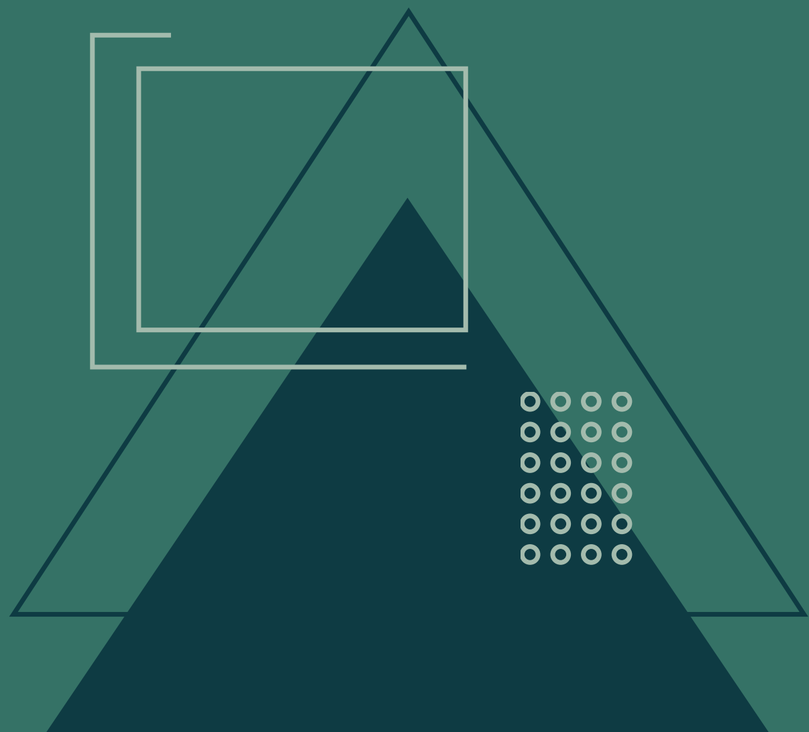
SIMULADOR RISC-V

Grupo:
Emanuel Sol Trindade
Lucas Delourenço
Henrique Ribeiro

SOBRE O SIMULADOR

O simulador foi construído em Python, utilizando a biblioteca PyQt5 e lógicas de Orientação a Objetos (ao fazer uso de classes e objetos)

Funcionalidade básica: O simulador processa o arquivo inserido por completo, seguindo a lógica de um pipeline, e armazenamos os resultados em uma matriz dicionário onde cada linha representa um ciclo e cada coluna um valor associado a um registrador. Guardamos em um vetor cada instrução executada em ordem (incluindo bolhas) . Depois esse vetor é lido e é usada uma lógica para escrever na tabela-diagrama da interface onde é simulado a execução do programa passado.



