Universidade do Minho

Arquitectura de Computadores II

L.E.S.I. – 3º ano

Módulo nº 3

Princípios gerais de encadeamento

1. Introdução

No final deste módulo os alunos deverão ser capazes de:

- avaliar os ganhos/perdas conseguidas com o encadeamento de instruções;
- identificar anomalias que penalizam o desempenho de arquitecturas encadeadas.

2. Material de apoio

A bibliografia relevante para este módulo é a secção 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.

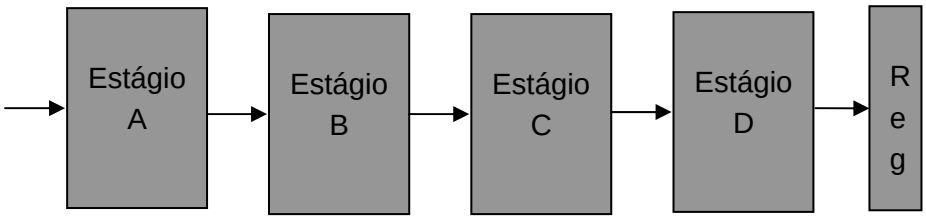
$T_{estagio\_i}$ – latência da lógica combinatória do estágio i
$T_{registo}$ – latência dos registos

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

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.

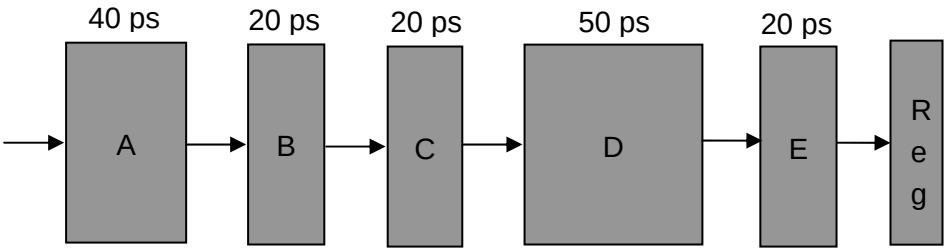


Sabendo que a latência dos registos é de 20 ps calcule o tempo de execução de uma instrução e o débito para:

i) Uma organização de ciclo único;
ii) Uma organização com 2 estágios encadeados;
iii) Uma organização com 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.

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 e ineficiente corresponde em introduzir bolhas para garantir que estas dependências são resolvidas. 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

Identificar as bolhas necessárias para uma execução correcta do código apresentado abaixo. 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 3

```
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 4

```
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 5

Este programa inclui saltos condicionais. Tenha em atenção que:

- a implementaçãp PIPE- prevê que os saltos são sempre tomados;
- a confirmação se um salto condicional deveria ou não ter sido tomado só pode ocorrer 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 6

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