

Arq
1

Gisele Craveiro

04/03/13

Bibliografia: • Patterson e Hennessy - Organização e Projeto
de Comp.: A interface
Hardware/Software 3ª Edição

• Stallings 8ª Ed.

• Arq. de comp. \rightarrow ^{Aprox.} Quantitativa

P1: 29/04

P2: 17/06

}

$$MF = 0,3 * MT + 0,7 * MP$$



$$\frac{MPI + MP2}{2}$$


$$MT \text{ e } MP \gg S$$


Atendimento \rightarrow 22 das 14h às 17h

I1 \rightarrow 252

giselesc@usp.br

Capítulos 4 e 6 \rightarrow Patterson

 \rightarrow sinal de clock sincroniza execução do computador

 \rightarrow frequência menor. Menos ciclos por unidade de tempo.

1s \rightarrow • cl/ instrução demora um núm. de ciclos, não tempo.
A de uma executando mais ciclos por segundo executa
mais rapidamente.

n instruções $\Rightarrow n \cdot x \Rightarrow$ ciclos $\Rightarrow n \cdot x \cdot y$ segundos

Desempenho \rightarrow apenas tempo é considerado.

\hookrightarrow depende de muitas outras coisas além do sinal do clock.

\hookrightarrow sinal do clock (frequência)

\hookrightarrow no de instruções por ciclo

tem

mesma arquitetura:

A: $\left\{ \begin{array}{l} \text{tempo de ciclo de clock: } 250 \text{ ps} \\ \text{ciclos por instrução: } 2,0 \end{array} \right.$

\rightarrow cada instrução aprox. 2 ciclos

B: $\left\{ \begin{array}{l} 500 \text{ ps} \\ \text{CPI: } 1,2 \text{ ciclos} \end{array} \right.$

A	B	C
250	250	250

500 ps por instrução

no de instruções é idêntico neste caso

A	B	C
1,2	500	

600 ps por instr.

$S \cdot X = 600$

$X = \frac{600}{1,2} = 500$

20% mais rápido

Para fazer algo mais rápido, deve-se otimizar o caso mais comum, em que o impacto será maior.

Lei de Amdahl

↳ ferramenta para quantificar o ganho.

↳ quanto uma otimização pode melhorar o desempenho e se vale a pena investir, além da comparação de 2 alternativas de otimização

$$\text{Speedup}_{\text{enhanced}} = \frac{T_{\text{exec sem melhoria}}}{T_{\text{exec com melhoria}}}$$

Fraction enhanced (A ser melhorada)

Ex: exec. Total = 60s

parte que recebe melhoria = 20s

Fraction enhanced $\rightarrow 20/60$

melhora e ganho da melhoria $\rightarrow \text{Speedup enhanced} \rightarrow$

$$\text{Speedup}_{\text{overall}} = \frac{1}{(1 - \text{fraction}) + \frac{\text{fraction}}{\text{speedup}}}$$

parte que não muda SEM MELHORIA

FP \rightarrow cálculo de Ponto Flutuante

↳ Speedup de 10 em 20% de tempo.
↳ speedup de 2 em 50% de tempo.

$$f_r = 20\%$$

$$f_r = 50\%$$

$$\frac{1}{0,80 + 0,20/10} = 1,25$$

$$\frac{1}{0,5 + 0,5/2} = 1,33$$

Vale mais a pena 2 operações exatas do que 10 operações de ponto flutuante

$$\frac{1}{0,82} = 1,22$$

Arg
4

Possíveis melhorias

→ aumento na vel. do clock

→ single core, multicore,
pipeline, qto registradores

04/03/13

→ melhoria na org. de processador que diminuem a CPI

→ melhoria no compilador, que diminuem a CPI e/ou contagem de instr.

→ escolhas de algoritmo/linguagem que afetam a cont. de instr.

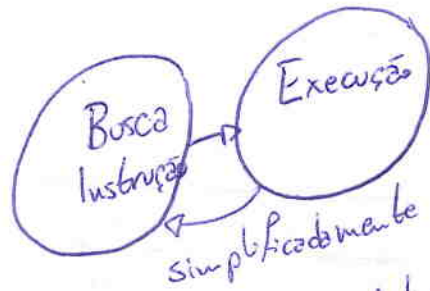
Prova

Arg
1

Como otimizar a Máquina?

