Universidade Federal do Rio Grande do Sul
Instituto de Informática

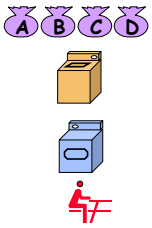
Organização de Computadores

Aula 10

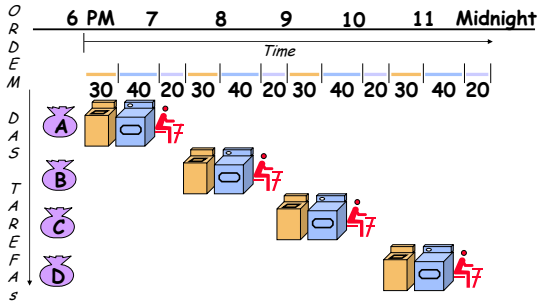
Pipelines
primeira parte

Pipelining: É evidente!

- Exemplo da lavanderia
- Ana, Beto, Carmen, Deoclécia tem cada um pacote de roupas para lavar, secar e dobrar
- Lavadora leva 30 minutos
- Secadora leva 40 minutos
- "Dobradora" leva 20 minutos

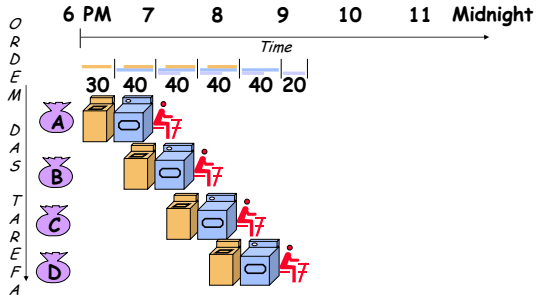


Lavanderia sequencial



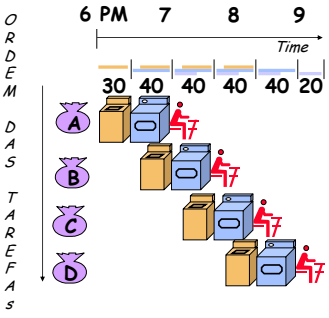
- Lavando em seqüência leva-se 6 horas para 4 cargas
- Se com pipeline, quanto tempo levaria?

Lavanderia com pipeline
Começa o trabalho ASAP



- Lavando com Pipeline tem-se 3.5 horas para 4 cargas

Lições do Pipeline



- Pipeline não ajuda no tempo de uma única tarefa, mas no throughput de uma carga de trabalho;
- Taxa de pipeline limitada pela tarefa mais lenta;
- Múltiplas tarefas operam concomitantemente
- Aceleração potencial: número de estágios do pipe
- Estágios desbalanceados reduzem aceleração
- Leva-se tempo para encher o pipe e esvaziar o mesmo

Pipeline em computadores

- Executar billões de intruções, então *throughput* é o que interessa;
- Qual o conjunto de instruções para fazer pipeline?
  - Tamanho de instruções variável ou todas as instruções de mesmo tamanho?
  - Operandos que acessam a memória em qq instrução ou acesso à memória somente durante load e store?
  - Registradores fonte em vários locais possíveis na instrução ou registradores sempre localizados no mesmo campo?

Inicialmente

- Pipeline é uma técnica de implementação de processadores que permite a sobreposição temporal das diversas fases da execução de instruções.
- Aplicada esta técnica ao nosso processador MIPS poderemos obter um desempenho melhor na execução das instruções lembrando que nossas instruções são executadas no nosso processador em 5 passos:
  - Busca da instrução na memória,
  - Leitura do registradores e decodificação,
  - Execução de uma operação ou cálculo de endereço,
  - Acesso a um operando na memória,
  - Escrita do resultado em um registrador.

Inicialmente

- Como parâmetros para nossa proposta de Pipeline, vamos empregar os seguintes tempos da nossa arquitetura tradicional.

