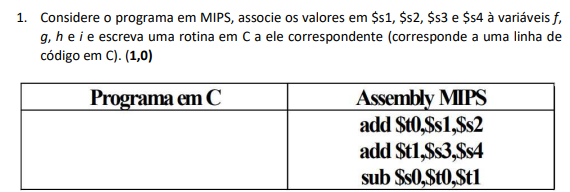
Nome: Vinícius Bretas Avezani de Mello Silva TADS AOC

1



add $t0, $s1, $s2

add $t1, $s3, $s4

sub $s0, $t0, $t1

s0 = (f + g) - (h + i);

gpt:

int f, g, h, i;

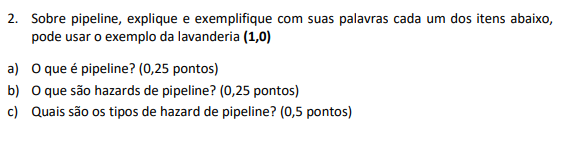
int t0, t1, s0;

t0 = f + g;

t1 = h + i;

s0 = t0 - t1;

2



a)Pipeline é uma técnica na arquitetura do processador para tirar maior proveito de seus recursos funcionais, separando em etapas ou áreas seus componentes para trabalhar de forma paralela.

b) Hazards de pipeline é tudo que prejudica a performance de suas instruções, fazendo que a pipeline tenha que fazer um stall ou fowarding até que tudo se alinhe novamente para processeguir com a sua instrução.

c) hazard estrutural, hazard de dados, hazard de controle

gpt:

a) Pipeline é uma técnica de processamento utilizada em arquitetura de computadores para melhorar o desempenho, dividindo o fluxo de execução de instruções em várias etapas independentes. Em vez de esperar que uma instrução seja totalmente executada antes de iniciar a próxima, o pipeline permite que várias instruções sejam processadas simultaneamente em diferentes estágios.

Usando a analogia da lavanderia, imagine que você tem várias roupas para lavar, secar e dobrar. Em vez de lavar uma peça de roupa por vez e esperar até que esteja completamente limpa antes de iniciar a próxima, você pode dividir o processo em etapas. Enquanto uma roupa está sendo lavada, você pode colocar outra na máquina de secar e uma terceira pode ser dobrada. Isso permite que você processe várias roupas ao mesmo tempo, aumentando a eficiência.

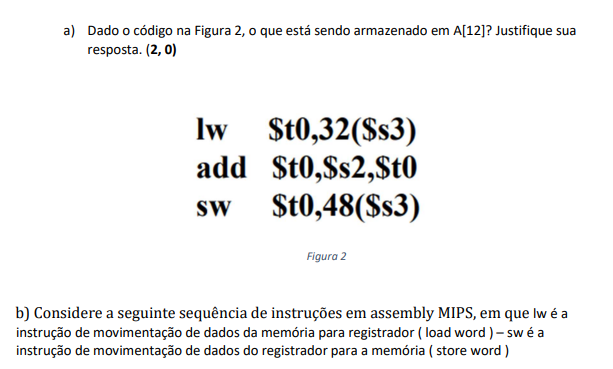
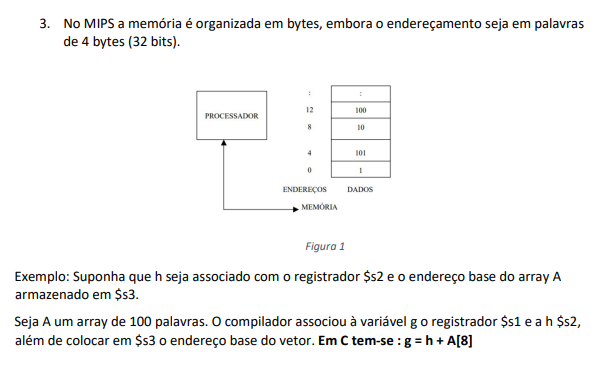
b) Hazards de pipeline são situações em que a correta execução das instruções em um pipeline é afetada por dependências entre as instruções. Essas dependências podem resultar em atrasos ou conflitos na busca, decodificação, execução ou escrita de dados, causando um impacto negativo no desempenho do pipeline.

Continuando com o exemplo da lavanderia, um hazard de pipeline pode ocorrer quando você precisa dobrar uma roupa antes que ela esteja completamente seca. Isso cria uma dependência, pois você não pode prosseguir para a etapa de dobrar até que a etapa de secagem esteja concluída. Esse atraso na execução das etapas pode diminuir a eficiência geral do processo.

c) Os tipos de hazard de pipeline são:

* Hazard de dependência de dados (Data Hazard): Ocorre quando uma instrução depende do resultado de uma instrução anterior que ainda não foi produzido. Isso pode resultar em conflitos de dados, onde a instrução que depende do resultado precisa esperar até que o resultado esteja disponível.
* Hazard de dependência de controle (Control Hazard): Ocorre quando a sequência de execução é alterada devido a uma instrução de desvio (branch) condicional. Isso pode resultar em instruções desnecessárias sendo buscadas ou instruções importantes sendo descartadas, causando um impacto no fluxo de instruções.
* Hazard de dependência de estrutura (Structural Hazard): Ocorre quando recursos físicos do pipeline não estão disponíveis para execução devido a limitações de hardware. Por exemplo, se um recurso for compartilhado entre múltiplas etapas do pipeline e não houver suporte para execução simultânea, pode ocorrer um atraso.
* Hazard de dados antecipados (Data Forwarding Hazard): Ocorre quando o resultado de uma instrução é encaminhado diretamente para uma instrução subsequente que depende desse resultado. Isso evita a necessidade de esperar que o resultado seja gravado em um registrador antes de ser utilizado, reduzindo os atrasos causados pela dependência de dados.
* Hazard de salto atrasado (Delayed Branch Hazard): Ocorre quando uma instrução de desvio (branch) condicional é tomada e as instruções subsequentes já foram buscadas e decodificadas. Isso pode resultar em instruções desnecessárias sendo executadas antes de desviar para a sequência correta, causando atrasos.

3

Encontre o hazard no código da Figura 3 abaixo e resolva-o? Dê seu nome e explique com suas palavras o que ele representa, desenhe a solução utilizando os símbolos de estágio da pipeline. (2,5 pontos)

1. T0 = A[8]

T0 = s2 + T0

A[12] = T0

Ou a[12] = s2+A[8]

Ou a[12] = s2 + 10

#$t1 tem o end. de v[k]

Lw $t0, 0($t1)

Lw $t2, 4($t1)

Sw $t2, 0($t1)

Sw $t0, 4($t1)

O conteúdo do s2 irá receber um stall por causa que precisa da instrução que vai preencher o $t2 ser terminado, quando o seu dado for preenchido e sua instrução finalizada a instrução sw irá poder continuar.

Gpt:

No código dado, há um hazard de dados conhecido como "hazard de dados antecipados" ou "data forwarding hazard". Esse hazard ocorre quando uma instrução depende de um resultado que ainda não foi produzido na etapa de escrita, mas que está sendo gerado em uma etapa posterior da pipeline.

No caso específico, o hazard ocorre nas instruções Sw $t2, 0($t1) e Sw $t0, 4($t1), onde os valores dos registradores $t2 e $t0 são atualizados antes de serem lidos pelas instruções anteriores (Lw $t0, 0($t1) e Lw $t2, 4($t1)). Essas instruções dependem dos valores atualizados para serem executadas corretamente.

A solução para esse hazard é utilizar a técnica de data forwarding, também conhecida como "encaminhamento de dados". O encaminhamento de dados permite que os resultados atualizados sejam enviados diretamente para as instruções que dependem deles, evitando a necessidade de esperar que sejam gravados em registradores antes de serem utilizados.

A seguir, está o diagrama de pipeline representando as etapas de execução das instruções, incluindo a solução do hazard utilizando símbolos de estágio:

markdownCopy code

\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_

| | | | | | | | | |

| IF | | ID | | EX | | MEM | | WB |

|\_\_\_\_\_\_\_\_| |\_\_\_\_\_\_\_\_| |\_\_\_\_\_\_\_\_| |\_\_\_\_\_\_\_\_| |\_\_\_\_\_\_\_\_|

| | | | |

| | | | |

| \_\_\_\_\_\_\_\_\_| | | |

| | | | |

| | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | | |

| | | | | | |

| | | \_\_\_\_\_\_\_\_\_ | | | |

| | | | | | | | |

| | | | \_\_\_\_\_ | | | | |

| | | | | || | | | |

| | | | | \_\_\_\_\_\_\_\_ | | | |

| | | | | | || | | |

| | | | | | || | | |

| | | | | | || | | |

| | | | | |\_\_\_\_\_\_\_\_|| | | |

| | | | |\_\_\_\_\_\_\_\_\_\_\_\_| | |

| | | |\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_| |

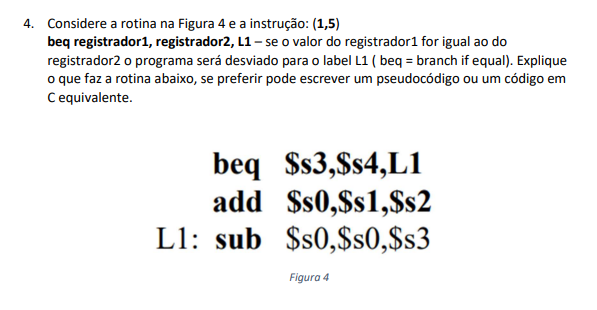
| |\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|

|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|

Na solução do hazard, o resultado atualizado dos registradores $t2 e $t0 é encaminhado diretamente para as instruções que dependem deles (Lw $t0, 0($t1) e Lw $t2, 4($t1)). Isso é representado pelos conectores entre as etapas da pipeline.

Dessa forma, as instruções Lw $t0, 0($t1) e Lw $t2, 4($t1) conseguem acessar os valores atualizados dos registradores $t2 e $t0, respectivamente, evitando o hazard e permitindo a execução correta do código.

