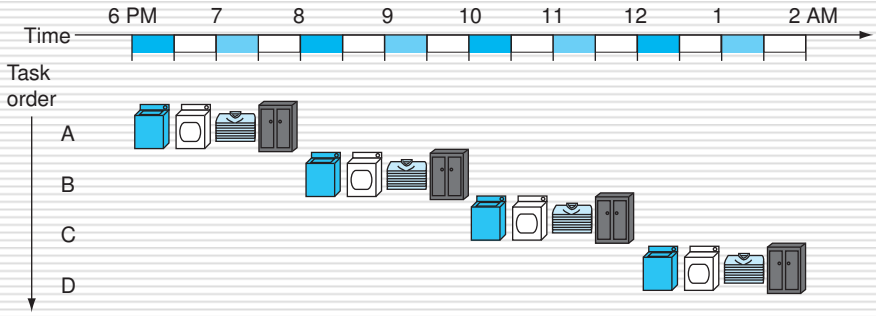
ARQUITETURA DE COMPUTADORES

Pipeline

Pipelining – o conceito (1)

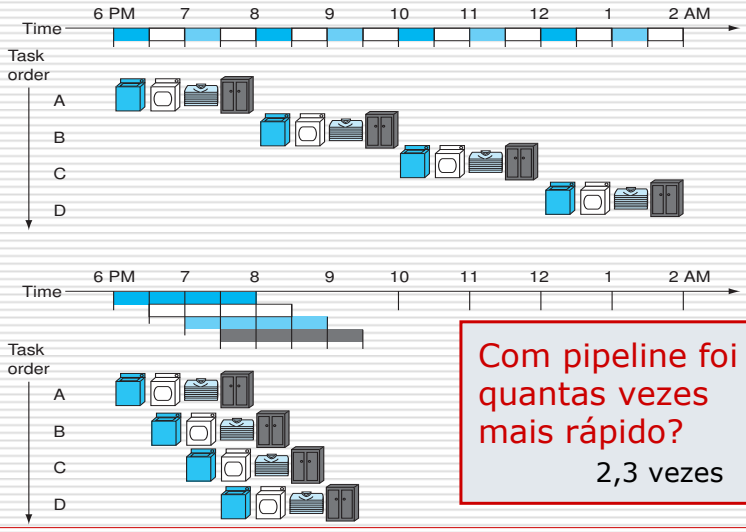
- Lavagem de roupa suja:
  - Colocar a roupa suja na lavadora
  - Quando a lavadora terminar, colocar a roupa molhada na secadora
  - Quando a secadora terminar, colocar a roupa seca na mesa de passar
  - Quando terminar de passar, pedir ao colega de quarto para guardar a roupa

Pipelining – o conceito (2)



3

Pipelining – o conceito (3)



Com pipeline foi quantas vezes mais rápido?

2,3 vezes

4

Pipelining – análise do exemplo (1)

- Às 7:30 todos os recursos operam simultaneamente
- Existe um tempo inicial até que o pipeline "encha"
- O ciclo é constante (e limitado pelo estágio mais lento)
- Paradoxo – tempo para tratar uma trouxa (latência) [cte] *versus* vazão total [↑]

5

Pipelining – análise do exemplo (2)

Suponha que sejam lavadas 20 trouxas de roupa.
Quantas vezes mais rápido seria usar o pipeline?

"Trouxas" são Instruções

$$\text{Ganho por usar pipeline} = \frac{\text{Nº de estágios} * \text{Nº de trouxas}}{\text{Nº de estágios} - 1 + \text{Nº de trouxas}}$$

