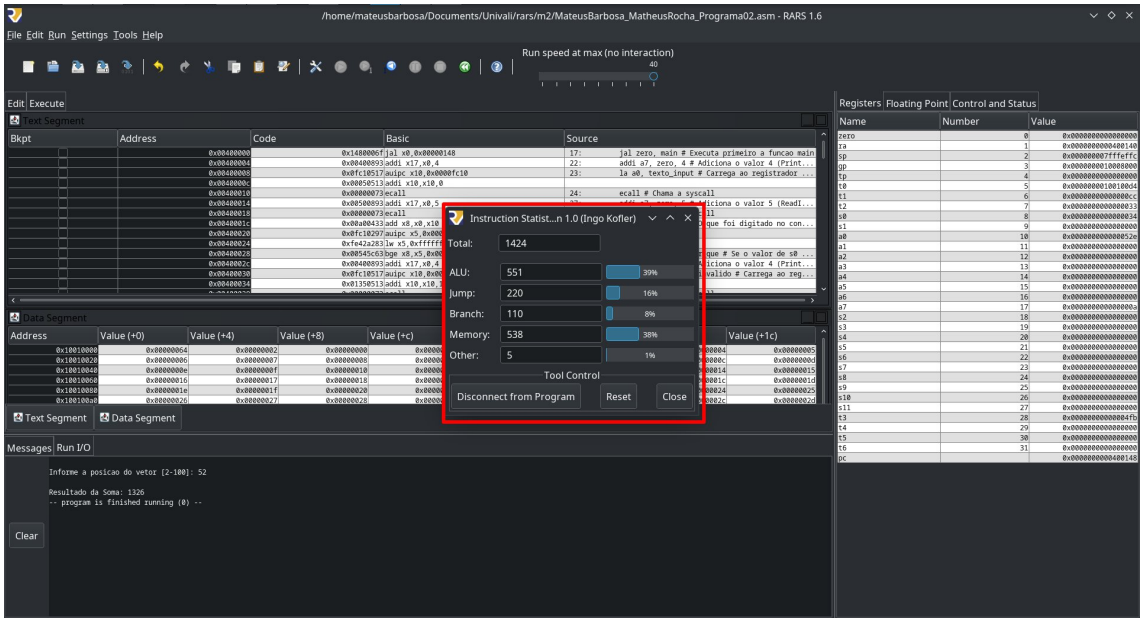

Figura 7: Instruction Statistics do Vetor de Tamanho 52

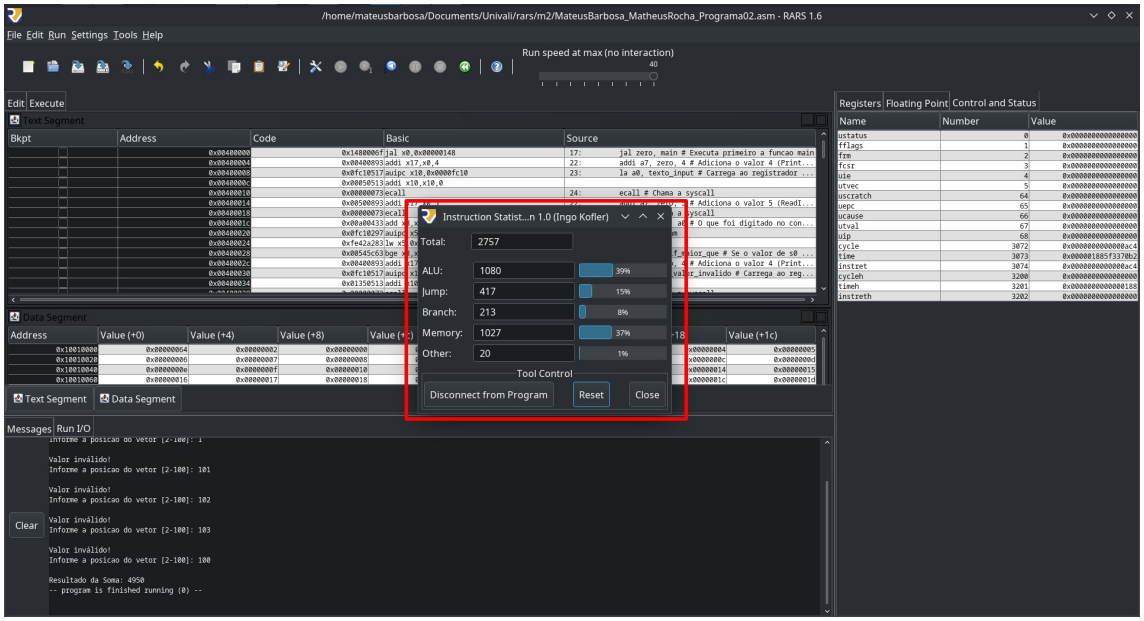

Figura 8: Instruction Statistics do Vetor de Tamanho 100

4. Analise dos Resultados

O primeiro código *(seção 2. Programa 01)*, que usa uma abordagem iterativa, é mais eficiente em termos de tempo de execução. Ela utiliza um loop para percorrer as posições do vetor e somar seus valores, evitando chamadas recursivas e empilhamento de quadros de função. Isso faz com que ele precise de menos instruções para finalizar a tarefa.

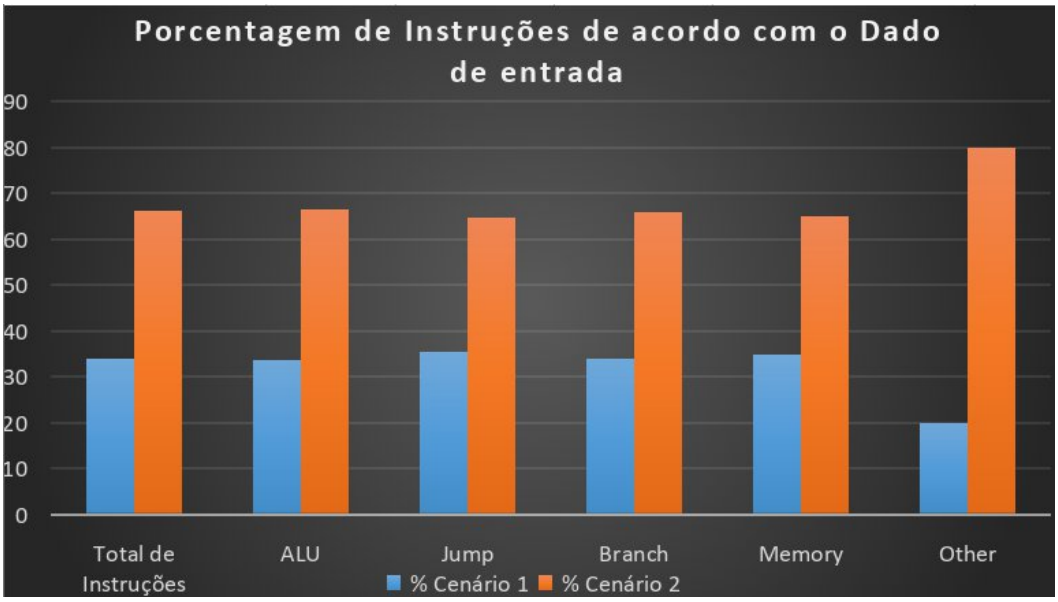

Figura 9: Gráfico mostrando a diferença na execução do Programa 01 com o vetor tendo 2 tamanhos diferentes 52 (Cenário 1) e 100(Cenário 2)

O segundo código *(seção 3. Programa 02)*, usa uma abordagem recursiva, é menos eficiente em termos de tempo de execução. Isso ocorre porque a recursão envolve chamadas de função adicionais e empilhamento de quadros de função, o que pode resultar em um maior consumo de memória, pilha e tempo de execução.

Comparando os dados do cenário 1 e cenário 2 da recursividade (Figura 10) com o dados do cenário 1 e cenário 2 da iteração (Figura 9), vemos que de acordo com os dados eles possuem as mesmas diferenças entre os cenários, se escalando de maneira linear.

