Relatório

Cauã Ribas, Haran Souza, Nilson Andrade

Universidade do Vale do Itajaí - Univali
Escola Politécnica
Ciência da Computação
(cauaribas, haran ,nilson.neto) @edu.univali.br

Arquitetura e Organização de Processadores

Avaliação 03 - Programação em linguagem de montagem

Thiago Felski Pereira

13/11/2023

1. Introdução:

Este relatório descreve dois programas usando a Linguagem de Montagem do Risc-V desenvolvidos para a disciplina de Arquitetura e Organização de Computadores. Os programas têm o objetivo de manipular vetores, solicitando ao usuário o tamanho do vetor e realizando operações como inicialização e soma de seus elementos. Também recriamos os códigos usando uma linguagem de Alto Nível, C/C++. E por fim, fizemos um comparativo da eficiência dos códigos.

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++

```
int vet_soma(int vet[TAM], int pos) {
        int soma=0;
        for (int i=0; i<pos; i++) {
                soma = soma + vet[i];
        }
        return soma;
}
```

2.3 Código fonte em Linguagem de Montagem do Risc-V

Código do Risc-V:

```
# Disciplina: Arquitetura e Organização de Computadores
# Atividade: Avaliação 03 – Programação de Procedimentos
# Grupo: - Cauã Ribas
#        - Nilson Andrade
#        - Haran Souza
#
.data # Dados
        vetor: .word
        min_tam: .word 2
        max_tam: .word 100
        texto_input: .asciz "\nInforme até qual posição o vetor será
somado(2-100): "
        texto_resultado: .asciz "\nResultado da soma dos elementos do vetor: "
```

```
.text # Codigo
	jal zero, main # Executa o main antes das funções

loop_define_tam_vetor:
	# Imprime: String texto_input, ate que o usuario defina um valor valido para o tamanho do vetor
	addi a7, zero, 4 # Adiciona o valor 4 ao registrador de serviço a7 (PrintString)
	la a0, texto_input # Carrega o texto texto_input ao registrador a0
	ecall # Chama o syscall

	# Solicita: Int tamanho do vetor
	addi a7, zero, 5 # Adiciona o valor 5 ao registrador de serviço a7 (ReadInt)
	ecall # Chama o syscall
	add s0, zero, a0 # Input do usuario (registrador a0), é salvo no registrador s0

	blt s0, t0, loop_define_tam_vetor # Verifica se a0 é menor que o tamanho mínimo
	bgt s0, t1, loop_define_tam_vetor # Verifica se a0 é maior que o tamanho máximo

inicializar_vetor: # Dinamicamente preenche o vetor ate a posicao informada
	# Salvando os Registradores na Pilha - Push
	addi sp, sp, -12
	sw s0, 0(sp)
	sw s1, 4(sp)
	sw ra, 8(sp)

	add s0, zero, a0 # Inicializando s0 com o endereço base do vetor
	add s1, zero, a1 # Inicializando s1 com o numero de posições do vetor

	li t0, 0 # Inicializando variavel i com 0
	j inicializar_loop_vetor # Inicia o loop

inicializar_loop_vetor:
```

```
        bge t0, s1, inicializar_vetor_fim # Verificando se o numero de posições do vetor foi alcançado(se t0 >= s1 termina o loop)

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

        sw t0, 0(t2) # Guarda o valor informado pelo usuario no t0, com um offset de 0 bits, no vetor (a0)

        addi t0, t0, 1 # Incremento do contador: i = i + 1

        j inicializar_loop_vetor # "Reinicia o loop"

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

        jalr ra # Retorna ao chamador

soma_vetor_rec:
        # Salvando Registradores na Pilha - Push
        addi sp, sp, -16 # Armazena 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 a1, return_zero

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

        jal soma_vetor_rec # Chamada recursiva de soma_vetor_rec, ate que seja retornado 0
```

```
        # Depois de todas as chamadas recursivas, inicia-se os retornos com os calculos das posicoes
        sw a0, 12(sp)
        lw t0, 4(sp)
        lw t1, 8(sp)

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

        lw t3, 12(sp)
        add a0, t3, t2 # soma(vet, pos - 1) + vet[i]
        j return_soma_vetor_rec

return_zero:
        add a0, zero, zero # Retorna 0
        jalr ra, 0 # Retorna ao chamador

return_soma_vetor_rec:
        add a0, a0, zero # Copia o valor de retorno da funcao atual para a0
        lw ra, 0(sp) # Carrega o registrador de retorno
        addi sp, sp, 16 # Remove o espaco na pilha usado pelos 4 registradores
        jalr ra, 0 # Retorna para o chamador

main:
        lw t0, min_tam
        lw t1, max_tam

        jal ra, loop_define_tam_vetor # Chama a funcao para solicitar o tamanho do vetor
        add s0, zero, a0 # Input do usuario (registrador a0), é salvo no registrador s0

        # Imprime: String texto_resultado
```

```
    addi a7, zero, 4 # Adiciona o valor 4 ao registrador de serviço a7
(PrintString)
    la a0, texto_resultado # Carrega o texto texto_resultado ao registrador a0
    ecall # Chama o syscall

    # Argumentos:
    # a0 - Registrador de endereço base do vetor
    # a1 - Registrador do tamanho do vetor (numero de posições)
    la a0, vetor # Carrega o endereco de memoria do vetor em a0
    add a1, zero, s0 # Define o tamanho do vetor e a posição
    jal ra, inicializar_vetor

    # Argumentos:
    # a0 - Registrador de endereço base do vetor
    # a1 - Registrador do tamanho do vetor (numero de posições)
    jal ra, soma_vetor_rec

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

    # Return 0
    addi a7, zero, 10 # Adiciona o valor 10 (Exit(0)) ao registrador de serviço
a7
    ecall # Chama a syscall
```

