
 <p>Universidade do Minho</p>	<p>Módulo 9</p> <p>Arquiteturas super-escalares</p>	
--	--	---

1. Introdução

No final deste módulo os alunos deverão ser capazes de:

- avaliar as potencialidades e limitações inerentes ao processo de escalonamento de instruções em arquiteturas super-escalares

Para atingir este resultado os alunos deverão dominar os seguintes conceitos:

- Tipos de dependências de dados (RAW/WAR/WAW)
- Escalonamento de instruções
- Execução de instruções fora de ordem
- Renomeação de registos

1.1. Conteúdos e Resultados de Aprendizagem relacionados

Conteúdos	3.4 – Arquiteturas super-escalares
Resultados de Aprendizagem	R3.4 – Analisar o impacto de múltiplas unidades funcionais no desempenho

2. Exemplo

1. Considere o seguinte extrato de programa escrito em Y86:

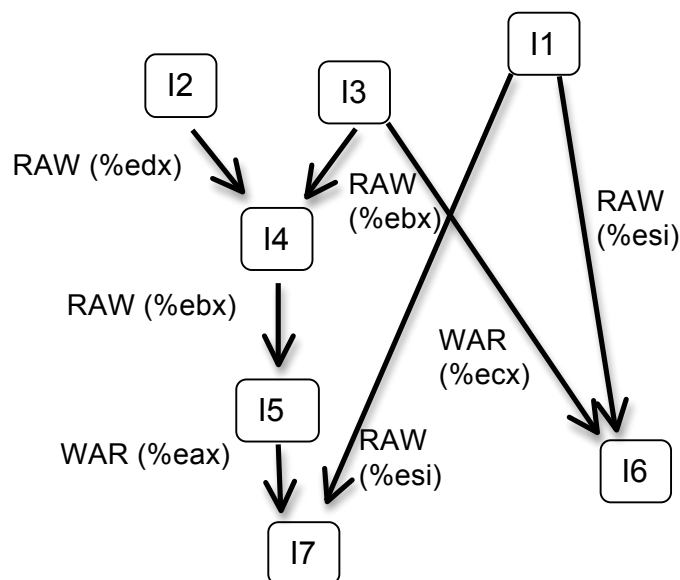
```

I1:    irmovl $8,%esi
I2:    mrmovl 0(%ecx), %edx
I3:    mrmovl 4(%ecx), %ebx
I4:    addl %edx, %ebx
I5:    rmmovl %ebx, 0(%eax)
I6:    addl %esi, %ecx
I7:    addl %esi, %eax

```

1.1 Apresente um grafo com as dependências de dados existentes neste programa. Indique também o tipo de cada dependência.

Resolução:



1.2 Indique como é que este programa seria escalonado numa arquitetura com a capacidade de executar duas instruções por ciclo, um acesso à memória e uma operação aritmética por ciclo. Considere que as instruções são executadas pela ordem do programa e que as dependências RAW load/use originam um ciclo de *stall*. Qual o CPI ideal desta arquitetura e qual o CPI médio obtido na execução do programa?

Resolução:

Ciclo	Aritmética/salto	Acesso à memória
1	<code>irmovl \$8,%esi</code>	<code>mrmovl 0(%ecx), %edx</code>
2		<code>mrmovl 4(%ecx), %ebx</code>
3		
4	<code>addl %edx, %ebx</code>	
5	<code>addl %esi, %ecx</code>	<code>rmmovl %ebx, 0(%eax)</code>
6	<code>addl %esi, %eax</code>	

Apenas as instruções independentes (ou seja, sem dependências entre si) podem ser executadas num mesmo ciclo. O CPI ideal é 0,5 pois esta arquitetura pode executar duas instruções por ciclo. Neste programa são executadas 7 instruções em 6 ciclos, logo o CPI médio é $6/7 = 0,86$

1.3 Repita o exercício anterior, considerando agora uma microarquitetura com a capacidade de executar as instruções fora da ordem do programa.

