**Lista 4**

Nelson Gomes Neto

1. **Considere um sistema de computação que possui um CPU com palavras de 32 bits, cujo repertório é o do processador MIPS. Considere inicialmente que a CPU foi implementada com a técnica multi-ciclo e que a quantidade de ciclos de cada instrução é dada na tabela abaixo. Qual o tempo de execução (em ciclos de clock) e o CPI para uma implementação multi-ciclo executar o programa abaixo? Suponha que o registrador $t4 possui o valor 100 e os conteúdos das posições de memória 100 e 104 são respectivamente 5 e 10.**

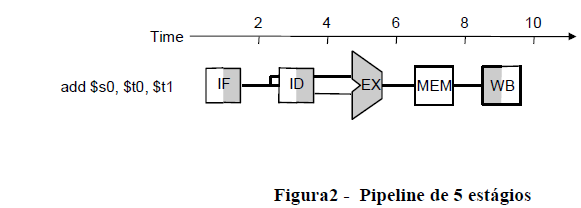
Tabela 1 – Ciclos por instrução para implementação multi-ciclo

|  |  |
| --- | --- |
| **Instrução** | **Nr. de ciclos** |
| Aritméticas e deslocamento | 4 |
| Load word – lw | 5 |
| Store word - sw | 4 |
| Jump | 3 |
| Beq | 4 |
| Lui | 3 |
| Jal e Jr | 4 |

|  |  |
| --- | --- |
| **lw $t1, 4($t4) +5 lw $t3, 8($t4) +5 lw $t2, 12($t4) +5 lw $t0, 0($t4) +5 bne $t0, $t1, endX +4** sub $t0, $t0, $t1  sub $t3, $t3, $t2  addi $t0, $t0, 1  addi $t2, $t2, 1  sub $t2, $t2, $1  sub $t1, $t1, $3 j endY | **endX:**  **sub $t3, $t3, $t2 +4**  **add $t0, $t0, $t1 +4**  **addi $t3, $t3, -1 +4**  **addi $t2, $t2, -1 +4**  **add $t2, $t2, $t1 +4**  **add $t1, $t1, $t3 +4**  **endY:**  **sw $t0, 0($t4) +4**  **sw $t1, 4($t4) +4**  **sw $t2, 12($t4) +4** |

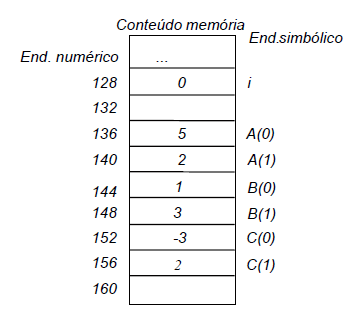
As linhas executadas estão marcadas de vermelho, somam: **60 ciclos de clock**, e totalizam **14 instruções**. Resultando num **CPI = 60 / 14 = 4.286**.

1. **Considere agora que a CPU foi implementada com um pipeline de 5 estágios conforme mostrado na Figura 1 (abaixo) e deve executar o mesmo programa da questão anterior.**
2. **Calcule o tempo de execução (em ciclos de clock) e o CPI no pipeline da Figura 1 considerando que NOP’s são inseridos na ocorrência de conflito de dados e de controle. Assuma o tempo de execução das instruções nos estágios do pipeline conforme dado na Figura 1. Qual o speed-up da implementação em pipeline em comparação com uma implementação multi-ciclo?**  
   Contando novamente a quantidade de ciclos, tendo 5 ciclos como o maior tamanho de uma instrução: Teremos **27 ciclos, e 14 instruções**. Resultando num **CPI = 27 / 14 = 1.9286**. E teve um speed-up de: **(60 / 14) / (27 / 14) = 2.22**, sendo assim, é 2.22 vezes melhor que a implementação multiciclo.
3. **Aplique otimizações para resolver todos os conflitos de dados e de controle. Qual(is) a(s) otimização(ões) que você sugere para melhorar o desempenho do pipeline? Calcule o tempo de execução e o CPI com a(s) otimização(ões) sugerida(s). Qual o speed-up em comparação com a implementação multi-ciclo e com a implementação em pipeline sem otimizações?**  
   Podemos melhorar o desempenho utilizando curto-circuito. Com isso, podemos reduzir para **21 ciclos**, reduzindo o **CPI para: 21 / 14 = 1.5**. Melhorando o speed-up ainda mais, para: **(60 / 14) / (21 / 14) = 2.8571**, se comparado com o multiciclo, e para: **1.9268 / 1.5 = 1.285**.