Explicação Lógica do Código:

O programa inicia solicitando ao usuário o tamanho desejado para o vetor. O loop loop_define_tam_vetor é responsável por exibir a mensagem texto_input e utilizar a chamada do sistema ReadInt para obter a entrada do usuário. A entrada é armazenada em s0 (registrador temporário). O programa verifica se o tamanho do vetor (s0) está dentro dos limites predefinidos (min_tam e max_tam). Caso não esteja, o programa repete o processo até receber um tamanho válido.

Uma vez que o tamanho do vetor é determinado, o programa passa para a função inicializar_vetor. Esta função utiliza um loop (inicializar_loop_vetor)

para preencher dinamicamente o vetor até a posição especificada pelo usuário (a1). Inicialmente, os registradores de salva-estado (s0, s1, ra) são salvos na pilha.

Dentro do loop, o programa calcula a posição no vetor (t2) com base no índice t0 e o endereço base do vetor (s0). O valor fornecido pelo usuário (t0) é então armazenado no vetor. O contador (t0) é incrementado, e o processo se repete até que a posição informada pelo usuário seja atingida. Após o loop, os registradores são restaurados da pilha, e a função retorna ao chamador.

A função soma_vetor é responsável por calcular a soma dos elementos do vetor de maneira iterativa. Assim como na função anterior, os registradores de salva-estado são preservados na pilha. Os registradores s0 e s1 são inicializados com o endereço base do vetor e o número de posições do vetor, respectivamente.

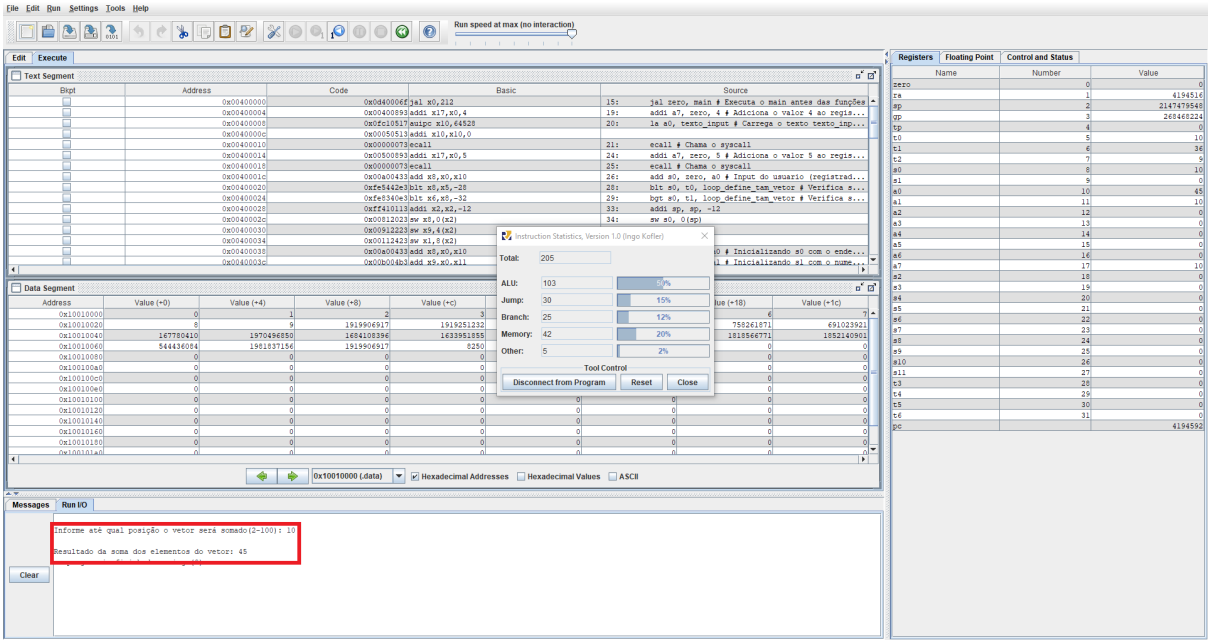
Um loop (loop_soma_vetor) é utilizado para iterar sobre o vetor. Dentro do loop, a posição no vetor (t2) é calculada com base no índice t0 e o endereço base do vetor (s0). O valor contido na posição atual é então carregado (lw t2, 0(t2)) e somado ao total (s2).