11/03/13

- cap 6 a 4 - Patterson
- cap 12.4 - Stallings (8a ed)



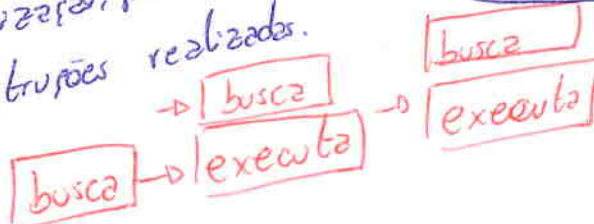
→ Prefetch

enquanto processador está executando uma instrução paralelamente pode ser feita a busca da execução seguinte

Antigamente → CPU dedicada → ficava ociosa na E/S → foi feita a multi-programação → enqt. processador está esperando E/S ele coloca outro processo p/ ser executado.

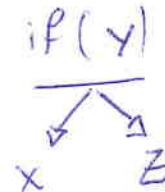
↓
debra n° de execuções pelo tempo de diminuição pela metade do tempo, em condições ideais.

↓
grande otimização, pois afeta todas as instruções realizadas.



→ mais demorada que limita o tempo, pois a busca é mto mais simples que a execução. Então ela fica ociosa

Desvio condicional → não se sabe ainda se a instrução a ser executada é a X ou a Z

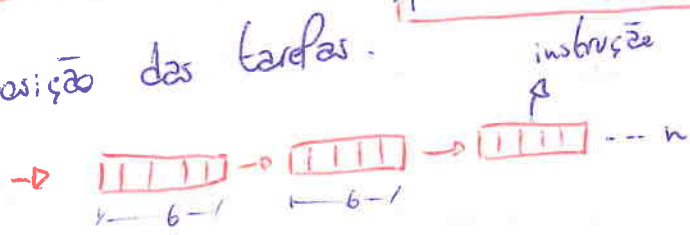


Pipelining → não é paralelismo (1 inst. por unid. de tempo)

paralelismo de estágios

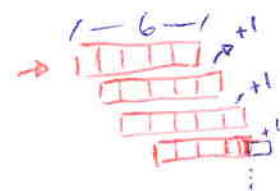
sobre posição das tarefas.

antes



tempo total = $6n$

cl pipeline



ao fim da primeira instrução o pipeline está cheio e todas estão trabalhando. Assim que a 1ª acaba só a 2ª

tempo total = 6 (da primeira) + $1(n-1)$

no cenário ideal

Penalidade (limpar pipeline) → devido a desvio condicional por exemplo.

→ Processador diferente com capacidade de cada porção ser responsável por uma parte da instrução. Busca, decodificação, etc.

#Custos:

overhead da movimentação de dados e controle mais complexo da ULA.

armazenamento temporário

dependência de dados

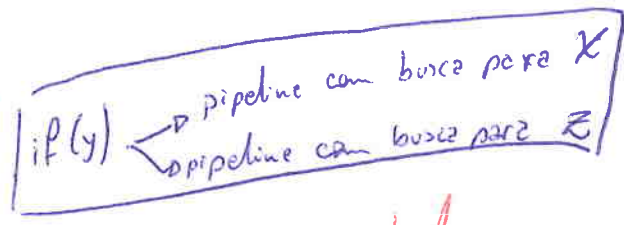
variável não pode ser acessada se ainda não tiver recebido o valor fornecido na instrução anterior.

$Speedup_k = \frac{n \cdot k}{k + (n-1)}$

↳ fator de aceleração → qto mais estágios melhor, mas nto tb não é bom devido aos custos

Para se resolver os desvios

no Múltiplos fluxos → dois pipelines



↳ projeto de hardware é mais complicado.

- problema de ifs entalhados
- problema de uso de barram e registradores.

Arg
3 ↳ busca antecipada da instrução alvo. Mantém no registrador até a condição ser vista
ex: bgt ↳ branch greater than → se condição true, branch. Caso contrário PC+1

↳ memória de laço de repetição → ~~uma~~ pipeline chuta que sempre voltará no loop. Ele só errará uma vez.
↳ rápida e dedicada para instruções de loop.

↳ previsão de desvio
↳ sempre chuta p/ direita ou esq.
↳ baseada em OPCODE (até 75% de sucesso) e estatísticas

Mesma Arquitetura \rightarrow MIPS, por exemplo

\hookrightarrow pode haver diferença na implementação, por isso o CPI pode ~~haver~~ ser diferente em cada máquina

ex: Multiplicação \rightarrow diversas adds
 \hookrightarrow circuito implementado com multiplicação direta

Não existe paralelismo de instruções, mas sim de estágios. Apenas 1 instrução terminada por vez.

Técnicas Dinâmicas para Desvio de Pipeline

\hookrightarrow Baseado em Histórico \rightarrow bit único (guarda apenas o último)
 \rightarrow tabela de históricos
armazenado em cache

\rightarrow Atraso de Desvio \rightarrow Fazer a busca ^{da instrução} na hora que obtiver o resultado do desvio.