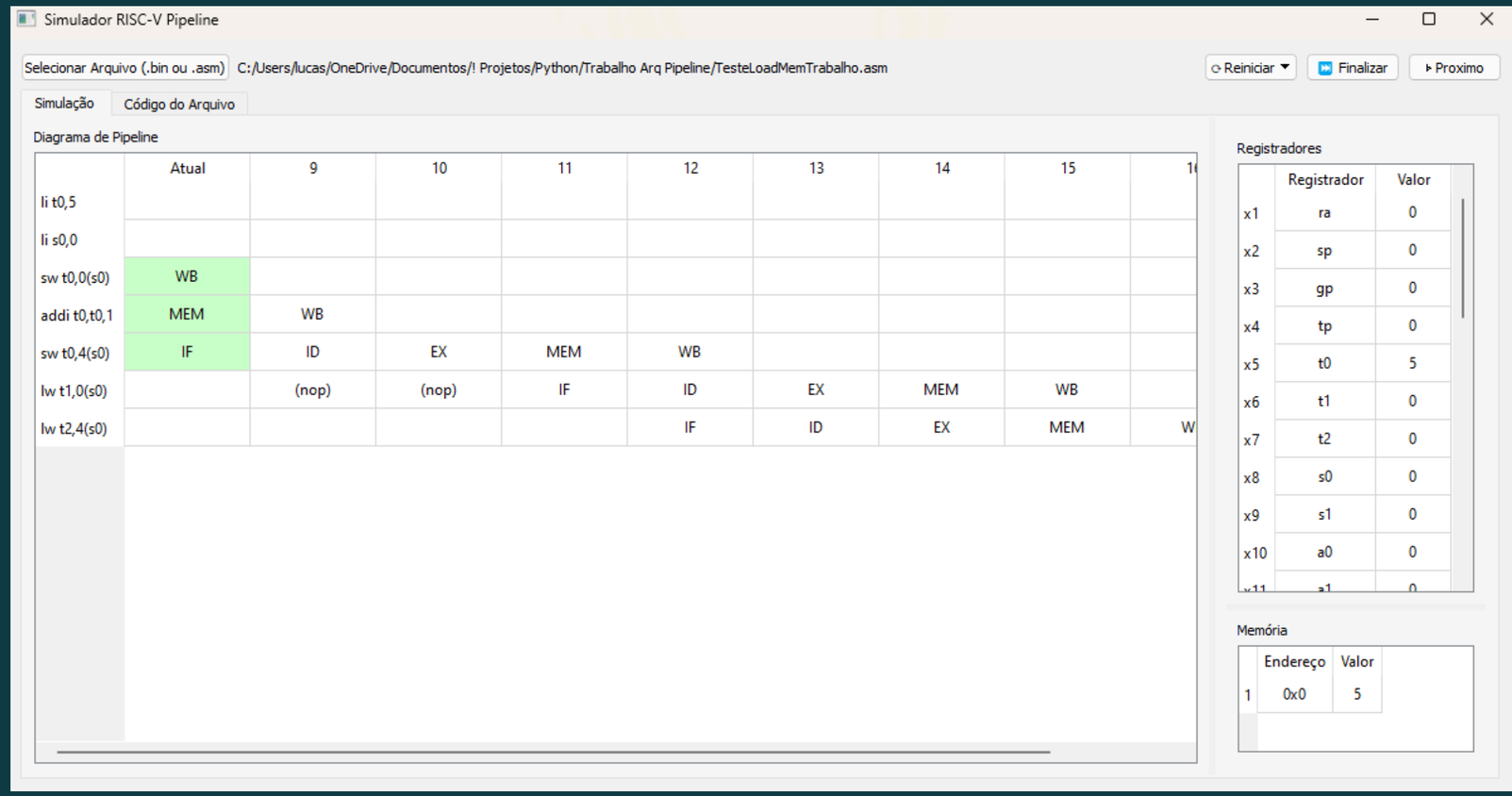
INTERFACE

Tabela Principal: Cada linha representa uma instrução e as colunas representam as etapas das instruções, sendo destacado em tom verde as instruções que serão executadas no ciclo atual.

Tabela Registradores: Cada linha representa o endereço de um registrador e temos duas colunas, a primeira possui o nome do registrador e a segunda contém o valor que está sendo guardado.

Tabela da Memória: Consiste de duas colunas, a primeira representa o endereço de memória e a segunda o valor que está nesse endereço.

Aba para ler o arquivo inserido



INSTRUÇÕES NO SIMULADOR RISC-V

-Cada instrução usada no simulador é representada por uma instância de uma classe;

-Temos a classe Instrucão, que é a classe pai, responsável por definir atributos e comportamentos que toda instrução deve ter;

-Cada tipo de instrução é representada por uma subclasse, que muda seu comportamento caso a opção de fowarding tenha sido selecionada. Os estágios (IF/ID/EX/MEM/WB) são resolvidos dentro de cada subclasse, cada um deles sendo uma função separada, para serem chamadas a cada ciclo do pipeline.

Classe Principal

```python
class Instrucao:
    def __init__(self, texto, linha_idx, fowarding):
        self.estagio_atual = 0  # -1: ainda não entrou -> mudado para 0 para ja começar em IF
        self.concluida = False
        self.executada = False
        self.linha_idx = linha_idx
        self.fowarding = fowarding

    def avancar(self):
        if self.estagio_atual < 4:
            self.estagio_atual += 1
        else:
            self.concluida = True

    def obter_estagio(self):
        estagios = ["IF", "ID", "EX", "MEM", "WB"]
        if 0 <= self.estagio_atual < len(estagios):
            return estagios[self.estagio_atual]
        return ""

    def IF(self):
        self.instrucao, self.texto = self.textoOG.split(' ', 1)
        self.texto.strip()
        return None

    def ID(self, registradores, registradores_forward, labels = None):
        return None

    def executar(self, registradores=None, registradores_foward=None, labels=None):
        return None  # sobrescrito por instruções como beq

    def M(self, memoria, registradores_foward=None):
        return None

    def WB(self, registradores = None):
        return None
```

Subclasse

```python
class InstrucaoTipoR(Instrucao):

    def ID(self, registradores, registradores_forward, labels = None):
        self.rd, rs1, rs2 = self.texto.replace(",", " ").split()
        if(not self.fowarding):
            self.v1 = registradores.get(rs1, 0)
            self.v2 = registradores.get(rs2, 0)
        else:
            self.v1 = registradores_forward.get(rs1, 0)
            self.v2 = registradores_forward.get(rs2, 0)
        return None#self.linha_idx + 1

    def operacao(self, rs1, rs2):...

    def executar(self, registradores, registradores_foward, labels):
        self.resultado = self.operacao(self.v1, self.v2)
        self.executada = True
        if self.fowarding and not self.rd != "zero":
            registradores_foward[self.rd] = self.resultado

    def WB(self, registradores):
        if self.rd != "zero":
            registradores[self.rd] = self.resultado
```