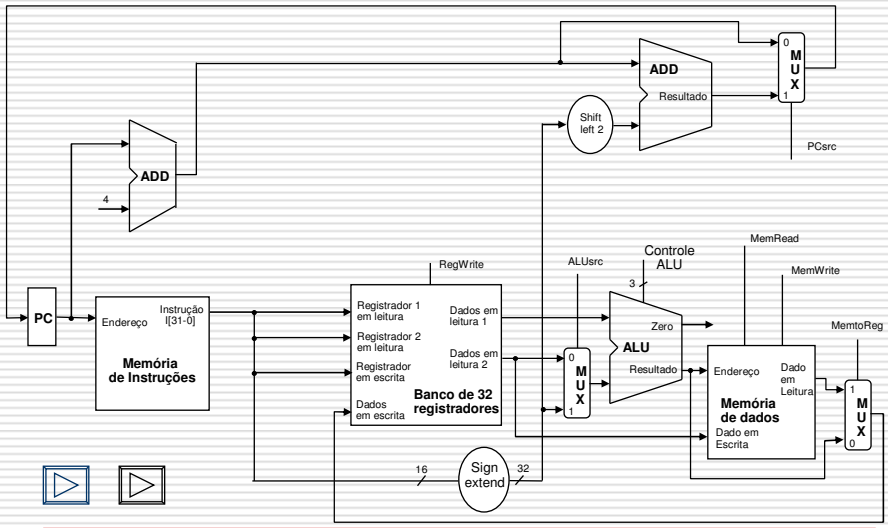
Tempo usando pipeline

= 3,5 vezes

Qual seria o ganho máximo? 4 vezes (= Nº de estágios)

6

Processador MIPS monociclo



Pipelining MIPS

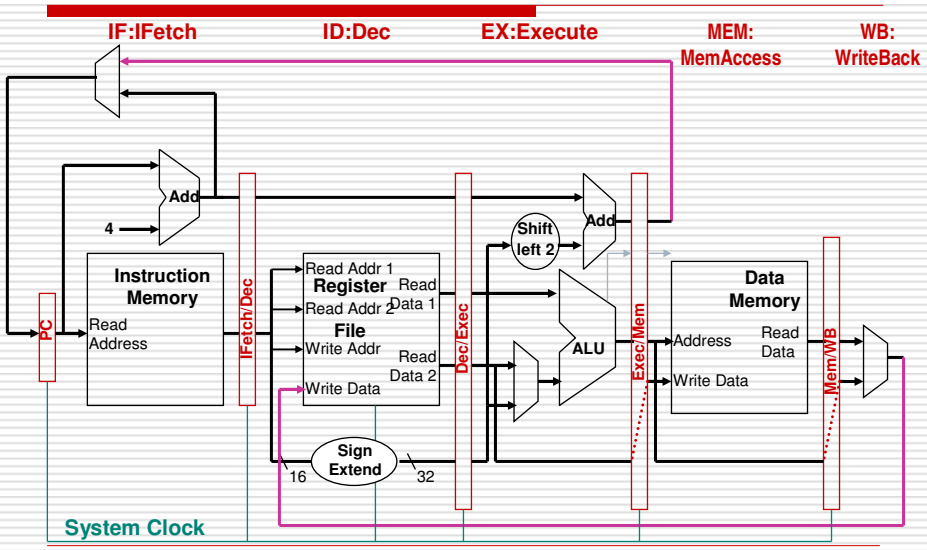
- Buscar instrução na memória
- Ler registradores / decodificar instrução
- Executar a operação ou calcular o endereço
- Acessar operando na memória de dados
- Escrever resultado num registrador