O contador (t0) é incrementado, e o processo é repetido até que todas as posições do vetor sejam percorridas. Após o loop, o total (s2) é armazenado no registrador de retorno (a0), e os registradores são restaurados da pilha. O programa, então, retorna ao chamador.

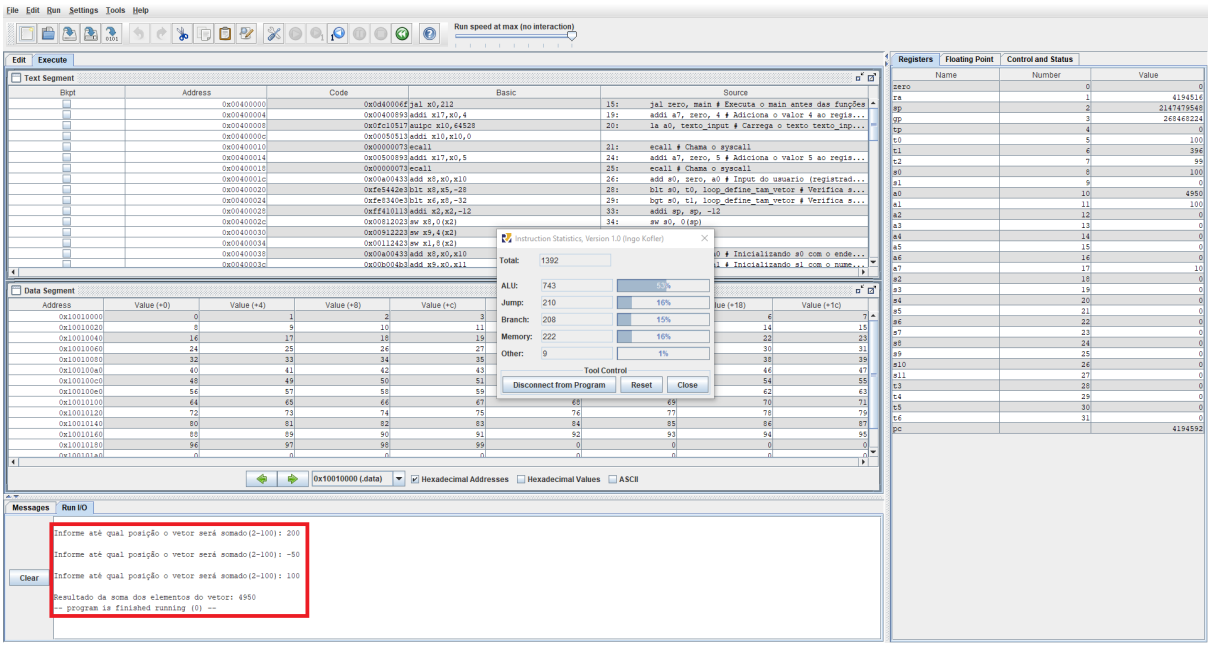
2.4 Resultados

Informações da execução:
— Tamanho do vetor: 10
— Resultado final da operação de soma:
Vetor com tamanho 10: 45

Abaixo estão as capturas de tela (Prints) da execução do programa inserindo os valores informados acima.