****

***Figura 1.*** *Pipeline de 5 estágios.*

1. **Um problema sério na implementação em pipeline é a ocorrência de conflitos de dados. Descreva 1 técnica de software e 1 de hardware que resolvem este tipo de conflito detalhando se o desempenho é degradado ou não e se algum suporte adicional (arquitetura ou hardware) se faz necessário.**  
   **Software**: Adição de nop’s. Degrada o desempenho do programa, pois adiciona mais instruções.  
   **Hardware**: Adicionando curto-circuito. O desempenho continua sendo o mesmo, mas é necessário adicionar um conjunto de itens no hardware para que o curto-circuito funcione corretamente.
2. **Considere agora o programa em linguagem de montagem do MIPS descrito a seguir. Os vetores A, B e C possuem dimensão igual a 2. A localização das variáveis na memória pode ser visualizada na Figura 2, a qual contém os valores numéricos e simbólicos de cada endereço, bem como o correspondente conteúdo de memória. Por exemplo, a variável i possui o endereço numérico 128 e o seu valor inicial é 0. O vetor A possui o endereço inicial igual a 136 e o valor de A(0) é 5.**



***Figura 2.*** *Localização e valores iniciais das variáveis na memória*

|  |  |
| --- | --- |
| **lui $3, 128 +3**  **srl $3, $3, 16 +4**  **lw $1, 0($3) +5**  **lui $2, 1 +3**  **srl $2, $2, 16 +4**  **beq $2, $1, fim +4**  **loop: lw $1, 8($3) +5 +5**  **lw $4, 24($3) +5 +5**  **beq $1, $4, oper\_b +4 +4**  **j oper\_c +3**  **oper\_b: lw $4, 16 ($3) +5**  **oper\_c: jal swap +4 +4**  **add $5, $5, $2 +4 +4** | **addi $3, $3, 4 +4 +4**  **slti $6, $5, 2 +4 +4**  **bne $6, $0, loop +4 +4**  **fim: break +0**  **swap: xor $1, $1, $4 +4 +4**  **xor $1, $1, $4 +4 +4**  **xor $4, $1, $4 +4 +4**  **xor $4, $1, $4 +4 +4**  **sw $1, 8($3) +4 +4**  **jr $31 +4 +4** |

Esse código teve o loop executado três vezes, onde oper\_c foi executado três vezes, oper\_c uma vez, e swap três vezes.

* 1. **Calcule o tempo de execução e o CPI do programa acima para uma implementação multi-ciclo da CPU**Executando o código, teremos **147 ciclos**, resultando num CPI de **147 / 36 = 4.08**.
  2. **Calcule o tempo de execução e o CPI do programa acima no pipeline da Figura 1. Para resolver os conflitos de dados e de controle insira NOP´s. Qual o speed-up em comparação com a implementação multi-ciclo?**Com pipeline, teremos **97 ciclos**, resultando num CPI de **97 / 36 = 2.69**. Conseguiremos um speed-up de: **(147 / 36) / (97 / 36) = 1.51** vezes mais rápido.  
     A adição de nop’s faz com que o código fique da seguinte forma:

|  |  |
| --- | --- |
| lui $3, 128  *nop*  *nop*  srl $3, $3, 16  *nop*  *nop*  lw $1, 0($3)  lui $2, 1  *nop*  *nop*  srl $2, $2, 16  *nop*  *nop*  beq $2, $1, fim  *nop*  *nop*  *nop*  loop: lw $1, 8($3)  lw $4, 24($3)  *nop*  *nop*  beq $1, $4, oper\_b  *nop*  *nop*  *nop*  j oper\_c  *nop*  *nop*  *nop* | oper\_b: lw $4, 16 ($3)  oper\_c: jal swap  *nop*  *nop*  *nop*  add $5, $5, $2  addi $3, $3, 4  slti $6, $5, 2  bne $6, $0, loop  *nop*  *nop*  *nop*  fim: break  swap: xor $1, $1, $4  *nop*  *nop*  xor $1, $1, $4  *nop*  *nop*  xor $4, $1, $4  *nop*  *nop*  xor $4, $1, $4  sw $1, 8($3)  jr $31 |

* 1. **O tempo calculado acima pode ser reduzido? Qual(is) a(s) técnicas de resolução de conflito que você sugere? Calcule o tempo de execução e o CPI com a(s) otimização(ões)  sugerida(s). Qual o speed-up obtido com as otimizações?**Pode ser reduzido. Podemos utilizer curto-circuito, especulação e reordanação do código. Com curto-circuito:  
     Teremos **73 ciclos**, resultando num CPI de **73 / 36 = 2.02**. E um speed-up de: **(147 / 36) / (73 / 36) = 2.013** vezes melhor que o multiciclo, e **(97 / 36) / (73 / 36) = 1.32** vezes melhor que o pipeline não otimizado. A implementação do curto-circuito faz com o código fique da seguinte forma:

|  |  |
| --- | --- |
| lui $3, 128  srl $3, $3, 16  lw $1, 0($3)  lui $2, 1  srl $2, $2, 16  beq $2, $1, fim  *nop*  *nop*  *nop*  loop: lw $1, 8($3)  lw $4, 24($3)  beq $1, $4, oper\_b  *nop*  *nop*  *nop*  j oper\_c  *nop*  *nop*  *nop* | oper\_b: lw $4, 16 ($3)  oper\_c: jal swap  *nop*  *nop*  *nop*  add $5, $5, $2  addi $3, $3, 4  slti $6, $5, 2  bne $6, $0, loop  *nop*  *nop*  *nop*  fim: break  swap: xor $1, $1, $4  xor $1, $1, $4  xor $4, $1, $4  xor $4, $1, $4  sw $1, 8($3)  jr $31 |