RISC \rightarrow Comp. com conj. de instr. reduzido $\left\{ \begin{array}{l} \text{microprograma} \rightarrow \text{sequência lógica} \\ \text{das instruções de uma dada arq.} \\ \text{com as microoperações.} \end{array} \right.$

\hookrightarrow grande n.º de registrad. (prop. geral)

\hookrightarrow uso de compiladores p/ otimizar uso dos registr.

\hookrightarrow ênfase na otimização da pipeline.

\hookrightarrow tamanho fixo das instruções.

\hookrightarrow menos modos de endereçamento.

\hookrightarrow unidade de controle. tem memória de controle para armazenar o microprograma. No RISC a lógica da instr. estava implementada no

CISC → custo do software ^{excedam} custo do hardware

↳ Gap semântico → grandes conj. de instruções

↳ Programação mais simples, porém o Hardware era ^{muito} mais complexo.

↳ facilitou p/ programador e compilador.

Características da execução

↳ pesquisar e analisar dados da execução.

↳ operandos utilizados, operações realizadas, modos de end.

→ Têm feitas muitas movimentações de dados, condicionais, desvios

penetrar mais caso
comum, para otimizar-lo.

Cap. 5

Implementação MIPS

↳ Arquitetura do conjunto de instruções ~~afeta~~ e diversos aspectos de implementação afetam a **frequência do clock** e a qtd de **CPI**

→ Existem 2 passos básicos para todas as instruções:

1. Buscar a instrução apontada pelo PC na mem.
2. Ler/Buscar os operandos, que podem estar nos registr. ou mem.

→ A maioria das instruções é executada de forma similar.

→ **Multiplexador**: seleciona para qual das linhas de dados o dado que vem vai passar.

↓
São 3. Quais são?

ARQ
1RISC x CISC

→ tinha que dar conta de qq situação ^{pr inst. de 2 bytes a 32 bytes, por ex.}
→ hardware mais complexo p/ facilitar programação.

~~08/04/13~~

08/04/13

- instruções mais simples
- menos instruções
- tamanho fixo de instruções

{ Simplificar o hardware

↳ afim de melhorar desempenho

Qual é o caso comum?

Como?

↳ verificando a ocorrência, através de testes com programas normalmente utilizados pelos usuários em alto nível

↳ verificaram a incidência disso no nível de instruções de máquina e ref. de mem.

Conclusões Sobre o Caso Common (CISC)

1. Operações de atribuição são mto freq. (movimentação de dados)
2. Operandos
 - ↳ escalares (int, float, ...)
 - ↳ variáveis locais
3. Desvios Condicionais ocorrem com frequência
 - ↳ afeta PIPELINE!
4. Chamadas e Retornos de procedimentos.

CISC → Demorado dependendo da qtdade de parâmetros
↳ pilha

→ poucas níveis de aninhamento

→ lida com var. locais

Solução Proposta
~~passar para~~

Máquina RISC

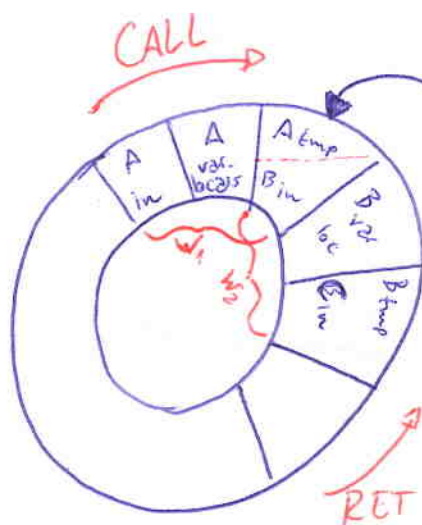
- grande nº de registradores
- Hardware Simplificado
- Projeto de Pipeline cuidadoso

→ muito mais rápido que ^{buscar} memória o tempo todo.
→ evita a troca e movimentação de dados
→ Hardware + Software (compilador fazer análise p/ uso eficiente)

ARQ
3

Janela de Registradores (RISC)

