Trabalho Prático 1 - Organização e Arquitetura de Computadores

Calculadora Sequencial

Grupo:

Fernando Valentim Torres	15452340
Laura Pazini Medeiros	15468452
Luysa de Souza Gonçalves	15474077
Pedro Henrique Perez Dias	15484075

Docente:

Sarita Mazzini Bruschi

São Carlos, 4 de abril de 2025

1 - Introdução

Este trabalho visa o desenvolvimento de uma calculadora sequencial, através do uso de Assembly para a arquitetura RISC-V. A calculadora deve ser capaz de realizar operações básicas de adição, subtração, multiplicação e divisão (+, -, *, /), além de outras duas funcionalidades adicionais de desfazer a última operação realizada (u) e de encerrar o próprio funcionamento (f).

2 - Objetivos

O desenvolvimento deste projeto tem como objetivo exercitar os conceitos relacionados ao Assembly RISC-V desenvolvidos em aula, como a sintaxe da linguagem e o funcionamento do fluxo de um programa. Além disso, o projeto busca incentivar os alunos a montar sua própria implementação de uma lista encadeada, que leva ao desenvolvimento de conceitos base na computação como estruturas de dados e alocação dinâmica de memória.

3 - Desenvolvimento

3.1 - Implementação da Lista Encadeada

A estrutura de dados utilizada no programa foi feita de modo que cada elemento da lista armazena seu valor e o endereço do elemento anterior, se assemelhando a uma pilha, permitindo realizar as operações requisitadas pelo trabalho. Cada nó possui 8 bytes — os primeiros 4 armazenam o resultado da operação, e os outros 4 o endereço do nó anterior.

A implementação da lista encadeada foi feita a partir das funções *lista_criar*, *lista_inserir*, *lista_remover_topo* e *lista_topo* permitindo realizar, respectivamente, a alocação dinâmica da lista, a inserção de um elemento na lista, a remoção de um elemento no topo da lista (último elemento), e a recuperação do elemento no topo da lista (último elemento).

3.2 - Funcionamento do programa

O programa inicia com a definição os códigos ASCII correspondentes às operações válidas nos registradores (+, -, *, /, u, f). Em seguida, a função *lista_criar* é chamada para a inicialização da lista encadeada que armazenará os resultados das operações. Essa lista é representada por um ponteiro (*ponteiro_lista*) e um contador de elementos (s1), que permitem controlar o histórico de operações.

Em sua primeira interação, o programa requisita ao usuário que digite um número, seguido da operação desejada. Se a operação for uma das quatro básicas, ele requisita um segundo número, realiza a operação correspondente e armazena o resultado na lista encadeada, chamando a função *lista_inserir*. O resultado é mostrado ao usuário, e o sistema entra em um loop de operações sucessivas com base no resultado anterior.

Caso a operação fornecida pelo usuário seja *undo* (u), a função *lista_topo* é chamada para verificar se há elementos no histórico. Se houver, a função *lista_remover_topo* remove o último resultado, e o programa tenta acessar o novo topo. Se não houver mais valores (ou se tentar desfazer sem histórico), uma mensagem de erro é exibida, e o valor anterior é reinserido para manter a consistência da lista.

Se o usuário tentar realizar uma divisão por zero, o programa detecta isso com uma verificação (*beq*) antes da operação e imprime uma mensagem de erro personalizada.

O programa continua executando o programa em loop, solicitando novas operações, até que o usuário digite f, o que aciona a *syscall* de código 10, que encerra a execução do programa.

3.3 - Tratamento de erros

Na implementação da calculadora realizamos os seguintes tratamentos de erros:

- Para números inválidos, o programa encerra sua execução.
- Para a realização da operação *undo* com a lista vazia, o programa imprime a mensagem “Nao existe valor anterior”.
- Para a realização da operação *undo* com a lista contendo apenas um resultado anterior, o programa mantém o elemento na lista e imprime a mensagem “Nao existe valor anterior”, seguido de seu valor.
- Para operações de divisão por 0, o programa imprime a mensagem "Sem dividir por 0, não sou bobinha(o)(e)".
- Para operadores inválidos, o programa exibe "Operação inválida" e requisita os valores da operação novamente.