1. **Considerando novamente o programa MIPS da questão 4 responda as questões a seguir. Geralmente se pode identificar 3 tipos de dependências em programas. Explique essas dependências e encontre pelo menos um exemplo de cada no programa.  
   Read-After-Write (RAW):** Ocorre quando uma instrução precisa ler um valor depois de ter sido feita a escrita. Ex:  
   lui $2, 1  
   srl $2, $2, 16 🡨 Nesse momento, ele precisa ler o valor salvo em $2, mas dependendo da implementação, se não existir o tratamento correto, o valor desejado ainda não estará salvo em $2 no momento em que é executada a parte de leitura da instrução “srl”.  
   **Write-After-Read (WAR):** Ocorre quando uma instrução escreve antes de outra instrução conseguir ler o dado. Ex:  
   xor $1, $1, $4  
   xor $4, $1, $4  
   **Write-After-Write (WAW):** Ocorre quando uma instrução tenta escrever antes de outra, quando devia escrever depois. Ex:  
   xor $4, $1, $4  
   xor $4, $1, $4
2. **Duas técnicas usadas pelos compiladores são: escalonamento estático de instruções e loop unrolling. Mostre como o loop do programa da questão 4 pode ser desenrolado e como as instruções podem ser escalonadas. No novo código mostre as mudanças realizadas e descreva como e porque tais mudanças podem melhorar o desempenho.  
   Loop unrolling:** Essa técnica envolve remover ou reduzir a necessidade de um incremento num índice (para o loop), e remover ou reduzir a necessidade de comparações (para verificar se chegou no fim do loop, por exemplo); isso faz com que a execução do código seja mais direta. Seguem as alterações no código:

|  |  |
| --- | --- |
| lui $3, 128  srl $3, $3, 16  lw $1, 0($3)  lui $2, 1  srl $2, $2, 16  beq $2, $1, fim  loop: lw $1, 8($3)  lw $4, 24($3)  beq $1, $4, oper\_b  j oper\_c  oper\_b: lw $4, 16 ($3)  oper\_c:  swap: xor $1, $1, $4  xor $1, $1, $4  xor $4, $1, $4  xor $4, $1, $4 | sw $1, 8($3)  addi $3, $3, 4  lw $1, 8($3)  lw $4, 24($3)  beq $1, $4, oper\_b  j oper\_c  oper\_b: lw $4, 16 ($3)  oper\_c:  swap: xor $1, $1, $4  xor $1, $1, $4  xor $4, $1, $4  xor $4, $1, $4  sw $1, 8($3)  addi $3, $3, 4  fim: break |

**Escalonamento estático de instruções:** Essa técnica envolve reordenar o código, de forma que sejam evitados “hazards”, sem que o código perca seu propósito original (ou seja, reordenar apenas instruções não conflitantes).

1. **Os processadores superescalares usam várias técnicas para explorar paralelismo de instrução ILP. Considerando tais técnicas responda às questões a seguir. Explique como funciona a técnica de previsão dinâmica de desvio baseada em preditores de 2 bits. Use o programa da questão 4 para mostrar como a previsão funciona. Quais as vantagens e desvantagens desta técnica?**A previsão dinâmica é feita através de uma análise de fluxo de desvios, para melhorar a chance de uma predição correta. Ela funciona utilizando dois bits como histórico dos dois últimos branchs, aumentando a taxa de acerto. Como, a única coisa que acontece quando ele erra a predição, é que o processador fica ocioso (o mesmo que faria quando não estava otimizado), então a única desvantagem é a complicação de implementação.  
   Isso pode ser utilizado no código da questão 4 pois na segunda comparação do loop, ele assumiria que ele faria o mesmo, e acertaria, aumentando assim a velocidade da execução do código.
2. **Explique como os conflitos do tipo RAW, WAR e WAW são resolvidos pelo algoritmo básico de escalonamento dinâmico (Tomasulo).  
   RAW:** Enquanto uma instrução está em conflito, outra instrução não conflitante é permitido ser executado.  
   **WAR:** As operações em conflito têm o registrador (que causa o conflito) renomeado.  
   **WAW:** O mesmo do WAR.
3. **De que maneira o reorder buffer ajuda no escalonamento dinâmico com especulação? Quais as principais modificações devem ser feitas em um processador para se introduzir suporte para especulação?**No momento em que é descoberto o erro de predição, o reorder buffer permite o flush das instruções especulativas.  
   Principalmente: Uma unidade de reserva, para armazenar os operandos enquanto a instrução não pode ser executada. E uma unidade de commit, que conterão buffers para reordenação das escritas.