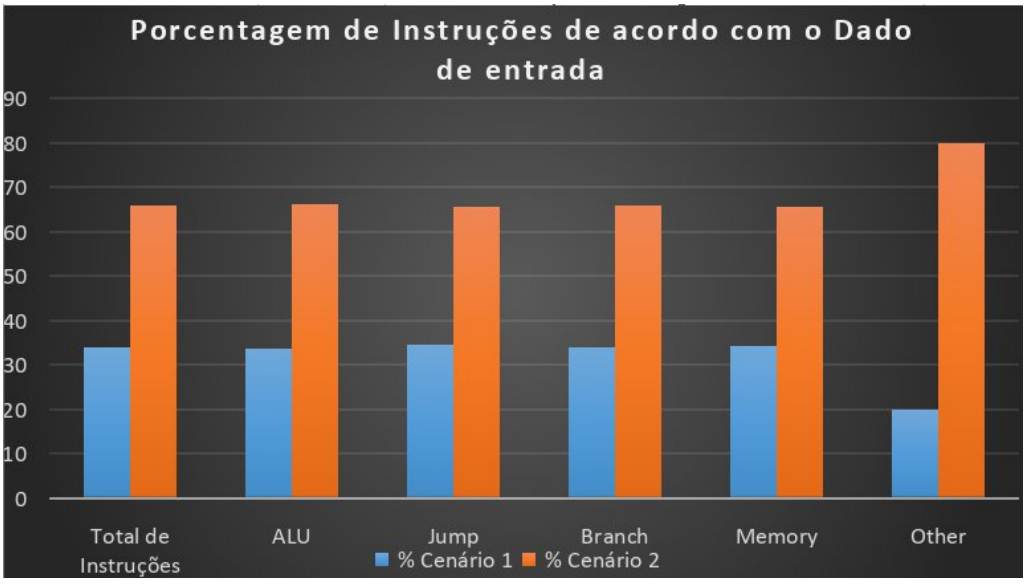

Figura 10: Gráfico mostrando a diferença na execução do Programa 02 com o vetor tendo 2 tamanhos diferentes 52 (Cenário 1) e 100(Cenário 2)

Mas o desempenho das soluções pode ser afetado de diferentes maneiras. Se a posição informada for pequena, ambas as soluções terão um desempenho semelhante, pois o número de iterações no loop será baixo. No entanto, à medida que a posição informada aumenta, a solução recursiva tende a ter um desempenho pior em comparação com a solução iterativa.

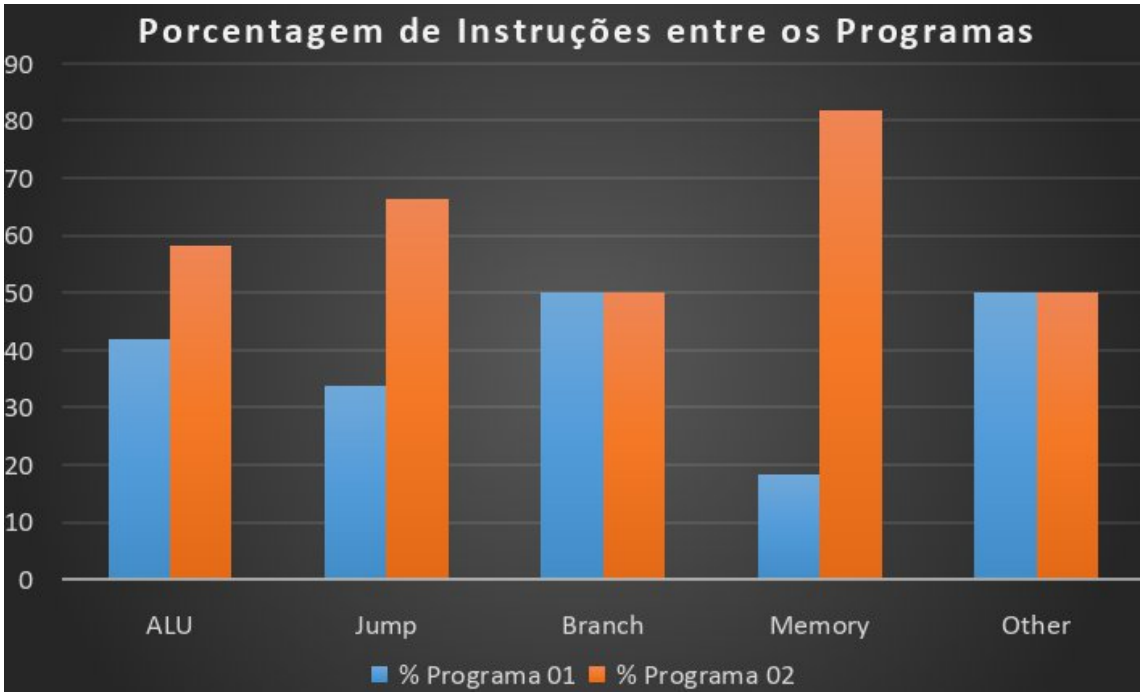

Figura 11: Gráfico mostrando a diferença na execução do Programa 01 e Programa 02, no uso de cada tipo das Instruções do RISC-V

Isso acontece porque a solução recursiva precisa empilhar e desempilhar quadros de função para cada chamada recursiva, o que consome tempo e memória adicional. Conforme o tamanho do problema aumenta, o número de chamadas recursivas também aumenta, resultando em um aumento significativo no tempo de execução e no consumo de memória. Por outro lado, a solução iterativa apenas executa um loop, sem o custo adicional de chamadas recursivas, e, portanto, tende a ter um desempenho melhor.