CINCO ESTÁGIOS

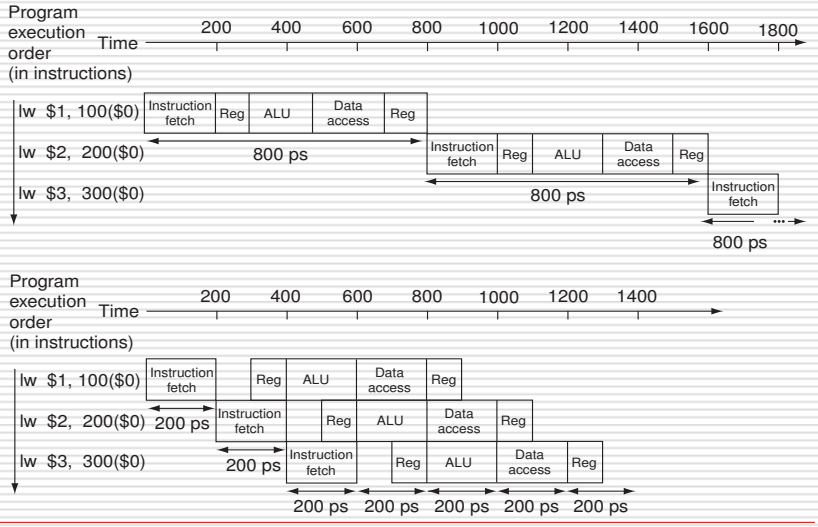
MIPS com pipeline

- O que precisa ser adicionado/modificado no caminho de dados do MIPS?
- Registradores entre cada estágio pipeline para isolá-los.

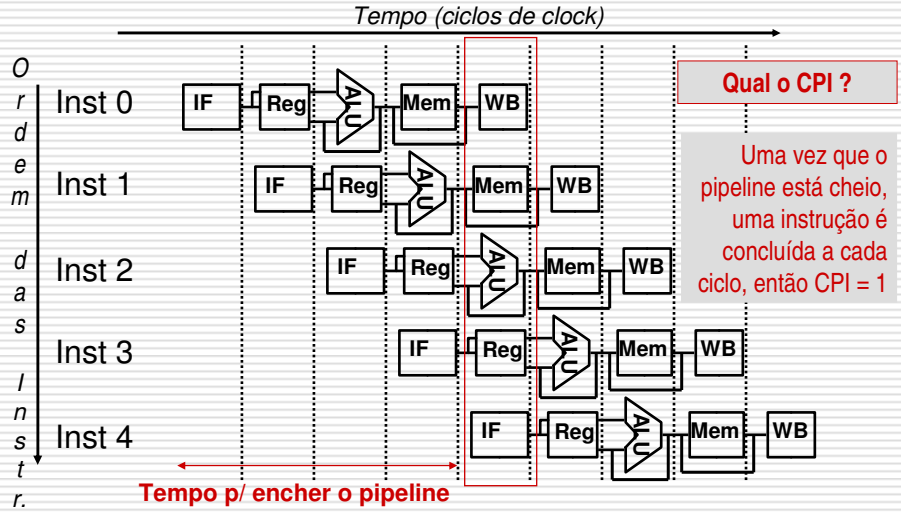
MIPS com pipeline



Execução monociclo *versus* execução pipeline



Processador MIPS com pipeline (1)

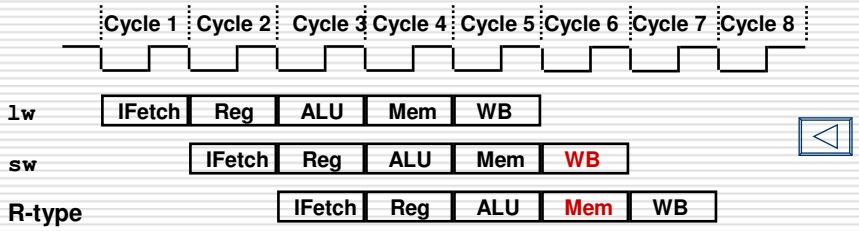


Qual o CPI ?

Uma vez que o pipeline está cheio, uma instrução é concluída a cada ciclo, então CPI = 1

Processador MIPS com pipeline (2)

Só implementa instruções lw, sw, R (add, and) e branch (beq)



- Ciclo de clock (tempo dos estágios pipeline) é limitado pelo estágio mais lento
- Para algumas instruções, alguns estágios são desperdiçados

13

O futuro é construído pelas nossas decisões diárias, inconstantes e mutáveis, e cada evento influencia todos os outros.

Alvin Toffler

14

Problemas no desempenho

Problemas para um projeto visando bom desempenho

- como dividir todas as instruções num mesmo conjunto de estágios?
- como obter estágios com tempos de execução similares?

