# Simulador de execução fora de ordem com scoreboard - MO401 - 2s2025

Este projeto é um simulador em Python para o algoritmo de Scoreboard, uma técnica de escalonamento dinâmico de instruções para resolver hazards de dados e permitir execução fora de ordem.

O simulador lê um programa em assembly e uma configuração de unidades funcionais, e então simula a execução, mostrando em qual ciclo cada instrução passar por cada um dos quatro estágios (Issue, Read Operands, Execute e Write Result).

## Detalhes da Implementação

Foram implemetadas as **estrutudas de dados**, no qual controlam a simulação, elas são:

1. `instruction_status`: uma tabela que monita o progresso de cada instrução
2. `fus_status`: uma tabela que monitora o estados de todas as UFs
3. `register_status`: uma tabela que rastreia qual UF está designada para escrever em casa registrador

## Documentação das Funções

### Inicialização e Parsing

| Nome da função | Objetivo |
|---|---|
| `ini_register_status` | Criar e inicializar a tabela `register_status` |
| `init_instruction_status(program_name)` | Ler um programa assembly e faz o parser para preencher a tabela `instruction_status` |
| `parser_fus_configs(config_name)` e `init_fus_status(fus_configs)` | Ler a congifuração das UFs e inicializa a `fus_status` |

### Estágios do Scoreboard

`issue(instr, fus_status, register_status)`

- Implementa o estágio de Issue, tenta enviar uma instrução para um UF livre
- **Implementação**: Realiza duas verificações:
    - hazard estrutural: procura uma UF do tipo requerido pela instrução que não esteja ocupada
    - WAW: verifica na tabela `register_status` se alguma outra instrução em execução planeja escrever no mesmo registrador de destino

`read_operands()`

- Implementa o estágio de Read Operands, aguarda até que os operandos de uma instrução estejam disponíveis para leitura

    - **Implementação**: Foca em resolver RAW:
        - Verifica o campo `Qj` e `Qk` da UF, se o campo não for `None`, significa que ele depende do resultado de outra UF
        - O estágio espera até que a UF que vai escrever tenha completado seu estágio de `write_result`

`execute(fu, cycle, inst)`

- Implementa o estágio de Execution, que é um contator de ciclos e a cada ciclo, decrementa o campo de `Cycles_left` da UF. Quando esse valor chegar a zero, a execução é marcada como completar no ciclo específico.

`write_results`

- Implementa o estágio de Write Result
- **Implementação**: antes de escrever o resultado, previne WAR
    - Verifica se alguma outra instrução precisa ler o valor antigo do registrado `Fi` desta instrução
    - Se houver conflito WAR, o estágio de escrita não acontece
    - Caso contrário, a escrita é realizada e ocorre a liberação do registrador na tabela `register_status` e da UF na `fus_status`

## Fluxo de Execução Principal

O simulador opera dentro de loop que avança cada ciclo:

- o loop continua enquanto houver instruções para emitir (`pc < len(instruction_status)`) ou enquanto ohouver instruções em qualquer estágio do pipeline (`has_pending_instructions`)
- os estágios são verificados em ordem reversas dentro de um mesmo ciclo
- foi adicionado uma variável de controle (`liberar_write`) para garantir que as fases de leitura e issue não ocorram no mesmo ciclo de uma escrita

## Testes para os Hazards

### RAW (Read after write) Hazard -- Read Operands

```
fld  f1, 0(x2)
fadd f3, f1, f4
```

Saída esperada:

```
Instruction/Cicle  Issue  Read  Execute  Write
  fld  f1, 0(x2)       1     2        3      4
  fadd f3, f1, f4      2     5        7      8
```

### WAR (Write after read) Hazard -- Write Results

```
fadd f5, f2, f3
fmul f2, f6, f7
```

Saída esperada:

```
Instruction/Cicle  Issue  Read  Execute  Write
  fadd f5, f2, f3      1     2        4      5
  fmul f2, f6, f7      2     3        7      8
```

## WAW (Write after write) Hazard -- Issue

```
fmul f8, f1, f2
fadd f8, f3, f4
```

Saída esperada:

```
Instruction/Cicle  Issue  Read  Execute  Write
  fmul f8, f1, f2      1     2        6      7
  fadd f8, f3, f4      8     9       11     12
```