Portanto, em termos de desempenho, a solução iterativa é preferível, especialmente quando a posição informada é grande. A complexidade O(n) em ambos os casos indica que o tempo de execução aumenta linearmente com o tamanho do problema, mas a solução iterativa tem um fator de multiplicação menor devido à ausência de chamadas recursivas.

Abaixo está uma explicação mais a fundo da eficiência dos códigos usando a notação Big O como base:

1. ***Soma dos Elementos do Vetor por Iteração:***
    - **a.** *Tempo de Execução:* O(n), onde "n" é o tamanho do vetor. O algoritmo percorre todos os elementos do vetor uma vez, realizando a soma. Portanto, o tempo de execução é linear em relação ao tamanho do vetor.
    - ***b.*** *Espaço de Memória:* O(1), ou seja, constante. O algoritmo não requer espaço adicional que cresça com o tamanho do vetor. Apenas algumas variáveis auxiliares são necessárias, independentemente do tamanho do vetor.
2. **Soma dos Elementos do Vetor por Recursividade:**
    - **a.** *Tempo de Execução Recursivo:* O(n), onde "n" é a posição informada no vetor. O tempo de execução é linear, pois o número de chamadas recursivas é igual à posição informada. Cada chamada recursiva resulta em uma chamada da mesma função, o que leva a um crescimento linear do tempo de execução à medida que "n" aumenta.
    - ***b.*** *Espaço de Memória Recursivo:* O(n), onde "n" é a posição informada no vetor. O espaço de memória necessário é proporcional ao número de chamadas recursivas, que é igual à posição informada. Cada chamada recursiva requer um novo quadro de função na pilha, resultando em um consumo de memória linear à medida que "n" aumenta.

De acordo com o Big O, podemos ver que o nível de complexidade do tempo de execução é o mesmo para ambos os cenários. Mas o acesso a memória é muito maior na função recursiva, pelo acesso recorrente da pilha.

No final, obtivemos os dados de ambos os programas incluindo todos os cenários realizados em ambos os códigos, e o programa 02 teve uma execução de mais de 60% do total das instruções. Lembrando que na recursividade, a pilha foi usada com muito mais frequência, necessitando mais instruções.

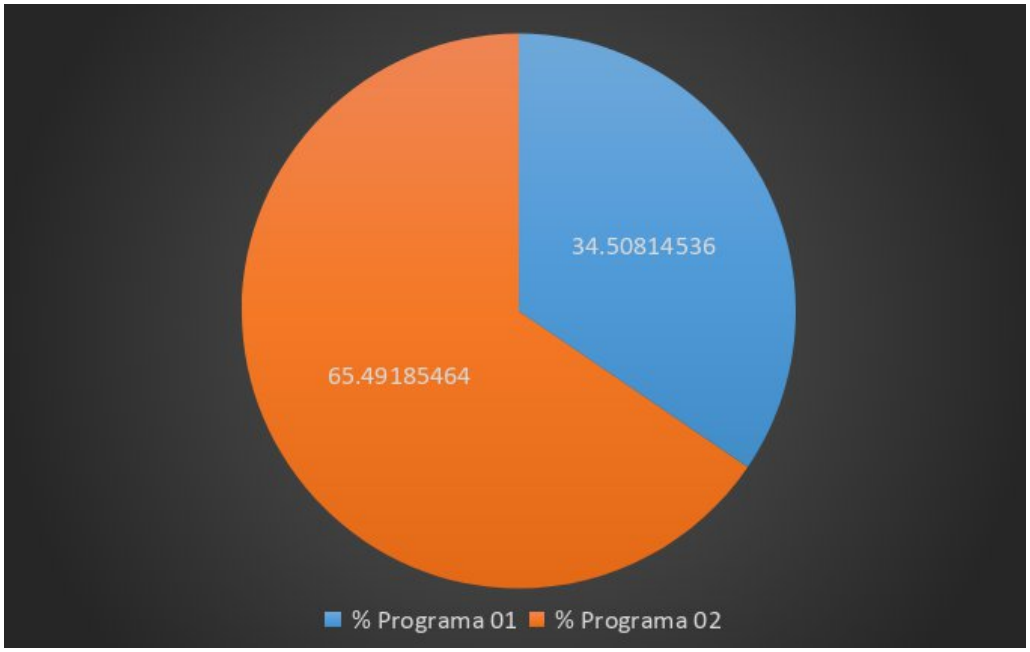

Figura 12: Gráfico mostrando a diferença na execução do Programa 01 e Programa 02, no uso total das Instruções do RISC-V