

Módulo 7

Princípios básicos de encadeamento



1. Introdução

No final deste módulo os alunos deverão ser capazes de:

- Analisar e descrever organizações encadeadas de processadores elementares
- Caracterizar limitações inerentes a organizações encadeadas (dependências) e conceber potenciais soluções

1.1. Conteúdos e Resultados de Aprendizagem relacionados

Conteúdos	3.2 – Datapath encadeado (pipeline)
Conteduos	3.3 – Dependências de Dados e Controlo
Resultados de	R3.2 – Analisar e descrever organizações encadeadas de processadores elementares
Aprendizagem	R3.3 – Caracterizar limitações inerentes a organizações encadeadas (dependências) e conceber potenciais soluções

2. Material de apoio

A bibliografia relevante para este módulo é constituída pelas secções 4.4 e 4.5 do livro "Computer Systems: a Programmer's Perspective", de Randal E. Bryant e David O'Hallaron.

3. Anomalias na execução encadeada

A execução de uma sequência de instruções numa arquitectura encadeada pode resultar em anomalias que alteram a funcionalidade do código. Estas anomalias resultam de **dependências de dados** ou de **dependências de controlo**. Existem várias técnicas para as evitar e/ou minimizar. A técnica mais elementar corresponde em introduzir bolhas para garantir que estas dependências são resolvidas (empatando (stalling) o pipeline até que a anomalia esteja resolvida). Nos exercícios seguintes é solicitado que identifique as bolhas a introduzir para as várias anomalias, considerando a implementação PIPE- do Y86 descrita na secção 4.5.1 do livro.

Esta implementação tem 5 estágios encadeados: extracção da instrução (F), descodificação (D), execução (E), memória (M) e actualização dos registos (W) - ver diagrama de blocos em anexo.

EXEMPLO

Identifique as dependências presentes no código apresentado abaixo. Identificar as bolhas necessárias para uma execução correcta. O código é apresentado com cada instrução etiquetada. A coluna esquerda da tabela é preenchida com a etiqueta correspondente à instrução apropriada.

I1: irmovl \$10, %eaxI2: rrmovl %ecx, %edxI3: rmmovl %eax, \$0(%edx)

	1	2	3	4	5	6	7	8	9	10	11	12	13
I1	F	D	E	М	W								
12		F	D	E	М	W							
bolha				1	Е	М	W						
bolha					Ì	Е	М	W					
bolha						ightharpoons	Е	М	W				
13			F	D	D	D	D	E	М	W			

Exercício 1

I1: irmovl \$10, %eaxI2: rrmovl %ecx, %edxI3: rmmovl %eax, \$0(%ecx)

1	2	3	4	5	6	7	8	9	10	11	12	13

Exercício 2

I1: rmmovl %eax, \$0(%ebx)

I2: rrmovl %ecx, %edx

i3: mrmovl \$0(%ebx), %edx

1	2	3	4	5	6	7	8	9	10	11	12	13

Exercício 3

Este programa inclui saltos condicionais. Tenha em atenção que:

- a implementação PIPE- prevê que os saltos são sempre tomados;
- a confirmação de um salto condicional só ocorre após a instrução que modifica os códigos de condição ter completado o estágio de execução;
- a instrução de salto condicional gera o sinal Bch no estágio de execução; quando este estágio está completo o sinal Bch é realimentado para o estágio de extracção, podendo a correcção da previsão ser avaliada neste instante;
- se o salto tiver sido mal previsto é necessário reter as instruções que tenham sido extraídas erradamente e recomeçar a extrair no novo endereço.

I1: xorl %eax, %eax

12: jne 14

I3: subl %esi, %edxI4: rrmovl %edx, %esi

15: halt

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

Exercício 4

Este programa inclui o retorno de uma função. Tenha em atenção que o endereço de retorno é lido da pilha, só ficando disponível depois de terminado o estágio de acesso à memória.

I1: call I3I2: halt

I3: ret

14: addl %eax, %eax # nunca é executada

1	2	3	4	5	6	7	8	9	10	11	12	13	14

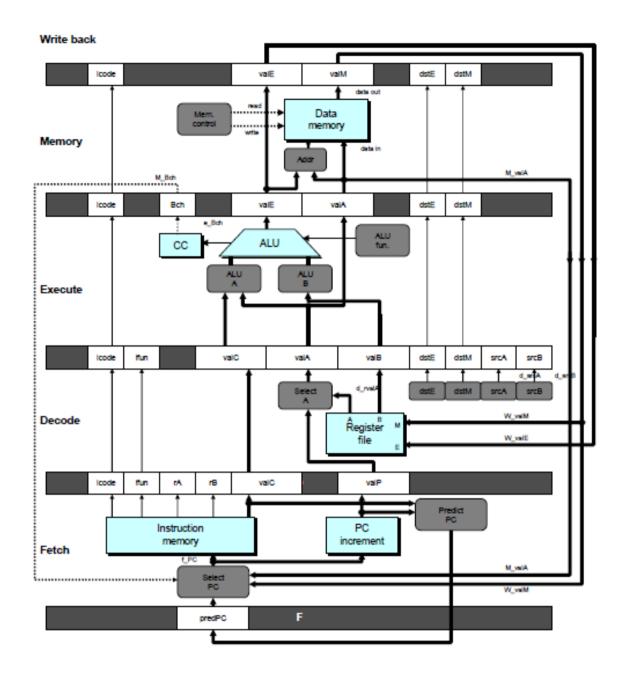


Ilustração 1 - Esquema da organização PIPE- do Y86