Resolução:

Ciclo	Aritmética/salto	Acesso à memória
1	<code>irmovl \$8,%esi</code>	<code>mrmovl 0(%ecx), %edx</code>
2		<code>mrmovl 4(%ecx), %ebx</code>
3	<code>addl %esi, %ecx</code>	
4	<code>addl %edx, %ebx</code>	
5		<code>rmmovl %ebx, 0(%eax)</code>
6	<code>addl %esi, %eax</code>	

Observando o grafo de dependências (alínea 1.1) verificamos que a instrução I6 depende apenas de I1 e I3, logo pode iniciar a execução logo que estas instruções sejam executadas (ou seja, antes de I4 e de I5). Note que esta alteração no escalonamento não conduz a uma melhoria no tempo de execução deste programa.

1.4 Assuma que pode utilizar um número adicional de registos designados por %r8 ... %r15 e que as instruções podem utilizar 3 operandos (e.g, `addl rs1, rs2, rd`). Apresente novamente o escalonamento deste programa.

Resolução:

Ciclo	Aritmética/salto	Acesso à memória
1	<code>irmovl \$8,%esi</code>	<code>mrmovl 0(%ecx), %edx</code>
2	<code>addl %esi, %ecx, %r8</code>	<code>mrmovl 4(%ecx), %ebx</code>
3	<code>addl %esi, %eax, %r9</code>	
4	<code>addl %edx, %ebx</code>	
5		<code>rmmovl %ebx, 0(%eax)</code>

Esta alteração permite remover as dependências WAR, originando mais flexibilidade no escalonamento, nomeadamente, I7 passa a poder ser executada antes de I5 pois já não altera o registo %eax.

Comentário final: Note que em todos estes exercícios as dependências RAW são sempre escalonadas da mesma forma. Por muito sofisticada que seja a arquitetura, este tipo de dependências constitui sempre um limite à exploração de paralelismo ao nível das instruções.

Ciclo	Aritmética/salto	Acesso à memória
1	<code>irmovl \$8,%esi</code>	<code>mrmovl 0(%ecx), %edx</code>
2		<code>mrmovl 4(%ecx), %ebx</code>
3		
4	<code>addl %edx, %ebx</code>	
5		<code>rmmovl %ebx, 0(%eax)</code>

3. Exercícios

Considere o seguinte programa escrito em Y86:

```
irmovl 4, %esi
irmovl $0, %eax
irmovl -1024, %ecx
soma: mrmovl 0(%ecx), %edx
      addl %edx, %eax
      rmmovl %eax, $1000(%ecx)
      addl %esi, %ecx
      jnz soma
```

Exercício 1

Identifique as dependências de dados existentes neste programa e represente essas dependências através de um grafo de dependências.

Será possível reordenar as operações por forma a diminuir o número de bolhas injetadas na execução deste programa num processador com a micro-arquitetura Y86 PIPE?

Exercício 2

Apresente o escalonamento deste programa numa arquitetura superescalar de duas vias, com a capacidade de executar uma operação aritmética/salto simultaneamente com um acesso à memória. Qual o CPI obtido na execução do programa. Faça o escalonamento apenas para as instruções executadas no ciclo. Para simplificar o exercício considere que as dependências RAW *load/use* **não** originam um ciclo de *stall*.

Ciclo	Aritmética/salto	Acesso à memória
1		
2		
3		
4		
5		

Exercício 3

Desdobre o corpo do ciclo 2 vezes e efetue novamente esse escalonamento, reordenando as instruções por forma a melhorar escalonamento. Pode utilizar um número adicional de registos designados por %r8, %r9, etc..., e instruções com 3 operandos (e.g, addl rs1, rs2, rd). Qual o CPI obtido agora na execução do programa.

Ciclo	Aritmética/salto	Acesso à memória
1		
2		
3		
4		
5		
6		
7		