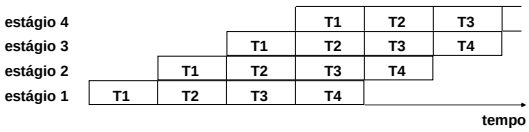
Tipo Instrução	Busca Instr	Ler Reg Dec	Oper UAL	Acesso Dado	Escrita Reg	TEMPO TOTAL
Load	2ns	1ns	2ns	2ns	1ns	8ns
Store	2ns	1ns	2ns	2ns		7ns
R	2ns	1ns	2ns		1ns	6ns
Branch	2ns	1ns	2ns			5ns

Pipelines
primeira parte

1. Introdução
2. Pipelines aritméticos
3. Pipelines de instruções
4. Desempenho
5. Conflitos de memória
6. Dependências em desvios

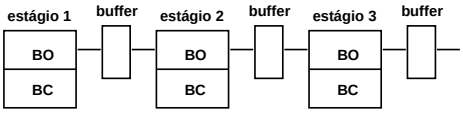
1. Introdução

- Objetivo: aumento de desempenho
  - divisão de uma tarefa em N estágios
  - N tarefas executadas em paralelo, uma em cada estágio
- Diagrama espaço – tempo



Introdução

- bloco operacional e bloco de controle independentes para cada estágio
- necessidade de buffers entre os estágios



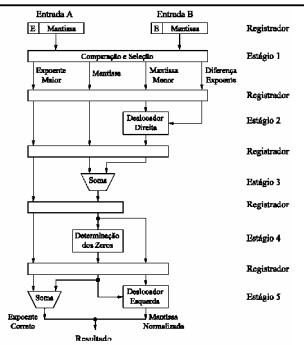
- pipelines de instruções
- pipelines aritméticos

2. Pipeline Aritmético

- O pipeline aritmético é empregado para acelerar as funções lógico e aritméticas das ULAs.
- É a divisão das operações aritméticas em suboperações.
- Foi a base para o surgimento dos Supercomputadores.
- Todos microprocessadores modernos possuem pipeline aritmético.

- Próximo slide um Somador de Ponto Flutuante com 5 estágios
- Os cinco estágios:
  - comparação dos operandos A e B,
  - ajuste da mantissa,
  - soma dos operandos A e B
  - verificação dos zeros da soma
  - ajuste do expoente final
- Resulta no final um expoente e uma mantissa

Pipeline Aritmético



Pipeline Aritmético

- **Exemplo: subtração em ponto flutuante**
- **Comparação de operandos**

0.157	x 10^6	−	0.842	x 10^5

- **Ajuste de mantissa**

0.157	x 10^6	−	0.0842	x 10^6

- **Resultado da Soma**

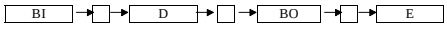
0.0728	x 10^6

- **Verificação dos Zeros**
- **Ajuste de Expoente**

0.728	x 10^5

3. Pipeline de Instruções

- **Hoje todos microprocessadores possuem pipeline de instrução.**
- **O básico é composto:**
  - **busca de instrução**
  - **decodificação**
  - **busca de operando**
  - **execução**

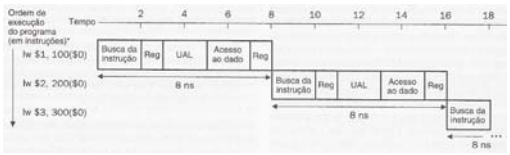


Pipelines de Instruções

- **Evolução no número de estágios:**
  - **2 estágios**
    - **fetch / decodificação, execução**
  - **3 estágios**
    - **fetch, decodificação / busca de operandos, execução**
  - **4 estágios**
    - **fetch, decodificação / busca de operandos, execução, store**
  - **5 estágios**
    - **fetch, decodificação / cálculo de endereço de operandos, busca de operandos, execução, store**
  - **6 estágios**
    - **fetch, decodificação, cálculo de endereço de operandos, busca de operandos, execução, store**
- **estágio só de decodificação é usual em processadores CISC**

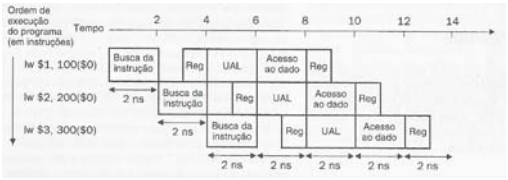
Pipeline da nossa Arquitetura MIPS

- **Sem uso de Pipeline**
  - **Total após 3 instruções 24ns**



Pipeline da nossa Arquitetura MIPS

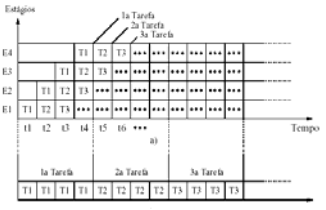
- **Com Pipeline**
  - **Clock deve ser da operação mais lenta, no caso 2ns**
  - **Total após 3 instruções 14ns**



- **Em condições ótimas o ganho de um pipeline é igual ao número de estágios deste.**

4. Desempenho

- **existe um tempo inicial até que o pipeline "encha"**
- **cada tarefa leva o mesmo tempo, com ou sem pipeline**
- **média de tempo por tarefa é no entanto dividida por N**



