INTRODUÇÃO

Introducao do projecto

Explicação da divisão entre módulos e o que cada um faz, high level explanation nothing much

Grafico com a interação entre os mesmos

IMPLEMENTAÇÃO - PARTE 1

objetivos

Módulo `MachineStructures.hs` O módulo `MachineStructures` define os elementos fundamentais e os tipos de dados utilizados durante o funcionamento do Interpreter.

`Inst`: Este é um tipo de dados algébrico que representa as instruções da máquina, como operações aritméticas (Add, Sub, Mult), operações lógicas (And, Neg), fluxo de controlo (Branch, Loop), manipulação da Stack e State (Push, Fetch, Store) entre outras. Cada construtor no Inst encapsula uma operação ou comando distinto que a máquina pode executar.

```
data Inst = Push !Integer | Add | Mult | Sub | Tru | Fals | Equ ...
```

```
type Code = [Inst]
```

O `Code` é uma lista de `Inst`

`Stack` : Uma lista de `StackValue`, onde o valor pode ser um número inteiro `(IntVal)` ou um booleano `(TT, FF)` . É utilizada para avaliar expressões e armazenar temporariamente valores durante a execução.

```
type Stack = [StackValue]
```

A `Stack` é uma lista de `StackValues`

`State`: Definido como `(Map.Map String StackValue)`, representa o armazenamento ou a memória da máquina. É essencialmente um map dos nomes das variáveis para seus valores, permitindo que a máquina armazene e recupere dados.

```
STATE   [x=30;y=10]
```

2. Módulo `Assembler.hs` O módulo Assembler é onde se encontra a funcionalidade central do interpretador. Processa uma lista de instruções (Code) juntamente com uma Stack e o State, executando as operações fornecidas. *Este módulo mostra a aplicação prática dos conceitos de programação funcional no tratamento de transformações de estado complexas.*

Lógica de execução

`exec`: Esta função é o coração do interpretador. Ela recebe uma tupla de Código, Pilha e Estado e aplica a primeira instrução da lista de Código à Pilha e ao Estado atuais. A função trata cada tipo de instrução de forma diferente, actualizando a pilha e o estado conforme necessário.

erros: Um tratamento de erros robusto é implementado para detetar erros de tempo de execução, como underflows de pilha ou operações inválidas (à esquerda "Erro de tempo de execução"). Isto assegura que o intérprete se comporta de forma previsível e segura em situações erróneas.

Instruções de fluxo de controlo: A ramificação e o ciclo são fundamentais para a execução condicional e as construções de ciclo. Alteram o fluxo de execução do programa com base nos valores da pilha ou transformam a lista de códigos para implementar a lógica de ciclo.

Resultado

`Instruções e manipulação da pilha: Instruções como Push, Add, Sub e Mult manipulam diretamer`

`Interação de estados: Instruções como Fetch e Store interagem com o estado para recuperar ou`

`Gestão do fluxo de controlo: Branch e Loop usam o estado atual da pilha (particularmente os`

Cenários de exemplo

`Computação aritmética: Para uma sequência de código como [Push 1, Push 2, Add], o interpreta`

`Lógica condicional: Numa instrução Branch, a decisão sobre o caminho a seguir baseia-se no v`

`Tratamento de erros: Se uma instrução Adicionar for encontrada com menos de dois itens na pi`

IMPLEMENTAÇÃO - PARTE 2