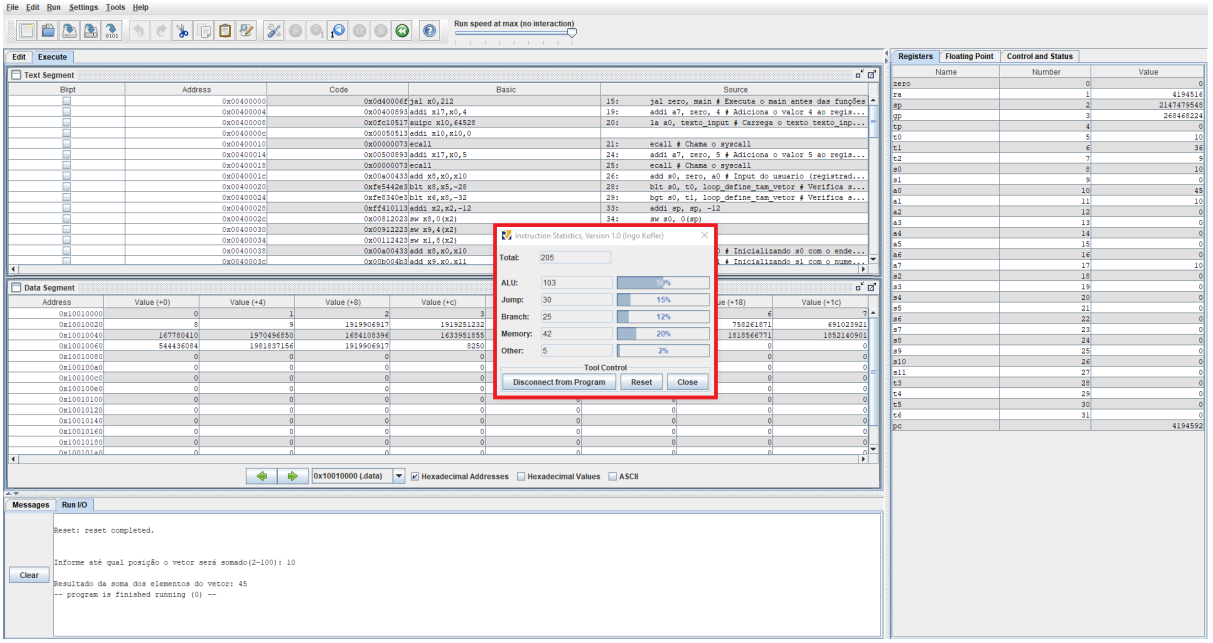


Vetor de tamanho 10.

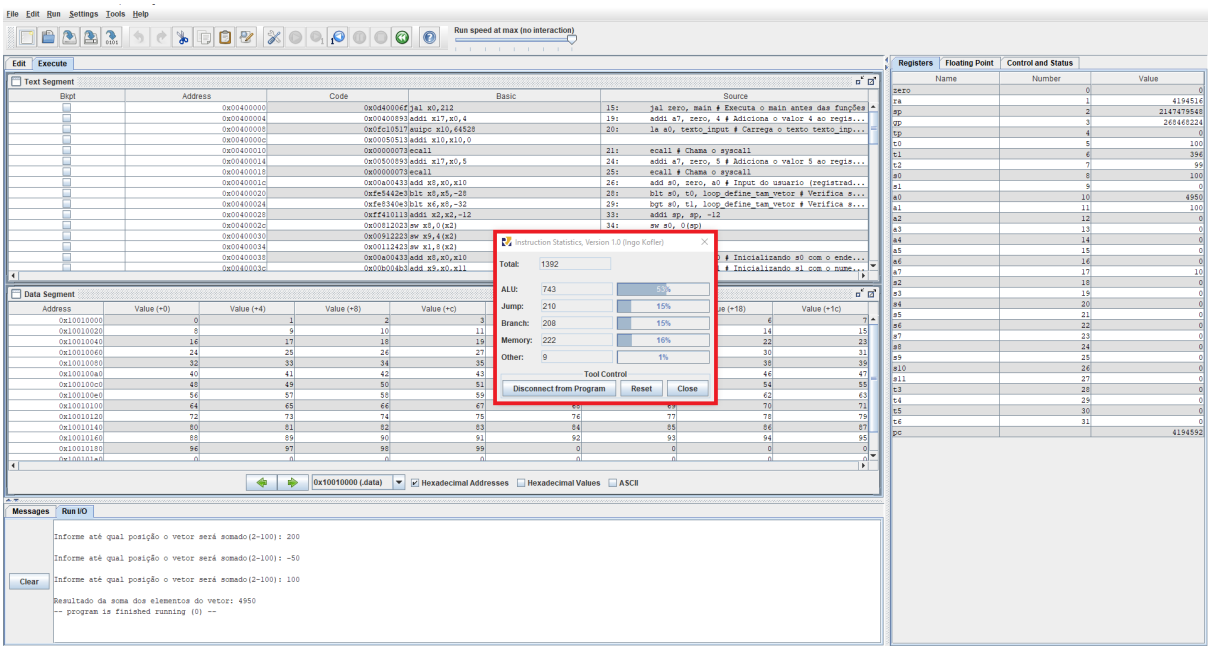


Informando tamanhos inválidos. Após informar o tamanho valido 100, obtém-se o resultado 4950.

Abaixo consta o Instructions Statistics no Programa 01 e seus resultados.