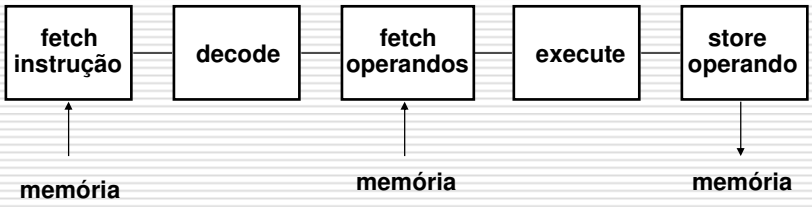
Problemas em tempo de execução

- conflitos de memória (Hazard estrutural)
  - acessos simultâneos à memória por 2 ou mais estágios
- dependências de dados (Hazard de dados)
  - instruções dependem de resultados de instruções anteriores, ainda não completadas
- instruções de desvio (Hazard de controle)
  - instrução seguinte não está no endereço seguinte ao da instrução anterior

15

Conflitos de memória

- Problema: acessos simultâneos à memória por 2 ou mais estágios

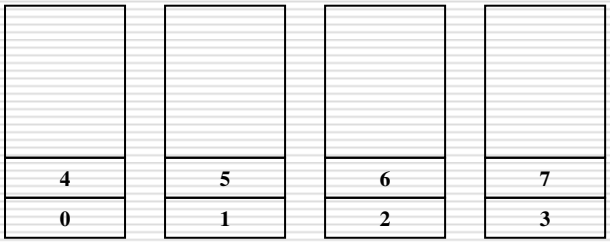


Soluções:

- Caches (memórias) separadas de dados e instruções
- Memória entrelaçada

16

Memória entrelaçada



- Bancos de memória separados, cada um com seu registrador de dados
- Acessos simultâneos são possíveis a bancos diferentes
- Barramentos entre memória e processador devem operar a velocidade mais alta

17

Processador MIPS (facilidades para o pipeline)

- Memória é acessada apenas por instruções LOAD e STORE
- Apenas um estágio do pipeline faz acesso a operandos de memória
  - apenas 1 instrução pode estar executando acesso a dados a cada instante
- Se memória de dados e instruções são separadas, não há nenhum conflito de acesso à memória

18

Dependência de dados

- *Problema*: instruções consecutivas podem fazer acesso aos mesmos operandos
  - execução da instrução seguinte pode depender de operando calculado pela instrução anterior
- Caso particular: instrução precisa de resultado anterior (p.ex. registrador) para cálculo de endereço efetivo de operando
- Tipos de dependências de dados
  - dependência verdadeira
  - Antidependência
  - dependência de saída

19

Dependência verdadeira (1)

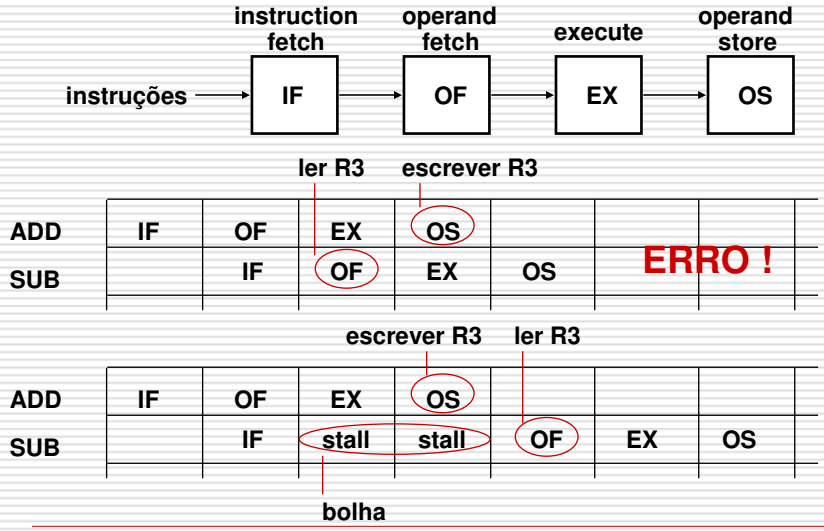
- Exemplo

```
1. ADD R3, R2, R1        ; R3 = R2 + R1
2. SUB R4, R3, 1         ; R4 = R3 – 1
```