ESTRUTURAS INTERNAS

```
self.text = []
self.instrucoes = []
self.registradores = {"zero":0 , "ra":0,
                      "t1":0, "t2":0, "s
                      "a4":0, "a5":0, "a
                      "s6":0, "s7":0, "s
self.registradoresForwarding = {"zero":0
                      "t1":0, "t2":0, "s
                      "a4":0, "a5":0, "a
                      "s6":0, "s7":0, "s
self.valsPorCiclo = [] #matriz
self.memPorCiclo = []
self.labels = {}
self.memoria = {}
self.ciclo_atual = 0
self.nop_count = 0
self.forwarding = False #por enquanto
```

Para o processamento interno das instruções, são utilizadas as seguintes estruturas:

- Dicionarios:
1. Para registradores
2. Para valores de forwarding
3. Para a memória
- Vetores:
4. Para guardar o .text
5. Para guardar os objetos Instrução, em ordem, que serão executados
- Matrizes:
6. Para valores de registradores por ciclo
7. Para valores da memória por ciclo
- Variaveis Globais:
8. Para saber se o forwarding está habilitado
9. Saber o ciclo atual que estamos
10. Saber a quantidade de bolhas geradas

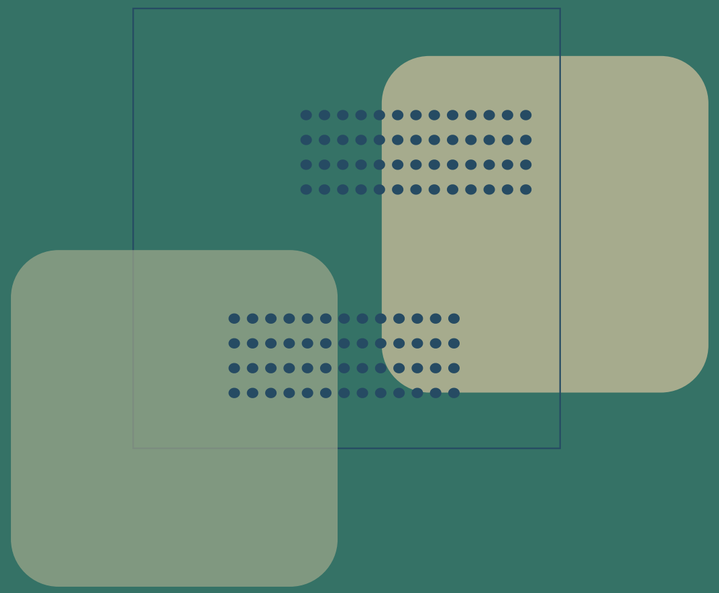
COMO É SIMULADA A EXECUÇÃO?

Quando um arquivo é escolhido e o botão “Iniciar” é pressionado, a função “carregar pipeline” é chamada, nela, o arquivo selecionado é lido e, ao utilizar funções auxiliares, preenchemos o dicionário de labels (guardando como chave o nome e, como valor, o endereço de pulo dela), o vetor text e o de Instruções.

Para o preenchimento do vetor de objetos Instruções, essas são executadas ciclo a ciclo, identificando a posição da próxima instrução dinamicamente, fazendo com que o programa execute todo o arquivo internamente ao ser iniciado e apenas mostre para o usuário o estados das estruturas no ciclo visualizado, simulando, assim, uma execução em tempo real.

Finalmente, é feita uma lógica para escrita e dinamicidade na interface, tratando dos elementos da tabela do pipeline, dos registradores, da memória e da vizualização do código na segunda aba.

FUNCIONAMENTO DO PIPELINE SIMULADO



Como é o comportamento do pipeline que simulamos?

Utilizamos a técnica de bolhas geradas por software, ou seja, essas são geradas antes da etapa IF das instruções que devem receber stall (como mostrado no diagrama-tabela). Elas são geradas sempre que há uma dependência/hazard de dados, com a quantidade de bolhas geradas dependendo se estamos usando forwarding ou não, e quando há uma instrução de desvio, em que sempre é gerada uma bolha, já que a detecção do desvio é feita na etapa ID, simulando o funcionamento da lógica das portas (xor, or, not) que é utilizada em pipelines reais.

Muito Obrigado!