Vetor de tamanho 10



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++

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

3.3 Código fonte em Linguagem de Montagem do Risc-V

Código do Risc-V:

```
# Disciplina: Arquitetura e Organização de Computadores
# Atividade: Avaliação 03 – Programação de Procedimentos
# Grupo: - Cauã Ribas
#            - Nilson Andrade
#            - Haran Souza
#
.data # Dados
        vetor: .word
        min_tam: .word 2
        max_tam: .word 100
        texto_input: .asciz "\nInforme até qual posição o vetor será
somado(2-100): "
        texto_resultado: .asciz "\nResultado da soma dos elementos do vetor: "

.text # Codigo
        jal zero, main # Executa o main antes das funções

loop_define_tam_vetor:
        # Imprime: String texto_input, ate que o usuario defina um valor valido
para o tamanho do vetor
        addi a7, zero, 4 # Adiciona o valor 4 ao registrador de serviço a7
(PrintString)
        la a0, texto_input # Carrega o texto texto_input ao registrador a0
```

```
        ecall # Chama o syscall

        # Solicita: Int tamanho do vetor
        addi a7, zero, 5 # Adiciona o valor 5 ao registrador de serviço a7 (ReadInt)
        ecall # Chama o syscall
        add s0, zero, a0 # Input do usuario (registrador a0), é salvo no registrador s0

        blt s0, t0, loop_define_tam_vetor # Verifica se a0 é menor que o tamanho mínimo
        bgt s0, t1, loop_define_tam_vetor # Verifica se a0 é maior que o tamanho máximo

inicializar_vetor: # Dinamicamente preenche o vetor ate a posicao informada
        # Salvando os Registradores na Pilha - Push
        addi sp, sp, -12
        sw s0, 0(sp)
        sw s1, 4(sp)
        sw ra, 8(sp)

        add s0, zero, a0 # Inicializando s0 com o endereço base do vetor
        add s1, zero, a1 # Inicializando s1 com o numero de posições do vetor

        li t0, 0 # Inicializando variavel i com 0
        j inicializar_loop_vetor # Inicia o loop

inicializar_loop_vetor:
        bge t0, s1, inicializar_vetor_fim # Verificando se o numero de posições do vetor foi alcançado(se t0 >= s1 termina o loop)

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

        sw t0, 0(t2) # Guarda o valor informado pelo usuario no t0, com um offset de 0 bits, no vetor (a0)
```

```
        addi t0, t0, 1 # Incremento do contador: i = i + 1

        j inicializar_loop_vetor # "Reinicia o loop"

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

        jalr ra # Retorna ao chamador

soma_vetor_rec:
        # Salvando Registradores na Pilha - Push
        addi sp, sp, -16 # Armazena 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 a1, return_zero

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

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

        # Depois de todas as chamadas recursivas, inicia-se os retornos com os
calculos das posicoes
        sw a0, 12(sp)
        lw t0, 4(sp)
        lw t1, 8(sp)

        # Pega o elemento na posição atual do vetor
        slli t1, t1, 2 # Move 2 bits para a esquerda: 4*i
```

```
        add t0, t0, t1 # Calcula a posicao no vetor desde o seu comeco: comeco do vetor + (4 * i)
        lw t2, 0(t0) # Valor do vetor na posicao i

        lw t3, 12(sp)
        add a0, t3, t2 # soma(vet, pos - 1) + vet[i]
        j return_soma_vetor_rec

return_zero:
        add a0, zero, zero # Retorna 0
        jalr ra, 0 # Retorna ao chamador

return_soma_vetor_rec:
        add a0, a0, zero # Copia o valor de retorno da funcao atual para a0
        lw ra, 0(sp) # Carrega o registrador de retorno
        addi sp, sp, 16 # Remove o espaco na pilha usado pelos 4 registradores
        jalr ra, 0 # Retorna para o chamador

main:
        lw t0, min_tam
        lw t1, max_tam

        jal ra, loop_define_tam_vetor # Chama a funcao para solicitar o tamanho do vetor
        add s0, zero, a0 # Input do usuario (registrador a0), é salvo no registrador s0

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

        # Argumentos:
        # a0 - Registrador de endereço base do vetor
        # a1 - Registrador do tamanho do vetor (numero de posições)
        la a0, vetor # Carrega o endereco de memoria do vetor em a0
```

```
    add a1, zero, s0 # Define o tamanho do vetor e a posição
    jal ra, inicializar_vetor

    # Argumentos:
    # a0 - Registrador de endereço base do vetor
    # a1 - Registrador do tamanho do vetor (numero de posições)
    jal ra, soma_vetor_rec

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

    # Return 0
    addi a7, zero, 10 # Adiciona o valor 10 (Exit(0)) ao registrador de serviço
a7
    ecall # Chama a syscall
```

Explicação Lógica do Código:

Um loop (loop_define_tam_vetor) é utilizado para solicitar ao usuário o tamanho do vetor, garantindo que o tamanho informado esteja dentro de limites predefinidos (entre min_tam e max_tam).