- Instrução 2 depende de valor de R3 calculado pela instrução 1
- Instrução 1 precisa atualizar valor de R3 antes que instrução 2 busque os seus operandos
- *read-after-write hazard*
- Pipeline precisa ser parado durante certo número de ciclos

20

Dependência verdadeira (2)



21

Dependência falsa (1)

Antidependência

- Exemplo

  1. ADD R3, R2, R1 ; R3 = R2 + R1
  2. SUB R2, R4, 1 ; R2 = R4 – 1

- Instrução 1 utiliza operando em R2 que é escrito pela instrução 2
- Instrução 2 não pode salvar resultado em R2 antes que instrução 1 tenha lido seus operandos
- *write-after-read hazard*
- Não é um problema em pipelines onde a ordem de execução das instruções é mantida
  - escrita do resultado é sempre o último estágio
- Problema em processadores super-escalares

22

Dependência falsa (2)

Dependência de saída

- exemplo

```
1. ADD R3, R4, R1      ; R3 = R4 + R1
2. SUB R2, R4, 1       ; R2 = R4 – 1
3. ADD R3, R1, R5      ; R3 = R1 + R5
```

- instruções 1 e 3 escrevem no mesmo operando em R3
- instrução 1 tem que escrever seu resultado em R3 antes do que a instrução 3, senão valor final de R3 ficará errado
- *write-after-write hazard*
- também só é problema em processadores super-escalares

23

Forwarding (1)

- Exemplo

```
ADD R3, R2, R0
SUB R4, R3, 8
```

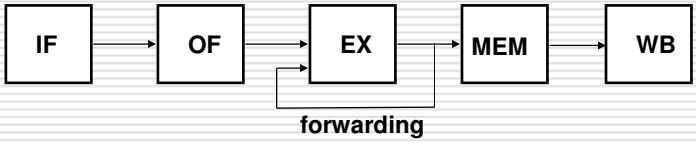
- Instrução SUB precisa do valor de R3, calculado pela Instrução ADD
  - Valor é escrito em R3 por ADD no último estágio - WB (write-back)
  - Valor é necessário em SUB no terceiro estágio
  - Instrução SUB ficará presa por 2 ciclos no 2º estágio do pipeline

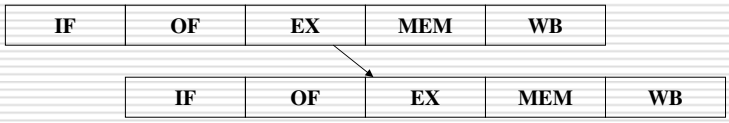
supõe-se escrita no banco de registradores na primeira metade do ciclo e leitura na segunda metade

24

Forwarding (2)

- caminho interno dentro do pipeline entre a saída da ALU e a entrada da ALU
  - evita *stall* do pipeline

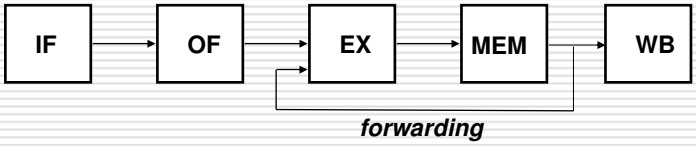


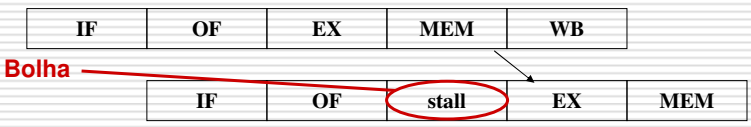


25

Forwarding (3)