08/04/13



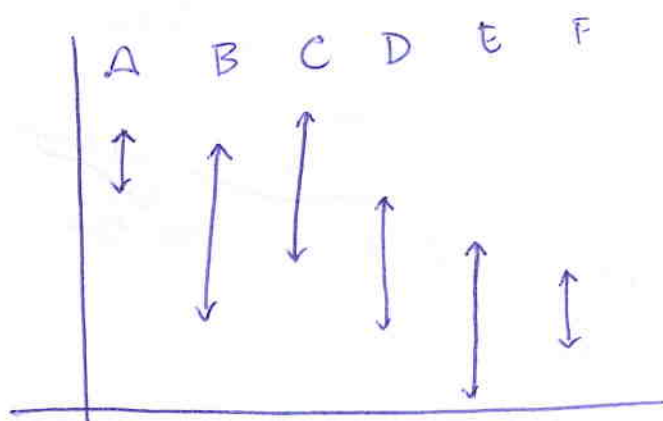
Isto vale a pena, pois ~~tem~~ no caso comum não existem muitas níveis de aninhamento.

o terceiro subconjunto que é usado p/ a saída de A e entrada de B. ~~B~~
Variáveis em comum para ambos

variáveis globais

~~variáveis globais~~

→ não é bom estarem ^{todos} no registrador o tempo todo pois dura o programa ~~todos~~ porém é possível otimizar.



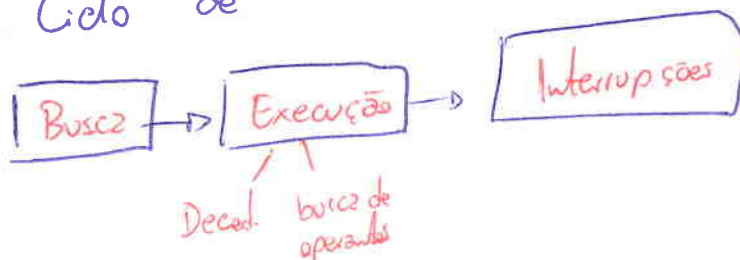
Armazenar 2 variáveis no mesmo registrador

ARQ
1

Caminho de Dados → Data Path

15/04/13

Ciclo de Instrução



ler memo por cima do data path

↳ entender o que está pronto no data path em cada estágio

↳ para que se entenda o funcionamento da pipeline

* visão geral de pipeline é fundamental

* E também Hazards e conflitos

Hazards → Conflito

1. Estrutural

2. Dados

3. Devio/Controle

imõe limitações ao pipeline

↳ não é possível realizar os estágios em paralelo sobre posições

1. → Restrição de Hardware → na dada implementação ^{uma característica} ~~há~~ colocada ^{pode causar} algum tipo de limitação

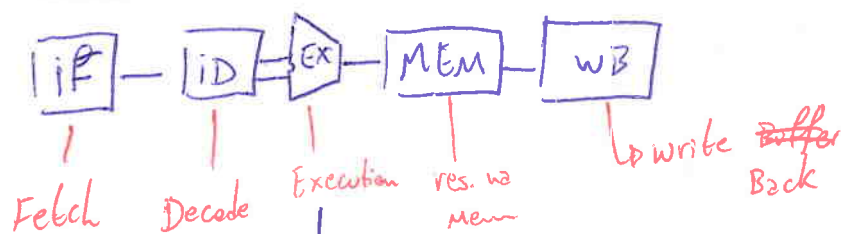
↳ ex: está realizando um load e por questão estrutural não é possível utilizar load ao mesmo tempo.

