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.
3. Sem pipeline

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Ciclo | 1 | 2 | 3 | 4 | 5 | 6 |
| subi $t2, $t2, 4 | IF | ID | EX | WB |  |  |
| lw $t1, 0($t2) |  |  |  |  | IF | ID |
|  | | | | | | |
| Ciclo | 7 | 8 | 9 | 10 | 11 | 12 |
| lw $t1, 0($t2) | EX | MEM | WB |  |  |  |
| add $t3, $t1, $t4 |  |  |  | IF | ID | EX |
|  | | | | | | |
| Ciclo | 13 | 14 | 15 | 16 | 17 | 18 |
| add $t3, $t1, $t4 | WB |  |  |  |  |  |
| add $t4, $t3, $t3 |  | IF | ID | EX | WB |  |
| sw $t4, 0($t2) |  |  |  |  |  | IF |
|  | | | | | | |
| Ciclo | 19 | 20 | 21 | 22 | 23 | 24 |
| sw $t4, 0($t2) | ID | EX | MEM |  |  |  |
| beq $t2, $0, loop |  |  |  | IF | ID | EX |



|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Ciclo | 1 | 2 | 3 | 4 | 5 | 6 |
| div.d F1, F2, F3 | IF | ID | EX | WB |  |  |
|  | bolha | bolha | bolha | bolha | bolha | bolha |
| sub.d F4, F5, F1 |  |  |  | IF | ID | EX |
|  | | | | | | |
| Ciclo | 7 | 8 | 9 | 10 | 11 | 12 |
| sub.d F4, F5, F1 | WB |  |  |  |  |  |
|  | bolha | bolha | bolha | bolha | bolha | bolha |
| s.d F4, 4(F10) | IF | ID | EX | MEM |  |  |
| add.d F5, F6, F7 |  | IF | ID | EX | WB |  |
|  | bolha | bolha | bolha | bolha | bolha | bolha |
| div.d F4, F5, F6 |  |  |  |  | IF | ID |
|  | | | | | | |
| Ciclo | 13 | 14 | 15 | 16 | 17 | 18 |
| div.d F4, F5, F6 | EX | WB |  |  |  |  |
|  |  |  |  |  |  |  |
| div.d F1, F2, F3 | IF | ID | EX | WB |  |  |
| sub.d F4, F5, F1 |  | IF | ID | EX | WB |  |
| s.d F4, 4(F10) |  |  | IF | ID | EX | MEM |
| add.d F5, F6, F7 |  |  |  | IF | ID | EX |
| div.d F4, F5, F6 |  |  |  |  | IF | ID |
| add.d F5, F6, F7 | WB |  |  |  |  |  |
| div.d F4, F5, F6 | EX | WB |  |  |  |  |
|  |  |  |  |  |  |  |

Conflito RAW: primeiro div.d e sub.d com o registrador F1

Conflito RAW: sub.d e s.d com o registrador F4

Conflito RAW: add.d e segundo div.d com o registrador F5

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

Conflito WAR: sub.d e 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, quanto no endereço correspondente na memória principal, fazendo que dessa forma não se trabalhe com dados desatualizados, pois 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.

b) É a técnica que o processador faz a escrita diretamente no Cache e fica como responsabilidade do sistema fazer a atualização dos dados na memória principal, dessa forma se usa menos tempo, pois não tem que ficar atualizando a memória principal a cada alteração, mas pode acontecer que o processador acabe operando com dados desatualizados, por que o sistema não mandou a instrução para a atualização dos dados na memória principal.

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, dessa forma os dados recentes que estão na RAM são copiados para o Cache.

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 tendem a acessar endereços de memória próximos e com isso quando eu acesso uma instrução a instrução com mais chance de ser acessada será a próxima, com isso as instruções próximas as que estão sendo usadas ficam armazenadas na memória cache.