1. Uma instrução é quebrada em etapas e cada etapa é executada em um ciclo de clock. A quantidade de clocks podem variar dependendo da instrução. Os componentes podem ser reusados em uma instrução em ciclos de clock diferentes, permitindo a redução de componentes parecidos.
2. Separar o caminho de dados em estágios (o exemplo mais comum é em cinco estágios). É necessário salvar os dados entre cada estágio e para isso é necessário registradores.
4. Conflito RAW: dependência do sub.d com o resultado de F1 produzido pelo primeiro div.d;

Conflito RAW: dependência do s.d com o resultado de F4 produzido pelo primeiro sub.d;

Conflito RAW: dependência do segundo div.d com o resultado de F5 produzido pelo add.d;

Conflito WAW: sub.d e segundo div.d com o registrador F4;

Conflito WAR: antidependência se sub.d com o add.d com o registrador F5.

1. CPU tempo ideal = (CPU ciclos de clock + Ciclos de Clock de Stall de Memória) x Período de ciclo de Clock =>

CPU tempo ideal = (IC x CPI + 0) x Período de ciclo de Clock =>

CPU tempo ideal = IC x 1 x Período de ciclo de Clock

Ciclos de clock de stall de memória = IC x (1 + 0,5) x 0.02 x 25 = IC x 0,75

CPU tempo real = (IC x 1 + IC x 0,75) x Período de ciclo de Clock =>

CPU tempo real = IC x 1,75 x Período de ciclo de Clock

Comparando os desempenhos com stalls pelo ideal:

a) É relativo à quando a escrita de dados é feita simultaneamente na memória Cache e no endereço correspondente na memória principal, fazendo com que sempre que houver uma alteração do valor a memória principal será atualizada, contudo essa técnica traz consigo um maior tempo, pois toda vez ter que atualizar o valor na memória principal tem um gasto de processamento.

b) É uma alternativa ao write-through, a técnica faz a escrita diretamente no Cache e só atualiza os dados na memória principal quando o bloco na Cache é substituído, dessa forma se usa menos tempo e processamento, pois não tem que ficar atualizando a memória principal a cada alteração, mas uma implementação write-back é mais complexa.

c) Se trata do reconhecimento que um dado que foi acessado recentemente tem mais chance de ser usado novamente do que um dado que já foi usado há muito tempo, instruções de loop aproveitam esse princípio.

d) Se trata do reconhecimento há uma maior chance de se acessar um dado de posições próximas a um dado já acessado, por exemplo programas sequenciais aproveitam esse princípio, com isso as instruções próximas as que estão sendo usadas ficam armazenadas na memória cache.