A função inicializar_vetor é responsável por preencher dinamicamente o vetor até a posição informada pelo usuário. Utiliza um loop (inicializar_loop_vetor) para iterar sobre o vetor e armazenar valores fornecidos pelo usuário.

A função soma_vetor_rec é central para a lógica do programa. Esta função realiza uma soma recursiva dos elementos do vetor. A recursão é implementada da seguinte maneira:

Condição de Base (Caso Base): Antes de realizar qualquer chamada recursiva, a função verifica se o tamanho do vetor (a1) é igual a zero. Se for, a função retorna imediatamente zero, indicando o fim da recursão.

Caso Recursivo: Se o tamanho do vetor não for zero, a função decrementa o tamanho do vetor (a1) em 1 e armazena esse valor na pilha. Em seguida, faz uma chamada recursiva para soma_vetor_rec. Este passo é crucial, pois a função agora soma o elemento atual do vetor (vet[i]) com o resultado da chamada recursiva que soma os elementos restantes do vetor.

Após todas as chamadas recursivas, o programa inicia os retornos, calculando a soma final dos elementos do vetor. return_zero: Se o tamanho do vetor atingir zero, a função retorna zero. return_soma_vetor_rec: Após a chamada recursiva, o resultado é retornado, sendo a soma do elemento atual do vetor com o resultado da soma dos elementos restantes.

3.4 Resultados

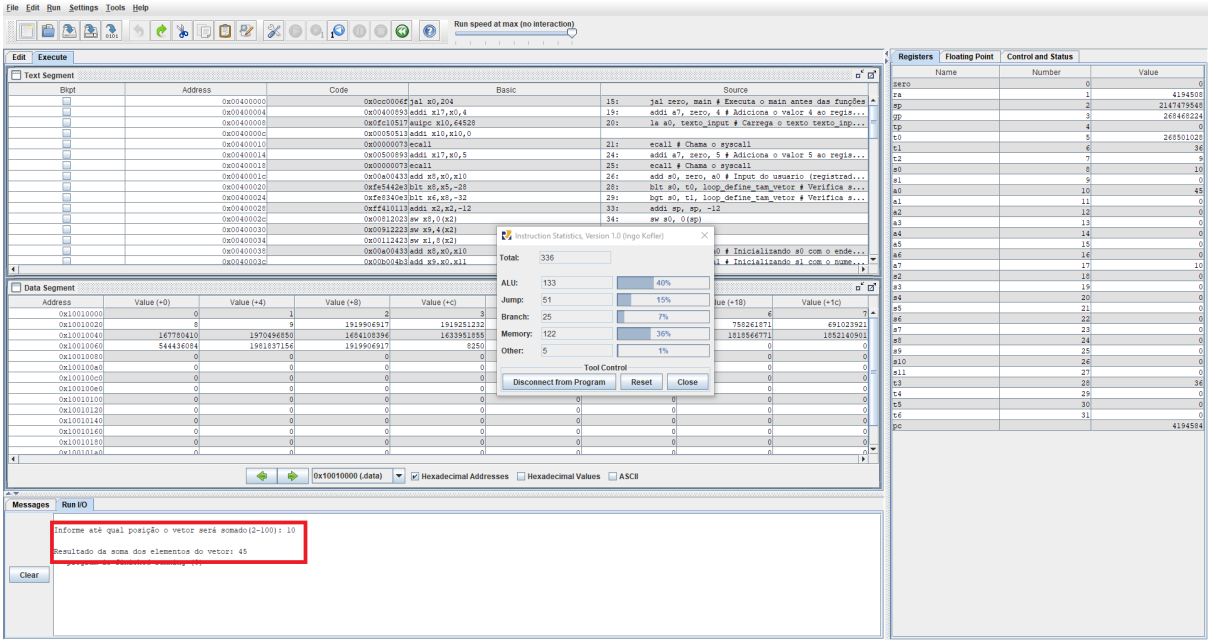
Informações da execução:

