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.

O código do projeto e as instruções para execução estão disponíveis no repositório do github: laurafcamargos/CalculadoraRISC-V

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

```
read_valid_integer:
    addi sp, sp, -16            # Reserva espaço na pilha
    sw ra, 0(sp)                # Salva endereço de retorno
    sw s0, 4(sp)                # Salva registradores salvos
    sw s1, 8(sp)
    sw s2, 12(sp)

read_integer_again:
    li a7, 4
    la a0, number_prompt
    ecall

    # Lê string do usuário
    li a7, 8
    la a0, input_buffer
    li a1, 16                   # Tamanho máximo da entrada
    ecall

    la s0, input_buffer         # s0 = ponteiro para a string
    li s1, 0                    # s1 = sinal (0=+, 1=-)
    li s2, 0                    # s2 = contador de dígitos
    li t2, 0                    # t2 = valor acumulado
    li t3, 10                   # t3 = base decimal (10)

    # Verifica sinal negativo
```

```
    lb t0, 0(s0)                # Carrega primeiro caractere
    li t1, '-'
    bne t0, t1, check_first_digit
    li s1, 1                    # Marca como negativo
    addi s0, s0, 1              # Avança para próximo caractere

check_first_digit:
    lb t0, 0(s0)                # Primeiro caractere após sinal
    li t1, 10                   # ASCII para newline
    beq t0, t1, invalid_input   # String vazia
    li t1, '0'
    blt t0, t1, invalid_input   # Caractere menor que '0'
    li t1, '9'
    bgt t0, t1, invalid_input   # Caractere maior que '9'

process_digits:
    lb t0, 0(s0)                # Carrega próximo caractere
    li t1, 10                   # ASCII para newline
    beq t0, t1, validation_done # Fim da string

    # Verifica se é dígito (0-9)
    li t1, '0'
    blt t0, t1, invalid_input
    li t1, '9'
    bgt t0, t1, invalid_input

    # Converte ASCII para valor numérico
    addi t0, t0, -48            # Converte '0'-'9' para 0-9

    # Multiplica acumulador por 10 e adiciona novo dígito
    mul t2, t2, t3              # t2 = t2 * 10
    add t2, t2, t0              # t2 = t2 + dígito_atual

    addi s0, s0, 1              # Próximo caractere
    addi s2, s2, 1              # Incrementa contador
    li t4, 11                   # Máximo de dígitos
    bge s2, t4, invalid_input   # Número muito grande
    j process_digits

validation_done:
    # Verifica se teve pelo menos 1 dígito
    beqz s2, invalid_input

    # Aplica sinal negativo se necessário
    beqz s1, positive_number
    neg t2, t2

positive_number:
    mv a0, t2                   # Retorna valor convertido
    lw ra, 0(sp)                # Restaura registradores
    lw s0, 4(sp)
    lw s1, 8(sp)
    lw s2, 12(sp)
    addi sp, sp, 16
    jr ra

```

```
invalid_input:
    li a7, 4                    # Mostra mensagem de erro
    la a0, invalid_number_error
    ecall

    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

```
store_result:
    mv t0, a0           # Salva o valor a ser armazenado

    # Aloca memória para novo nó
    li a0, 8            # Tamanho do nó (8 bytes)
    li a7, 9            # Código de syscall para alocação
    ecall

    # Configura novo nó
    sw t0, 0(a0)        # Armazena o resultado
    lw t1, list_head    # Pega endereço atual da cabeça
    sw t1, 4(a0)        # Armazena como próximo nó

    # Atualiza cabeça da lista
    la t2, list_head
    sw a0, 0(t2)        # Novo nó é agora a cabeça
    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

```
undo_operation:
    la t0, list_head
    lw t1, 0(t0)        # Pega cabeça atual
    beq t1, s11, undo_error_case

    lw t2, 4(t1)        # Pega próximo nó
    beq t2, s11, undo_last_operation

    sw t2, 0(t0)
    lw s0, 0(t2)        # Restaura resultado anterior
    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

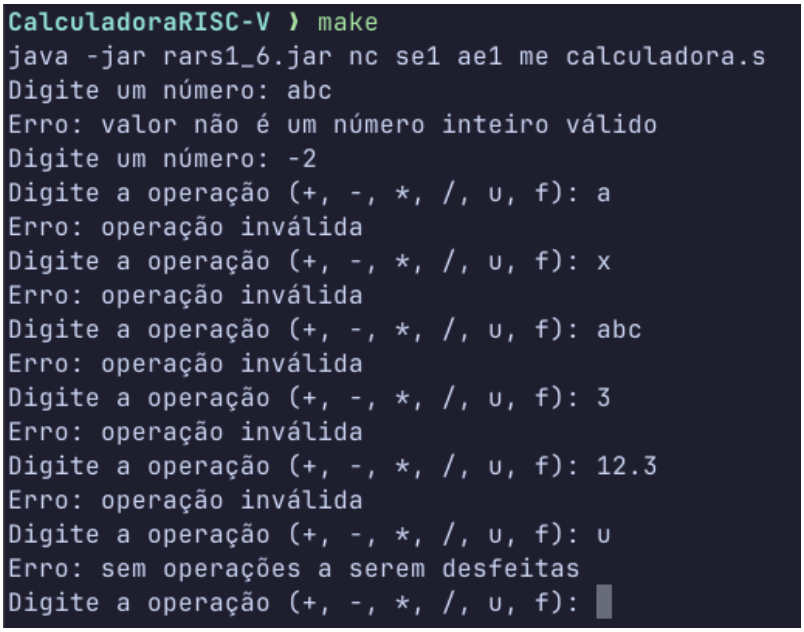

Figura 4: Exemplos de tratamento de erros

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 alocaad 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.