



**Universidade de São Paulo  
Instituto de Ciências Matemáticas e de Computação  
Departamento de Sistemas de Computação**

**SSC0902 - Organização e Arquitetura de Computadores**

Docente: Sarita Mazzini Bruschi  
Monitora: Catarina Moreira Lima

## **TRABALHO PRÁTICO 1: CALCULADORA SEQUENCIAL**

Gabriel Barbosa de Oliveira - gabriel.barbosa@usp.br - 12543415  
Laura Fernandes Camargos - laura.camargos@usp.br - 13692334  
Sandy da Costa Dutra - sandycdutra@usp.br - 12544570  
Nicholas Eiti Dan - nicholas.dan@usp.br - 14600749

São Carlos, 13 de abril de 2025

# 1 Objetivo

O objetivo deste trabalho foi implementar uma calculadora sequencial em Assembly RISC-V que realiza operações aritméticas básicas (adição, subtração, multiplicação e divisão inteira) com funcionalidades adicionais de desfazer operações (undo) e finalização do programa. O projeto visou consolidar os conhecimentos sobre programação em Assembly, manipulação de registradores, controle de fluxo e estruturas de dados, especialmente listas encadeadas.

## 2 Descrição Geral do Programa

A calculadora desenvolvida opera em um loop contínuo, solicitando inicialmente um número e, em seguida, uma operação. As operações suportadas são:

- **+**: Adição
- **-**: Subtração
- **\***: Multiplicação
- **/**: Divisão inteira (com tratamento de divisão por zero)
- **u**: Desfaz a última operação (undo)
- **f**: Finaliza o programa

Cada resultado é armazenado em uma lista encadeada, permitindo que a operação **u** (undo) restaure o estado anterior da calculadora. O programa também inclui tratamento de erros para operações inválidas, números malformados e divisões por zero.

## 3 Estratégia Adotada

### 3.1 Registradores

Os registradores foram utilizados conforme a convenção do RISC-V:

- **s0**: Armazena o resultado atual das operações.
- **a0, a1**: Usados para passagem de parâmetros e retorno de valores.
- **t0-t6**: Registradores temporários para cálculos intermediários.
- **ra**: Armazena o endereço de retorno para chamadas de função.

### 3.2 Lista Encadeada

A lista encadeada armazena os resultados das operações. Cada nó da lista contém:

- **4 bytes**: Valor do resultado.

- 4 bytes: Ponteiro para o próximo nó (ou -1 para NULL).

A label **store\_result** aloca dinamicamente um novo nó e o adiciona no início da lista. A label **undo** remove o nó mais recente e restaura o valor anterior.

### 3.3 Tratamento de Erros

Foram implementadas mensagens de erro específicas para:

- Divisão por zero.
- Operação inválida.
- Número inválido.
- Tentativa de desfazer operação quando não há operações anteriores.

## 4 Principais trechos de código

### 4.1 Leitura e validação dos números inteiros

```

1 read_valid_integer:
2     addi sp, sp, -16                # Reserva espaço na pilha
3     sw ra, 0(sp)                   # Salva endereço de retorno
4     sw s0, 4(sp)                   # Salva registradores salvos
5     sw s1, 8(sp)
6     sw s2, 12(sp)
7
8 read_integer_again:
9     li a7, 4
10    la a0, number_prompt
11    ecall
12
13    # Lê string do usuário
14    li a7, 8
15    la a0, input_buffer
16    li a1, 16                       # Tamanho máximo da entrada
17    ecall
18
19    la s0, input_buffer             # s0 = ponteiro para a string
20    li s1, 0                       # s1 = sinal (0=+, 1=-)
21    li s2, 0                       # s2 = contador de dígitos
22    li t2, 0                       # t2 = valor acumulado
23    li t3, 10                      # t3 = base decimal (10)
24
25    # Verifica sinal negativo
26    lb t0, 0(s0)                   # Carrega primeiro caractere
27    li t1, '+'
28    bne t0, t1, check_first_digit
29    li s1, 1                       # Marca como negativo
30    addi s0, s0, 1                 # Avança para próximo caractere
31
32 check_first_digit:

```

```

33     lb t0, 0(s0)                # Primeiro caractere após sinal
34     li t1, 10                  # ASCII para newline
35     beq t0, t1, invalid_input  # String vazia
36     li t1, '0'
37     blt t0, t1, invalid_input  # Caractere menor que '0'
38     li t1, '9'
39     bgt t0, t1, invalid_input  # Caractere maior que '9'
40
41 process_digits:
42     lb t0, 0(s0)                # Carrega próximo caractere
43     li t1, 10                  # ASCII para newline
44     beq t0, t1, validation_done # Fim da string
45
46     # Verifica se é dígito (0-9)
47     li t1, '0'
48     blt t0, t1, invalid_input
49     li t1, '9'
50     bgt t0, t1, invalid_input
51
52     # Converte ASCII para valor numérico
53     addi t0, t0, -48            # Converte '0'-'9' para 0-9
54
55     # Multiplica acumulador por 10 e adiciona novo dígito
56     mul t2, t2, t3              # t2 = t2 * 10
57     add t2, t2, t0              # t2 = t2 + dígito_atual
58
59     addi s0, s0, 1              # Próximo caractere
60     addi s2, s2, 1              # Incrementa contador
61     li t4, 11                   # Máximo de dígitos
62     bge s2, t4, invalid_input  # Número muito grande
63     j process_digits
64
65 validation_done:
66     # Verifica se teve pelo menos 1 dígito
67     beqz s2, invalid_input
68
69     # Aplica sinal negativo se necessário
70     beqz s1, positive_number
71     neg t2, t2
72
73 positive_number:
74     mv a0, t2                   # Retorna valor convertido
75     lw ra, 0(sp)                # Restaura registradores
76     lw s0, 4(sp)
77     lw s1, 8(sp)
78     lw s2, 12(sp)
79     addi sp, sp, 16
80     jr ra
81
82 invalid_input:
83     li a7, 4                    # Mostra mensagem de erro
84     la a0, invalid_number_error
85     ecall
86
87     j read_integer_again        # Volta a pedir o número

```

Listing 1: Leitura e validação de números completos

## **Explicação detalhada:**

### **1. Estrutura:**

- A função começa salvando registradores importantes na pilha
- Exibe um prompt solicitando a entrada do usuário

### **2. Processamento:**

- Lê a string de entrada do usuário para um buffer
- Verifica se o primeiro caractere é '-' para números negativos
- Percorre cada caractere validando se é um dígito (0-9)

### **3. Conversão:**

- Converte cada caractere ASCII para seu valor numérico
- Constrói o número final multiplicando por 10 e somando cada novo dígito
- Limita a 10 dígitos para evitar overflow

### **4. Tratamento de erros:**

- Rejeita entradas vazias
- Rejeita caracteres não numéricos
- Rejeita números muito longos
- Mostra mensagem de erro específica e repete a solicitação

### **5. Finalização:**

- Aplica o sinal negativo se necessário
- Retorna o valor convertido em a0
- Restaura todos os registradores antes de retornar

## **Características da implementação:**

- Aceita números positivos e negativos
- Verifica cada dígito individualmente
- Limita o tamanho máximo do número
- Fornece feedback claro em caso de erro
- Mantém a consistência mesmo com entradas inválidas

```

CalculadoraRISC-V ) make
java -jar rars1_6.jar nc se1 ae1 me calculadora.s
Digite um número: abc
Erro: valor não é um número inteiro válido
Digite um número: 12.8
Erro: valor não é um número inteiro válido
Digite um número: 2
Digite a operação (+, -, *, /, u, f): █

```

Figura 1: Exemplo de validação de entrada

## 4.2 Lista Encadeada

```

1 store_result:
2     mv t0, a0                # Salva o valor a ser armazenado
3
4     # Aloca memória para novo nó
5     li a0, 8                 # Tamanho do nó (8 bytes)
6     li a7, 9                 # Código de syscall para alocação
7     ecall
8
9     # Configura novo nó
10    sw t0, 0(a0)              # Armazena o resultado
11    lw t1, list_head          # Pega endereço atual da cabeça
12    sw t1, 4(a0)              # Armazena como próximo nó
13
14    # Atualiza cabeça da lista
15    la t2, list_head
16    sw a0, 0(t2)              # Novo nó é agora a cabeça
17    jr ra

```

Listing 2: Armazenamento de resultados

**Explicação:** Implementação da lista encadeada que armazena os resultados das operações para permitir o undo. Cada nó contém o valor e um ponteiro para o próximo nó.