3.4 - Dificuldades encontradas

Uma das dificuldades enfrentadas durante o desenvolvimento se deu em relação a inputs de operações inválidas. O impasse consistiu em como fazer o tratamento de operação inválida no começo do programa e no meio do programa, visto que são situações que levam a tratamentos diferentes. A solução encontrada foi utilizar o registrador *s8* como parâmetro de verificação, como uma *flag,* para indicar se o programa já passou pelo loop principal(indicando que o código deve retornar a ele) ou se o input inválido ocorreu na primeira operação (indicando que o código deve voltar à leitura inicial).

Outra dificuldade encontrada pelo grupo ocorreu no tratamento de erros em relação a *inputs* inválidos de números. Isso porque a syscall utilizada para leitura de valores inteiros do teclado aborta o programa ao receber valores diferentes do esperado, fazendo com que o tratamento de erros se tornasse bem complexo e de pouca utilidade.

3.5 - Programa em execução

Em seguida, serão apresentados alguns prints de tela com a execução do programa, em seu fluxo normal.

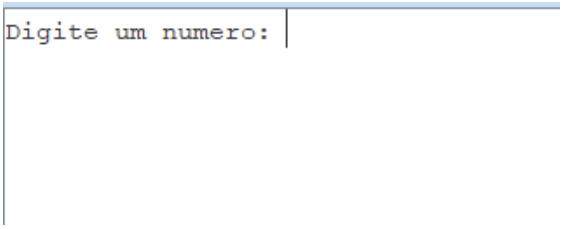

Figura 1. Tela inicial do programa

```
Digite um numero: 3
Escolha uma operacao: *
Digite um numero: 3
9
Escolha uma operacao:
```

Figura 2. Teste da operação multiplicação

```
Digite um numero: 3
Escolha uma operacao: *
Digite um numero: 3
9
Escolha uma operacao: +
Digite um numero: 4
13
Escolha uma operacao:
```

Figura 3. Teste da operação soma

```
Digite um numero: 3
9
Escolha uma operacao: +
Digite um numero: 4
13
Escolha uma operacao: -
Digite um numero: 3
10
Escolha uma operacao:
```

Figura 4. Teste da operação subtração

```
Digite um numero: 4
13
Escolha uma operacao: -
Digite um numero: 3
10
Escolha uma operacao: /
Digite um numero: 2
5
Escolha uma operacao:
```

Figura 5. Teste da operação divisão

```
Escolha uma operacao: -
Digite um numero: 3
10
Escolha uma operacao: /
Digite um numero: 2
5
Escolha uma operacao: u
10
Escolha uma operacao:
```

Figura 6. Teste da operação ‘undo’

```
Escolha uma operacao: /
Digite um numero: 2
5
Escolha uma operacao: u
10
Escolha uma operacao: f
-- program is finished running (0) --
```

Figura 7. Encerramento do programa

Em seguida, serão apresentados alguns prints de tela com a execução do programa, tratando alguns erros que podem ocorrer.

```
Digite um numero: 7
Escolha uma operacao: g
Digite um numero: 6
Operacao invalida
Digite um numero:
```

Figura 8. Validação de operações

```
Digite um numero: 9
Escolha uma operacao: u
Nao existe valor anterior
Digite um numero:
```

Figura 9. Ao iniciar o programa, não é possível desfazer uma operação, visto que não existe uma operação anterior

```
Digite um numero: 2
Escolha uma operacao: -
Digite um numero: 3
-1
Escolha uma operacao: u
Não existe valor anterior
-1
Escolha uma operacao:
```

Figura 10. Operação undo com apenas um resultado na lista

```
Digite um numero: 8
Escolha uma operacao: /
Digite um numero: 0
Sem dividir por 0, nao sou bobinha(o)(e)
```

Figura 11. Validação de divisão por zero

4 - Conclusão

Mediante estes resultados, pode-se afirmar que a realização deste trabalho prático cumpriu com seus objetivos iniciais, proporcionando uma experiência prática importante em relação à linguagem assembly. Além disso, a aplicação possibilitou consolidar os conhecimentos de estruturas de dados e alocação dinâmica de memória. Dessa forma, o trabalho prático consolidou temas abordados durante as aulas teóricas e incentivou o raciocínio e a resolução de problemas.