Lista de Exercício 02

1. Processadores monociclos são baseados em um único ciclo de clock, grande o suficiente para acomodar todas as instruções a serem consideradas. Toda a instrução começa sua execução em uma transição ativa do clock e completa a execução na próxima transição ativa do sinal do clock.
   Já os processadores multiciclo, cada passo de execução gasta um período de clock. A implementação multiciclo permite que uma unidade funcional seja utilizada mais de uma vez por instrução uma vez que ela está sendo usada em ciclos diferentes do clock.

2. Introduzir registradores, pois eles vão funcionar como mini buffers que guardarão os resultados do processamento de cada componente, flags que servirão de controladores extras para manipular os registradores extras e os componentes, tratamento de erros para evitar que haja conflito ao utilizar o pipeline.

3. Loop: 1 - subi $t2, $t2, 4
   2 - lw $t1, 0($t2) - Stall, t2 necessita que o valor seja calculado no subi da linha1.
   3 - add $t3, $t1, $t4 – Stall, aguardando o valor ser carregado em t1 pelo lw da linha2.
   4 - add $t4, $t3, $t3 – Stall, valor de t3 ainda esta sendo calculado no add da linha3.
   5 - sw $t4, 0($t2) – Stall, aguardando o valor de t4 ser calculado no add da linha4.
   6 - beq $t2, $0, loop – Stall, necessário que valor seja carregado para t2 no sw da linha5.

O tempo sem pipeline é de 38ns.

O tempo com pipeline é de 33ns.

Speedup é de $^{38}/_{33} = 1{,}15$ vezes mais rápido.

4. Com a utilização do Bypassing, não haverá mais conflitos, logo, não haverá stalls.

O tempo de execução do loop vai para 18ns

Speedup é de $^{38}/_{18} = 2{,}1$ vezes mais rápido.

5. Conflito de dados:
   RAW - sub.d e div.d; para ser executado o sub da linha 2 é necessário quer o valor de F1 seja calculado o div na linha 1
   WAW - s.d e sub.d; dependência entre a linha 3 e 2, registrador F5
   WAR - add.d e sub.b; F5 é lido no sub na linha 2 e depois é escrito no add da linha 4
   Conflitos na linha 5:
   WAW com a linha 3, F4 é sobrescrito novamente
   RAW com a linha 4, F5 precisa ser calculado em add.d
   RAR com a linha 4, F6 é lido novamente.

6.

7.

Quando está sempre acertando:

$$\text{CPU execution time} = (CPU\ clock\ cycles + Memory\ stall\ cycles) * Clock\ cycle$$
$$= (IC * CPI + 0) * Clock\ cycle$$
$$= IC * 1.0 * Clock\ cycle$$

Situação:

$$Memory\ stall\ cycles = IC * \frac{Memory\ accesses}{Instruction} * Miss\ rate * Miss\ penalty$$
$$= IC * (1 + 0.5) * 0.02 * 25$$
$$= IC * 0.75$$
$$CPU\ execution\ time_{\text{cache}} = (IC * 1.0 + IC * 0.75) * Clock\ cycle$$
$$= 1.75 * IC * Clock\ cycle$$

Resultando em:

$$\frac{CPU\ execution\ time_{cache}}{CPU\ execution\ time} = \frac{1.75 * IC * Clock\ cycle}{1.0 * IC * Clock\ cycle} = 1.75$$

8. a) Quando o sistema escreve para uma zona de memória, que está contida no cache, escreve a informação, tanto na linha específica do cache como na zona de memória ao mesmo tempo. Este tipo de caching providencia pior desempenho do que Write-Back Cache, mas é mais simples de implementar e tem a vantagem da consistência interna, porque o cache nunca está dessincronizada com a memória como acontece com a técnica Write-Back Cache.

   b) Usando esta técnica a CPU escreve dados diretamente no cache, cabendo ao sistema a escrita posterior da informação na memória principal. Como resultado, o CPU fica livre mais rapidamente para executar outras operações. Em contrapartida, a latência do controlador pode induzir problemas de consistência de dados na memória principal, em sistemas multiprocessados com memória compartilhada. Esses problemas são tratados por protocolos de consistência do cache.

   c) Um dado acessado recentemente tem mais chances de ser usado novamente, do que um dado usado há mais tempo. Isso é verdade porque as variáveis de um programa tendem a ser acessadas várias vezes durante a execução de um programa, e as instruções usam bastante comandos de repetição e subprogramas, o que faz instruções serem acessadas repetidamente. Sendo assim, o

Sistema de Memória tende a manter os dados e instruções recentemente acessados no topo da Hierarquia de Memória.

d) Há uma probabilidade de acesso maior para dados e instruções em endereços próximos àqueles acessados recentemente. Isso também é verdade porque os programas são sequenciais e usam de repetições. Sendo assim, quando uma instrução é acessada, a instrução com maior probabilidade de ser executada em seguida, é a instrução logo a seguir dela. Para as variáveis o princípio é semelhante. Variáveis de um mesmo programa são armazenadas próximas uma às outras, e vetores e matrizes são armazenados em sequência de acordo com seus índices. Baseado neste princípio, o Sistema de Memória tende a manter dados e instruções próximos aos que estão sendo executados no topo da Hierarquia de Memória.