Universidade do Minho

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	9.2 – *Datapath* encadeado (*pipeline*)
	9.3 – Dependências de Dados e Controlo
Resultados de Aprendizagem	R9.2 – Analisar e descrever organizações encadeadas de processadores elementares
	R9.3 – 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. Latência e débito

A frequência do relógio de um processador com uma organização encadeada é determinada pela latência da lógica combinatória do estágio mais demorado somada com a latência dos registos que preservam os resultados de cada estágio.

$T_{estagio\_i}$ – latência da lógica combinatória do estágio i

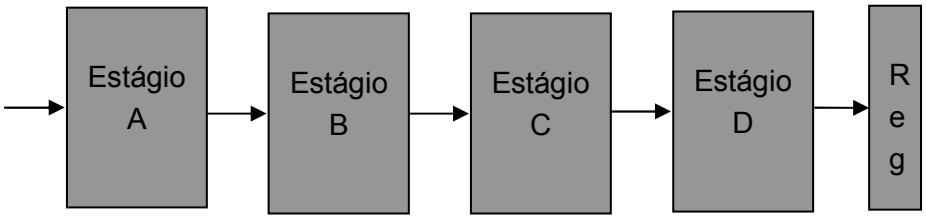
$T_{registo}$ – latência dos registos

$$f = \frac{1}{\max(T_{estagio\_i}) + T_{registo}}$$

O tempo de execução de uma instrução é o produto do número de estágios pelo período do relógio – assumindo que a instrução não é atrasada devido à ocorrência de anomalias.

Exercício 1

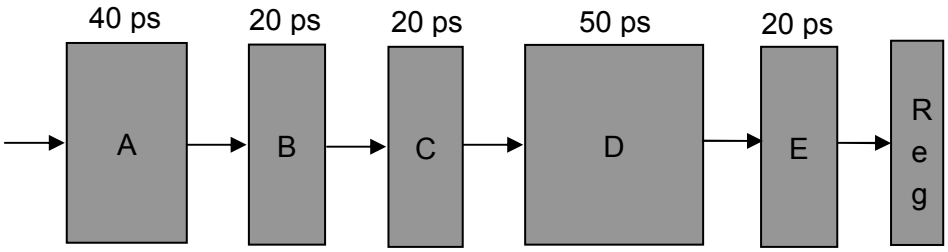
Considere que a lógica combinatória de um processador pode ser decomposta em 4 blocos de igual duração (60 ps) conforme ilustrado na figura.



Sabendo que a latência dos registos é de 20 ps calcule o tempo de execução de uma instrução e o débito para organizações de ciclo único,2 e 4 estágios encadeados.

Exercício 2

Considere que a lógica combinatória de um processador pode ser decomposta em 5 blocos com a duração indicada na figura.



Sabendo que a latência dos registos é de 20 ps calcule:

i) Para uma organização encadeada com 2 estágios como deve ser agrupados os blocos para maximizar o débito? Qual o débito e tempo de execução de cada instrução?

ii) Qual o máximo débito que pode ser obtido e a quantos estágios corresponde.

Exercício 3

Pretende-se analisar o desempenho de um programa com 1000 instruções a executar nas organizações propostas abaixo.

Considere que a lógica combinatória de um processador sequencial tem uma latência de 500 ps. Um bloco de registos tem uma latência de 20 ps. Considere também que a lógica sequencial pode ser dividida em qualquer ponto, permitindo sub-blocos com latências arbitrárias (exigindo-se apenas que a soma das latências de todos os sub-blocos combinatórios seja de 500 ps).

A partir das condições acima pretende-se desenhar várias versões encadeadas, criando sub-blocos de lógica combinatória e acrescentando os registos necessários. Cada novo estágio de *pipeline* criado a partir da versão sequencial incorre 2 custos: tempo de registo e, para este programa, 100 ciclos adicionais devido a dependências de dados e de controlo (causados por eventuais injecções de bolhas (*pipeline staling*)).

i) Qual a latência para uma instrução na organização sequencial? Qual o débito e tempo de execução do programa?

ii) Para organizações com 2, 4 e 10 estágios calcule a latência para uma instrução. Tendo em consideração o custo associado ao *stalling* do *pipeline* calcule o débito e tempo de execução deste programa?