— Tamanho do vetor: 10

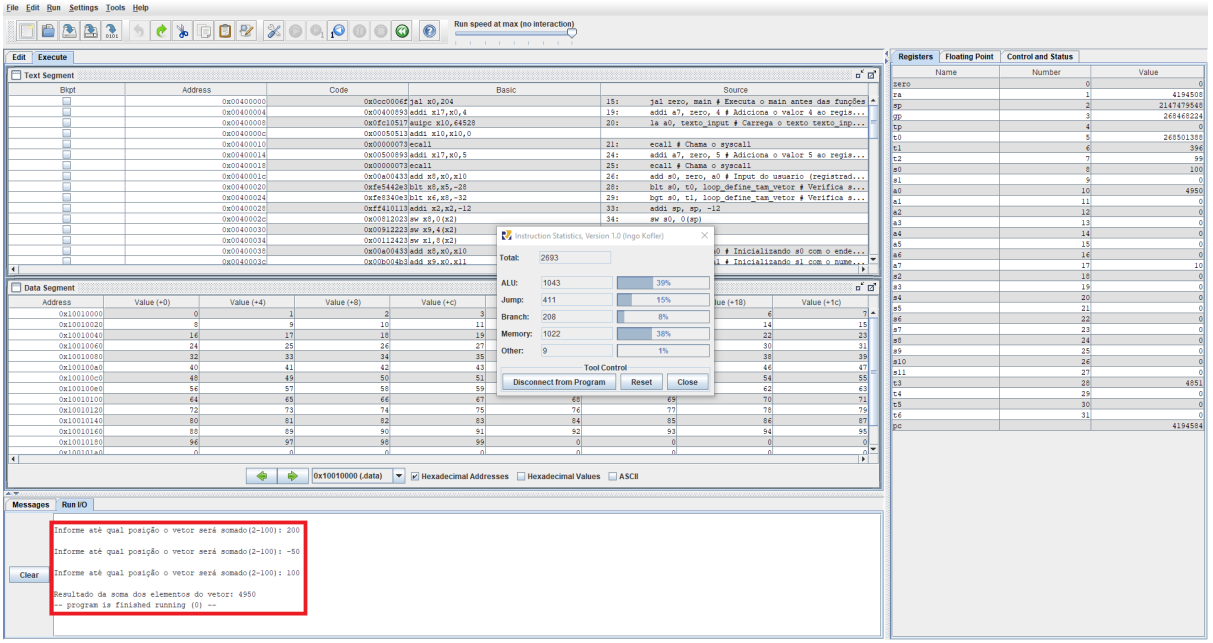
— Resultado final da operação de soma:

Vetor com tamanho 10: 45

Abaixo estão as capturas de tela (Prints) da execução do programa inserindo os valores informados acima.

