**Universidade do Vale do Itajaí**

**Escola Politécnica, Ciência da computação**

**Organização de Computadores**

**Prof. Thiago Felski**

**João Vitor Custódio e Miguel Luiz Dalpra Pereira**

# Avaliação 01 – Conflito de Pipeline

**Itajaí**

**09/05/2023**

Este programa feito em Python simula uma pipeline de um processador, aplicando técnicas de "bubble" e "forwarding" para tratar de dependências de dados entre as instruções, o programa utiliza várias funções para funcionar, dentre elas:

- Arrayficador: recebe uma lista de strings, que representam instruções em código binário, e retorna uma lista de strings, onde cada elemento é uma instrução.

- getOpcode, getRd, getRs1 e getRs2: recebem uma instrução em código binário e retornam, respectivamente, o código da operação, o registrador de destino, o primeiro registrador fonte e o segundo registrador fonte.

- descobreTipo: recebe uma instrução em código binário e retorna um número que representa o tipo de instrução: U_TYPE (tipo U), I_TYPE (tipo I), R_TYPE (tipo R) e BS_TYPE (tipo B ou S).

- fmtInstrucao: recebe uma instrução em código binário e retorna uma string que representa a instrução de forma legível, separando os campos de acordo com o tipo de instrução.

- temDependencia recebe uma lista de instruções, representadas por strings em código binário, e dois índices, a e b. Ela verifica se há dependência de dados entre as instruções a e b, ou seja, se a instrução em b utiliza um registrador que foi alterado pela instrução em a. Se houver dependência, a função imprime uma mensagem indicando as instruções que causaram o conflito e retorna True.

- bubbleSemFow e "bubbleComFow: implementam a técnica de "bubble", que consiste em inserir uma ou mais instruções NOP (ADDI zero, zero, zero) no código para evitar conflitos de dependência. A diferença entre as duas funções é que "bubbleComFow" aplica a técnica de "forwarding", que permite que os resultados de um ciclo sejam retro-propagado a estágios anteriores da pipeline antes da atualização do destino tirando a necessidade de verificar hazards duas intruções a frente.

- ciclarFatia: é uma função auxiliar que move os elementos de uma lista entre os índices a e b uma posição à direita, e coloca o elemento da posição b na posição a.

- temProibidonoMeio: verifica se há instruções proibidas (B-type, jal ou jalr) entre duas instruções dadas;

- temDependencianoMeio: recebe os índices de a e b e verifica se há dependência de b nas instruções entre a e b;

- reordenar: reordena duas instruções adjacentes se não houver dependência entre elas e não houver instruções proibidas no meio;

- reordenarComFow: função principal que percorre as instruções e reordena quando possível e quando não possível adiciona um NOP;

Para o cálculo de desempenho das três soluções, foi utilizada as formulas de desempenho disponíveis no material didático da disciplina.

Obtivemos o a quantidade de instruções como 19 para a primeira solução e o calculo seria o seguinte: Texec = 19 * 1,21 * 200 = 4598ps.  1/4598 = 0,000217486

Para a segunda solução, com 15 instruções, ficou da seguinte forma:

Texec = 15 * 1,29 * 200 = 3780ps.  1/3780 = 0,00026455

Para a terceira solução com 14 instruções: Texec = 14 * 1,28 * 200 = 3584ps

1/3584 = 0,000279018.

O multiciclo com 11 instruções teria como resultado o valor 1/85085 = 0,00011699 e chegamos a conclusão que o PIP é 2,39 vezes mais rápido.