## 4.3 Operação Undo

```

1 undo_operation:
2     la t0, list_head
3     lw t1, 0(t0)              # Pega cabeça atual
4     beq t1, s11, undo_error_case
5
6     lw t2, 4(t1)              # Pega próximo nó
7     beq t2, s11, undo_last_operation
8
9     sw t2, 0(t0)
10    lw s0, 0(t2)              # Restaura resultado anterior
11    jr ra

```

Listing 3: Implementação do undo

**Explicação:** A função remove o nó mais recente da lista e restaura o valor anterior, tratando casos especiais quando não há operações para desfazer.

```

CalculadoraRISC-V } make
java -jar rars1_6.jar nc se1 ae1 me calculadora.s
Digite um número: 2
Digite a operação (+, -, *, /, u, f): -
Digite um número: 3
Resultado: -1
Digite a operação (+, -, *, /, u, f): *
Digite um número: -1
Resultado: 1
Digite a operação (+, -, *, /, u, f): u
Operação desfeita. Resultado anterior: -1
Digite a operação (+, -, *, /, u, f): u
Operação desfeita. Nenhum resultado salvo.
Digite um número: █

```

Figura 2: Exemplo da operação undo em execução

## 5 Exemplos de Execução

### 5.1 Operações Básicas

```

CalculadoraRISC-V } make
java -jar rars1_6.jar nc se1 ae1 me calculadora.s
Digite um número: 3
Digite a operação (+, -, *, /, u, f): /
Digite um número: 2
Resultado: 1
Digite a operação (+, -, *, /, u, f): -
Digite um número: 7
Resultado: -6
Digite a operação (+, -, *, /, u, f): *
Digite um número: -2
Resultado: 12
Digite a operação (+, -, *, /, u, f): +
Digite um número: -13
Resultado: -1
Digite a operação (+, -, *, /, u, f):

```

Figura 3: Execução de operações aritméticas básicas

## 5.2 Tratamento de Erros

```
CalculadoraRISC-V > make
java -jar rars1_6.jar nc se1 ae1 me calculadora.s
Digite um número: abc
Erro: valor não é um número inteiro válido
Digite um número: -2
Digite a operação (+, -, *, /, u, f): a
Erro: operação inválida
Digite a operação (+, -, *, /, u, f): x
Erro: operação inválida
Digite a operação (+, -, *, /, u, f): abc
Erro: operação inválida
Digite a operação (+, -, *, /, u, f): 3
Erro: operação inválida
Digite a operação (+, -, *, /, u, f): 12.3
Erro: operação inválida
Digite a operação (+, -, *, /, u, f): u
Erro: sem operações a serem desfeitas
Digite a operação (+, -, *, /, u, f):
```

Figura 4: Exemplos de tratamento de erros

## 6 Dificuldades Encontradas

- **Manipulação de Strings:** A leitura e validação de números como strings demandou atenção especial para tratar caracteres inválidos e sinais negativos.
- **Gerenciamento de Memória:** A implementação da lista encadeada exigiu cuidado com a alocação de nós para evitar acessos indevidos e erros.
- **Tratamento de Erros:** O tratamento de input inadequado do usuário foi um desafio e adicionou complexidade extra ao código.
- **Liberação da Memória não usada:** Não é o caso com o simulador, mas em uma aplicação real, a memória alocada na heap precisaria ser liberada quando o nó não fosse mais utilizado.

## 7 Conclusão

O projeto foi concluído com sucesso, atendendo a todos os requisitos especificados. A calculadora demonstra eficiência na execução das operações básicas e na manipulação da lista encadeada para a funcionalidade de undo. As dificuldades encontradas foram superadas e a robustez do programa foi confirmada nos testes.