- exemplo 2
  LOAD R3, 100 (R0)
  ADD R1, R2, R3
- *Forwarding*: caminho interno dentro do pipeline entre a saída da memória de dados e a entrada da ALU





26

Exemplo (atuação do compilador)

Considere o seguinte código em linguagem C:
A = B + E;
C = B + F;

... que em mnemônicos MIPS seria: (supondo que todas as variáveis sejam endereçáveis como offset a partir de $t0)

```
lw    $t1, 0($t0)
lw    $t2, 4($t0)
add   $t3, $t1, $t2
sw    $t3, 12($t0)
lw    $t4, 8($t0)
add   $t5, $t1, $t4
sw    $t5, 16($t0)
```

Encontre os *hazards* e reordene as instruções para evitar pipeline stalls.

27

Exemplo

```
lw    $t1, 0($t0)
lw    $t2, 4($t0)
add   $t3, $t1, $t2
sw    $t3, 12($t0)
lw    $t4, 8($t0)
add   $t5, $t1, $t4
sw    $t5, 16($t0)
```

Dependência de dados

Dependência de dados

```
lw    $t1, 0($t0)
lw    $t2, 4($t0)
lw    $t4, 8($t0)
add   $t3, $t1, $t2
sw    $t3, 12($t0)
add   $t5, $t1, $t4
sw    $t5, 16($t0)
```

28

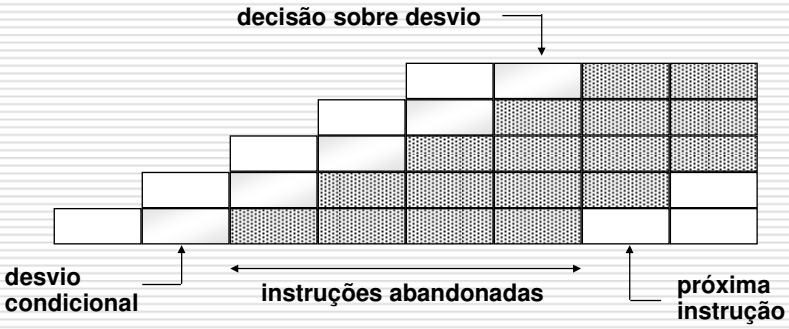
O homem que não lê não tem mais mérito que o homem que não sabe ler.

Mark Twain

29

Dependências em desvios (1)

- Efeito de desvios condicionais
  - se o desvio ocorre, o pipeline precisa ser esvaziado
  - não se sabe se o desvio ocorrerá ou não até o momento de sua execução



30

Dependências em desvios (2)

- Instruções abandonadas não podem ter afetado conteúdo de registradores e memórias
  - Isto é usualmente automático, porque escrita de valores é sempre feita no último estágio do pipeline
- Deve-se procurar antecipar a decisão sobre o desvio para o estágio mais cedo possível
- Desvios incondicionais:
  - Sabe-se que é um desvio desde a decodificação da instrução (segundo estágio do pipeline)
  - É possível evitar abandono de número maior de instruções
  - Problema: em que estágio é feito o cálculo do endereço efetivo do desvio?

31

Técnicas de tratamento de desvios condicionais

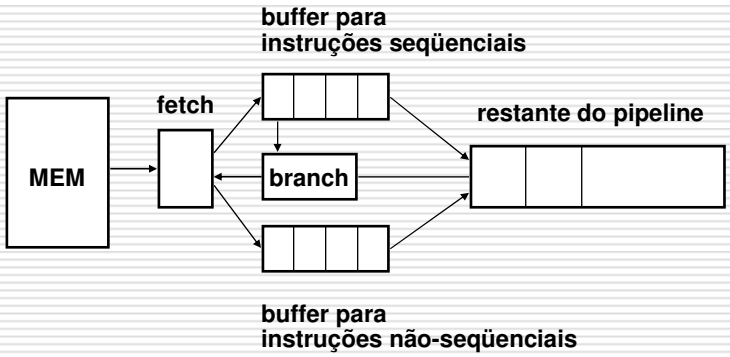
Prevenir efeitos dos desvios
- buffers paralelos de instruções