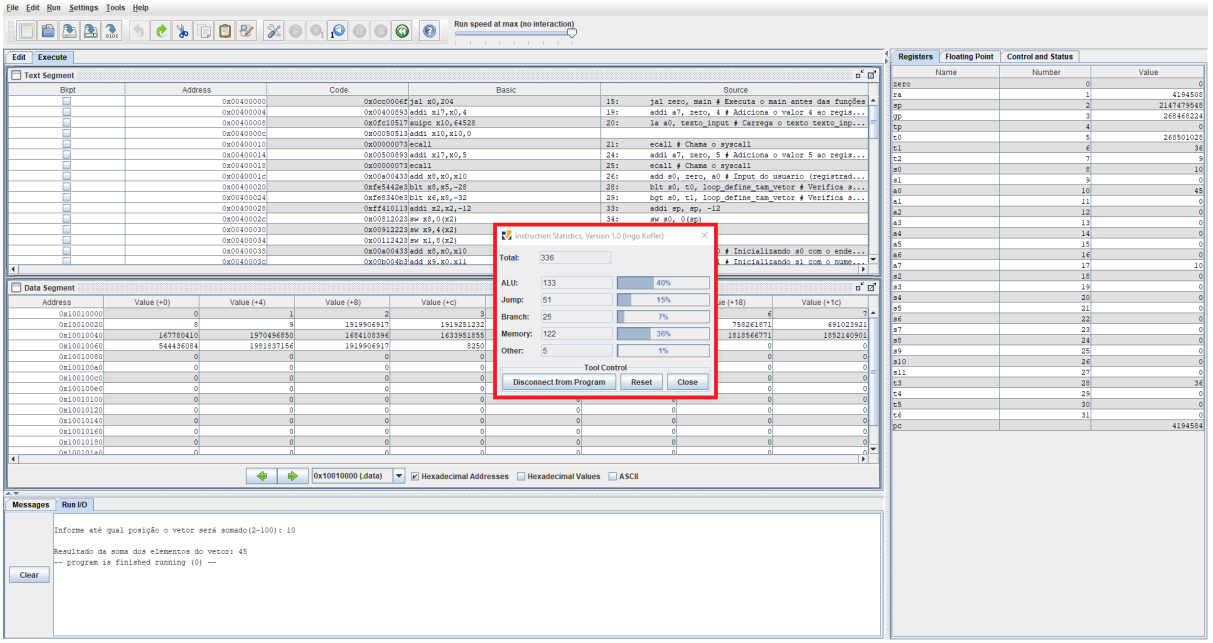


Vetor de tamanho 10

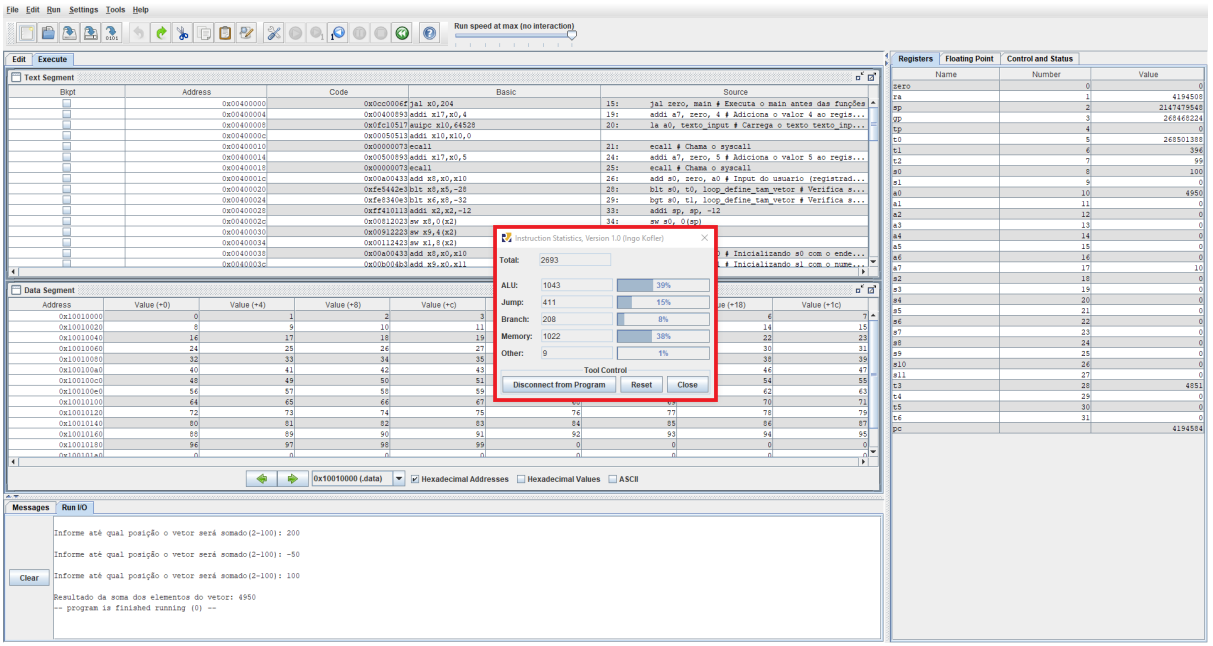


Informando tamanhos inválidos. Após informar o tamanho valido 100, obtém-se o resultado 4950.

Abaixo consta o Instructions Statistics no Programa 02 e seus resultados.



Vetor de tamanho 10



Vetor de tamanho 100

4. Análise dos Resultados:

O primeiro código (programa 01), que usa uma abordagem iterativa, utiliza uma abordagem iterativa, enquanto o segundo código (programa 02) emprega recursão para calcular a soma dos elementos do vetor.

O primeiro código (programa 01) 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.

4.1 Qual solução obtém o melhor desempenho?

O total de comandos executados são praticamente a metade em relação à função recursiva. A execução de ALU's no código recursivo chega a ser de até 41% a mais em comparação ao código não recursivo. Comandos Jump chegam a ser executados praticamente o dobro na função recursiva em relação a não recursiva.

As consultas à memória apresentam o maior salto nas estatísticas, com um salto de até 360% a mais em consultas na memória no código recursivo. Um detalhe interessante é a execução de comandos do tipo Branch a Other, em ambas as funções é permanecido o mesmo resultado de chamadas. Conclui-se que, a solução não recursiva demonstra claramente possuir o melhor desempenho, possuindo uma taxa inferior de processamento de dados, menos pulos para outros trechos de código e poucas consultas na memória.

4.1 Como os dados pioram ou melhoram as soluções, e por que isso acontece?

Os dados pioram drasticamente na execução de instruções de memória quando ocorrem o empilhamento e desempilhamento de registradores na chamada da função recursiva, consequentemente as diversas chamadas ocasionam as várias chamadas de instruções ALU's, resultando em um maior processamento de dados, consultas direto na memória e a maior quantidade de comandos Jump para a chamada da função e realização da soma do vetor.

Uma semelhança entre os dois programas são as execuções de comandos do tipo Branch, que em ambos os casos desempenham a função de um contador, tanto para controlar a finalização da execução da função quanto para determinar limites do valor inserido pelo usuário. Além disso, ambos os programas possuem apenas duas chamadas de ecall, que são instruções do tipo Other e permanecem iguais.