→ Alguém receberia prioridade e outro aguardaria

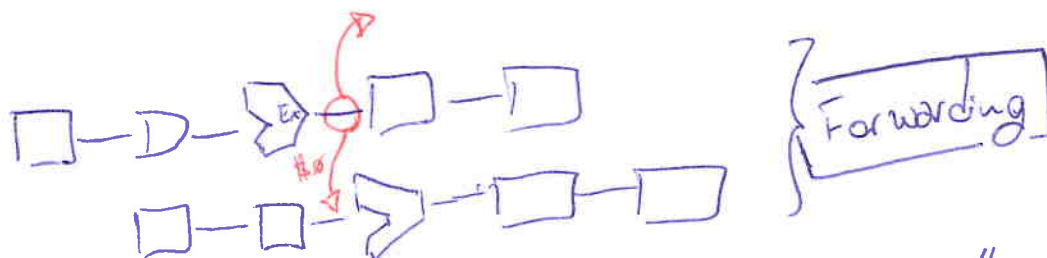
2. → add \$s0, \$t0, \$t1

sub \$t2, \$s0, \$t3

sub depende que o \$s0 esteja pronto para ser utilizada



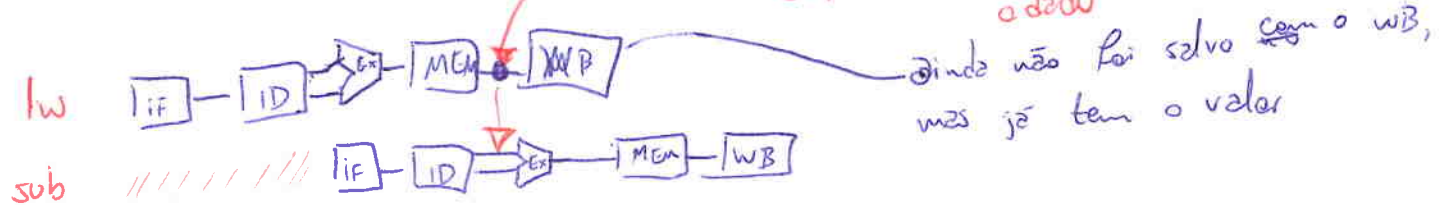
conteúdo ficará pronto na saída da execução



é uma alternativa para as melhorias que seriam feitas via software

lw \$S0, Z0(\$t1)

sub \$t2, \$S0, \$t3

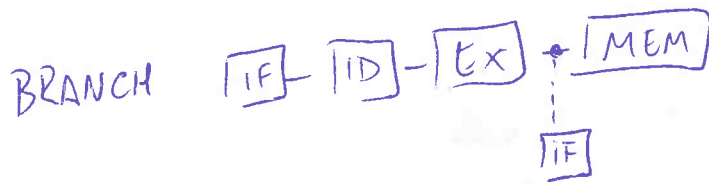


Forwarding → usar o que ficou pronto o mais rápido possível.

3. Caso você "chute" o ~~desvio~~ desvio e este estiver errado, então você sofrerá uma penalidade.

Atraso de desvio

↳ é aguardado até que EX aconteça, para que se tenha certeza e não haja penalidade.



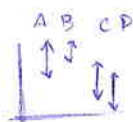
Risc → entender o porquê das características.

↳ otimizar operações mais utilizadas do CSC.

variáveis globais

→ compilador verificar melhor forma de se otimizar o uso dos registradores, reutilizando-os no tempo em que elas não são utilizadas mutuamente.

↳ **Coloração de grafos** → vizinhas estão sendo utilizadas ao mesmo tempo



CSC

→ facilitava ~~o~~ desenvolvimento de SO e Compiladores → pois complexidade ficava a cargo de hardware.

→ compilador ficava "bloqueado", pois não podia alterar funcionamento de funções "grandes" e complexas implementadas por hardware.

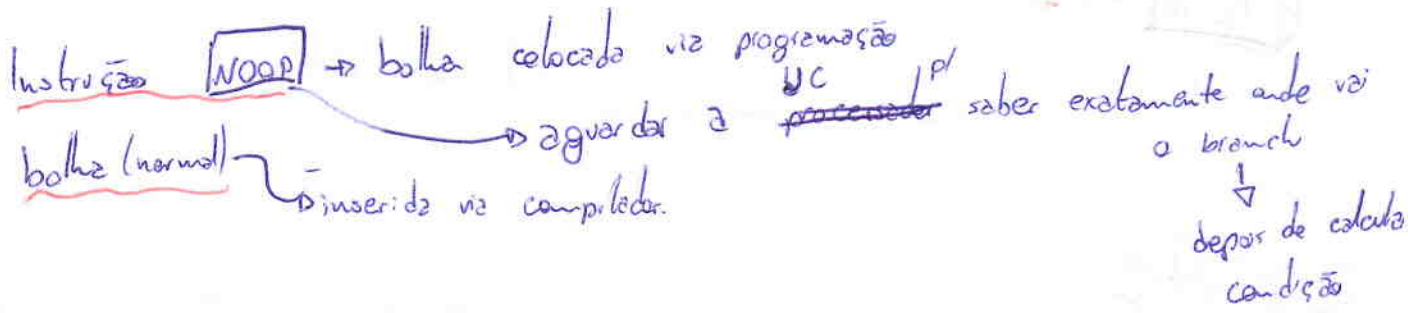
→ memória era limitada → programas menores eram melhores p/ evitar paginação e seeks no disco.

↳ mudar conforme memória fosse barata.

Características **RISC**

- 1 instrução por ciclo
 - operações de reg. para registrador. (maior velocidade)
 - poucas e simples modos de end.
 - poucas e simples formatos de instr.
 - sem microprograma
- para que sejam mais simples as instr.
implementada via hardware
sem necessidade de mais

Branch → precisa ir na mem. calcular e buscar o end.



- é colocado um noop depois do jump, pois ele só saberá o endereço da próx. opção após a execução. Evita de carregar o end. seguinte e ocupar a CPU. Coloca a bolha para já saber o valor.

Problemas

RISC e CISC não são diretamente comparáveis