Prever direção do desvio
- predição estática
- predição dinâmica

Eliminar problema pela redefinição do desvio condicional
- "delayed branch"

32

Buffers paralelos de instruções



33

Predição estática

- Supor sempre mesma direção para o desvio
  - desvio sempre ocorre
  - desvio nunca ocorre
- Compilador define direção mais provável
  - instrução de desvio contém bit de predição, ligado / desligado pelo compilador
  - início de laço (ou desvio para frente): desvio improvável
  - final de laço (ou desvio para trás): desvio provável
- Até 85 % de acerto é possível

34

Predição dinâmica (1)

- Tabela look-up associativa armazena triplas
  - endereços das instruções de desvio condicional mais recentemente executadas
  - endereços de destino destes desvios
  - bit de validade, indicando se desvio foi tomado na última execução
- Quando instrução de desvio condicional é buscada na memória
  - é feita comparação associativa na tabela, à procura do endereço desta instrução
  - se endereço é encontrado e bit de validade está ligado, o endereço de desvio armazenado na tabela é usado
  - ao final da execução da instrução, endereço efetivo de destino do desvio e bit de validade são atualizados na tabela
- Tabela pode utilizar diversos mapeamentos e algoritmos de substituição

35

Predição dinâmica (2)

- Variação: *branch history table*
  - Contador associado a cada posição da tabela
  - A cada vez que uma instrução de desvio contida na tabela é executada ...
    - Contador é incrementado se desvio ocorre
    - Contador é decrementado se desvio não ocorre
  - Valor do contador é utilizado para a predição

36

Delayed branch

- Desvio não ocorre imediatamente, e sim apenas após uma ou mais instruções seguintes
- Caso mais simples: pipeline com 2 estágios – fetch + execute
  - Desvio é feito depois da instrução seguinte
  - Instrução seguinte não pode ser necessária para decisão sobre ocorrência do desvio
  - Compilador reorganiza código
  - Tipicamente, em 70% dos casos encontra-se instrução para colocar após o desvio
- Pipeline com N estágios
  - Desvio é feito depois de N – 1 instruções

37

Exemplo

Address	Normal	Delayed	Optimized
100	Load X,A	Load X,A	Load X,A
101	Add A,1	Add A,1	Jump 105
102	Jump 105	Jump 106	Add A,1
103	Add A,B	Nop	Add A,B
104	Sub C,B	Add A,B	Sub C,B
105	Store A,Z	Sub C,B	Store A,Z
106		Store A,Z	

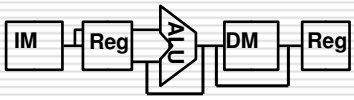
38

Outros exemplos de pipelines

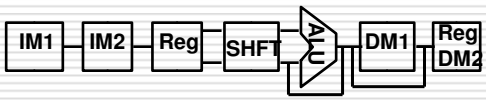
- ARM7

PC update IM access | decode reg access | ALU op; DM access; shift/rotate; commit result | (write back)

- StrongARM-1



- XScale



39

Literatura

- *Patterson & Hennessy – cap 6*
- *Stallings – cap 11.5*
- *Sites de fabricantes*

40

Resumo

- Todos processadores modernos usam pipeline
- Pipeline não diminui a latência, mas aumenta a vazão
- Ganho potencial: CPI = 1
- Clock limitado pelo estágio mais lento
  - Pipeline desbalanceado leva a ineficiência
  - Profundidade do pipeline pode prejudicar códigos pequenos
- É preciso detectar e resolver os hazards
  - Bolhas afetam negativamente o CPI

Há tempo para todo propósito embaixo do céu e na terra.

Eclesiastes 3:1

Questões adicionais...

- O que é busca antecipada (*prefetching*)? Qual a relação com pipeline?
- Investigue o conceito de "*loop buffer*". Relacione com desempenho do processador e com pipeline.