4



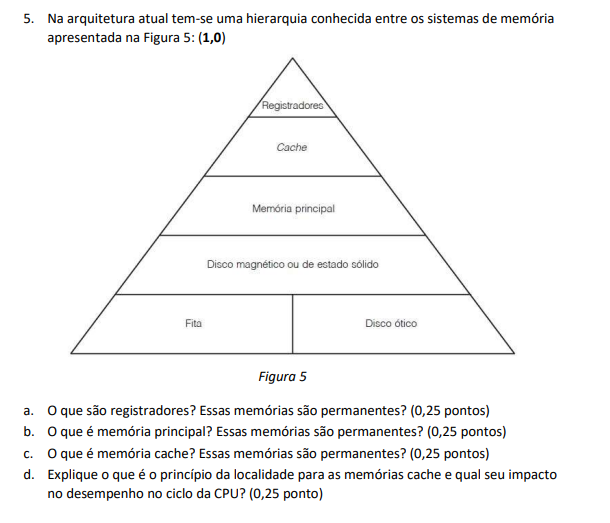
If (s3 == s4)

S0 = s0 + s3;

S0 = s1 + s2;

Eu gosto de pensar que as labels podem ser tratadas como funções, nesse caso o conteúdo dentro do if caso true seria a branch L1 no enúnciado

5



1. Registradores são dados armazenados na cpu para acesso rápido
   1. Sua memoria não é permanente, assim que a cpu for desligar esses dados são perdidos.
2. Memoria principal é o ram, onde os dados são armazenados
   1. Sua memoria não é permanente, assim que desligar esses dados são perdidos.
3. Memoria cache é uma memoria de acesso rápido mas não integrada na cpu como registradores, porém mais rápida que a memoria principal.
   1. Sua memoria não é permanente, assim que desligar esses dados são perdidos.
4. Cache se localiza o quanto perto do cpu possível para acesso rápido de seus dados

gpt:

a) Registradores são unidades de armazenamento de dados dentro de um processador. Eles são pequenas áreas de armazenamento localizadas dentro do processador e são usadas para armazenar dados temporários e resultados intermediários durante a execução de instruções. Os registradores são rápidos e têm acesso direto pelo processador, o que os torna essenciais para o desempenho eficiente do processamento de dados. No entanto, os registradores têm capacidade limitada em comparação com a memória principal.

Os registradores são voláteis, o que significa que seu conteúdo é perdido quando o computador é desligado. Eles não são permanentes, pois seu objetivo é fornecer armazenamento rápido e temporário para dados durante a execução de um programa.

b) Memória principal, também conhecida como RAM (Random Access Memory), é uma forma de armazenamento de dados em um computador. Ela é usada para armazenar instruções, dados e variáveis durante a execução de programas. A memória principal é acessada diretamente pela CPU e é composta por uma matriz de células de memória, cada uma com um endereço único.

A memória principal é um tipo de memória não volátil, o que significa que ela retém seus dados mesmo quando o computador é desligado. No entanto, seu conteúdo pode ser alterado durante a execução do programa.

c) Memória cache é uma memória de alta velocidade localizada entre a CPU e a memória principal. Ela tem o objetivo de armazenar temporariamente os dados e instruções mais frequentemente usados pelo processador, de modo a reduzir o tempo de acesso à memória principal.

A memória cache é construída com tecnologia mais rápida e cara do que a memória principal. Ela armazena cópias de partes da memória principal em níveis hierárquicos, como cache L1, cache L2 e cache L3. Quanto mais próximo o nível de cache está da CPU, mais rápido é o acesso aos dados.

Assim como os registradores, a memória cache também é volátil. Ela não é permanente e seu conteúdo pode ser alterado durante a execução do programa. No entanto, a memória cache é gerenciada pelo hardware do computador de forma transparente para o usuário.

d) O princípio da localidade é uma propriedade do acesso à memória em que é observado que, durante a execução de um programa, há uma tendência de que as instruções e os dados referenciados sejam agrupados em áreas próximas da memória.

Existem dois tipos principais de localidade:

* Localidade espacial: Refere-se ao fato de que, quando uma instrução ou dado é acessado, é provável que as instruções ou dados subsequentes sejam acessados em locais próximos. Por exemplo, durante a execução de um loop, as instruções dentro do loop tendem a ser acessadas repetidamente em uma sequência próxima.
* Localidade temporal: Refere-se ao fato de que, quando uma instrução ou dado é acessado, é provável que ele seja acessado novamente em um futuro próximo. Por exemplo, variáveis temporárias que são usadas repetidamente em um trecho de código tendem a ser acessadas novamente em iterações subsequentes do loop.

O princípio da localidade é fundamental para o desempenho da memória cache. A memória cache explora essa propriedade armazenando as instruções e dados mais frequentemente acessados próximos à CPU, reduzindo assim o tempo de acesso à memória principal. Ao reduzir a latência de acesso à memória, o princípio da localidade melhora o desempenho no ciclo da CPU, uma vez que a CPU pode buscar dados e instruções com mais rapidez, evitando a espera pela recuperação de dados da memória principal.