4. Anomalias

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 **dependências de código**. 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, %eax

I2: rrmovl %ecx, %edx

I3: rmmovl %eax, $0(%edx)

	1	2	3	4	5	6	7	8	9	10	11	12	13
I1	F	D	E	M	W								
I2		F	D	E	M	W							
bolha					E	M	W						
bolha						E	M	W					
bolha							E	M	W				
I3			F	D	D	D	D	E	M	W			

Exercício 4

I1: irmovl %10, %eax

I2: rrmovl %ecx, %edx

I3: rmmovl %eax, $0(%ecx)

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

Exercício 5

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 6

Este programa inclui saltos condicionais. Tenha em atenção que:

- a implementaçãp 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

I2: jne I4

I3: subl %esi, %edx

I4: rrmovl %edx, %esi

I5: halt

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

Exercício 7

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 I3
I2:   halt
I3:   ret
I4:   addl %eax, %eax    # nunca é executada
```

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

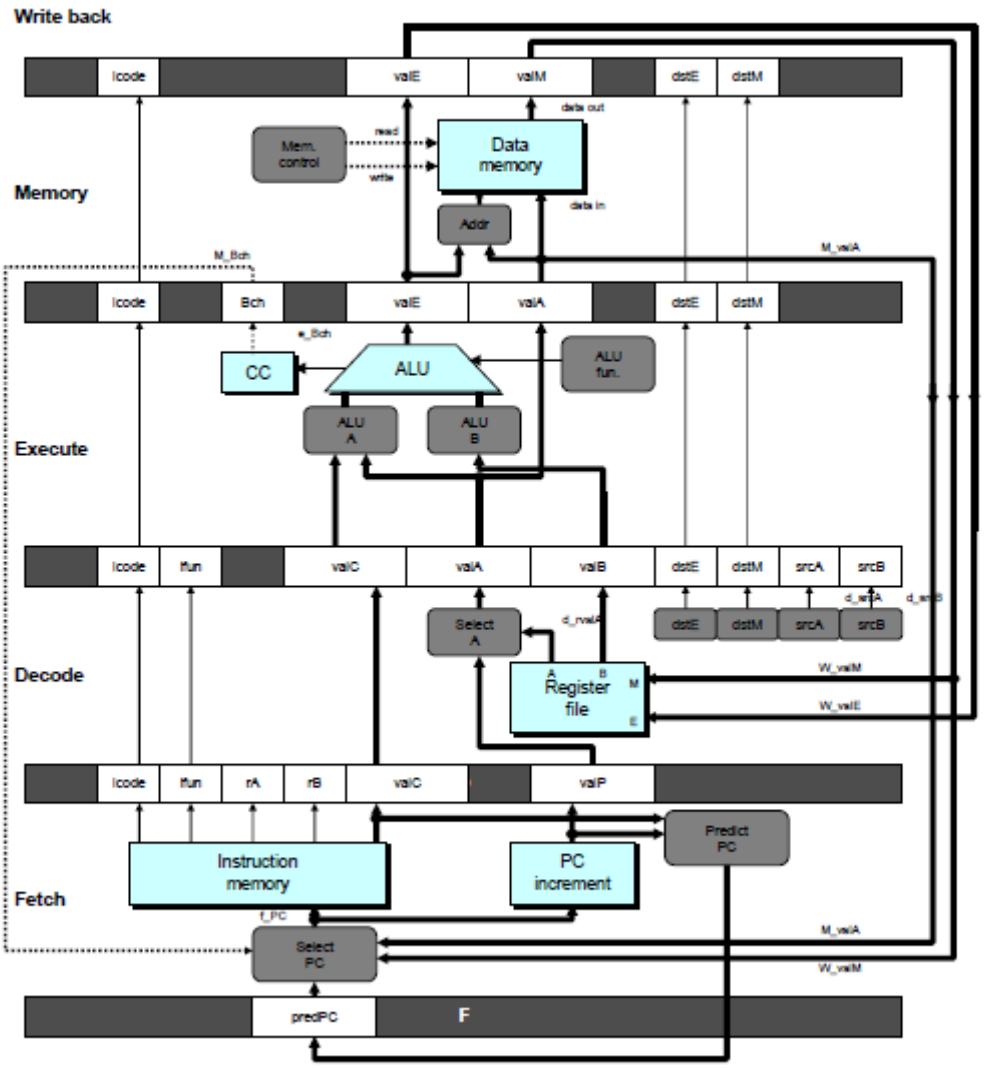


Ilustração 1 - Esquema da organização PIPE- do Y86