Desempenho

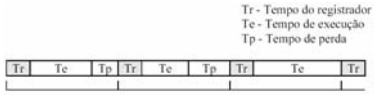
- O tempo do relógio é dado pelo tempo do estágio mais lento
  $T = \max[T\,E\,i\,]\,j + TR$

- **Tempo total com pipeline = N + (s – 1)**
- **Tempo total sem pipeline = s N**
  - **s tarefas**
  - **N estágios**
  - **primeira tarefa: N ciclos de relógio**
  - **s – 1 tarefas seguintes: s – 1 ciclos de relógio**

$$\text{Speed–up teórico} = \frac{s\,N}{N + (s - 1)} \qquad \text{limite} = N$$

Problemas no desempenho

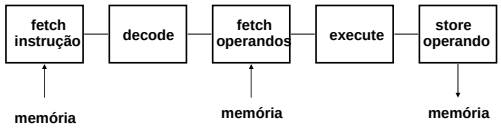
- como dividir todas as instruções num mesmo conjunto de estágios?
- como obter estágios com tempos de execução similares?



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

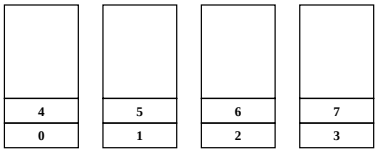
5. Conflitos de memória

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



- soluções
  - caches separadas de dados e instruções
  - memória entrelaçada

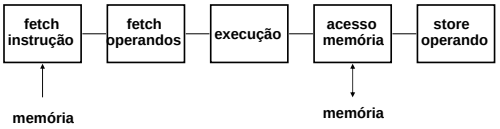
Memória entrelaçada



- 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

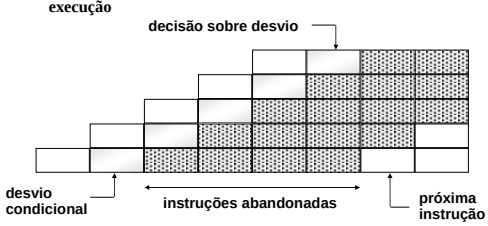
Processadores RISC

- 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 caches de dados e instruções são separadas, não há nenhum conflito de acesso à memória



6. Dependências em desvios

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



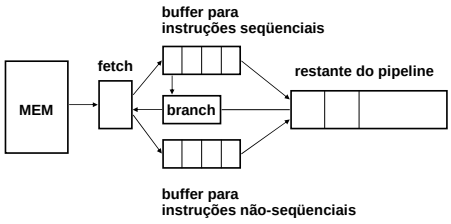
Dependências em desvios

- 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?

Técnicas de tratamento dos desvios condicionais

1. Executar os dois caminhos do desvio
   - buffers paralelos de instruções
2. Evitar o problema
   - "delayed branch"
3. Prever o sentido do desvio
   - predição estática
   - predição dinâmica

Buffers paralelos de instruções



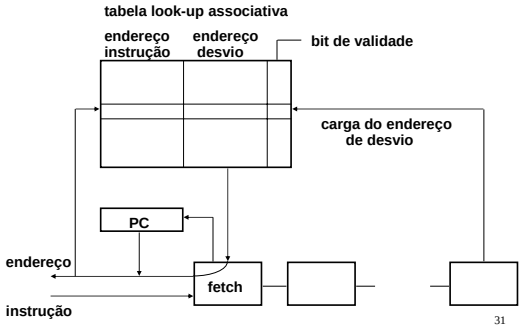
Delayed branch

- Compilador reorganiza as instruções do programa
  - Desloca instruções que não afetam o resultado para depois do desvio. É uma forma de adiantar a execução do desvio diminuindo o número de ciclos perdidos quando do descarte
  - Deve levar em conta as dependências entre instruções e manter a semântica do programa
- Funciona desde que instruções sejam independentes
- 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

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

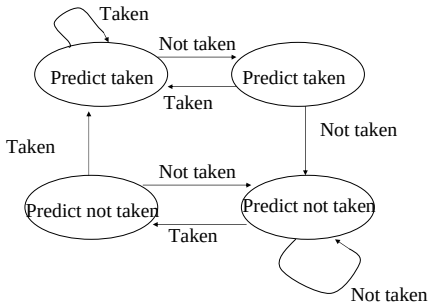
Predição dinâmica



Predição dinâmica

- 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

Esquema de predição de 2 bits



Predição dinâmica

- **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**