Microprocessadores

Execução em Paralelo
"Pipelines"

António M. Gonçalves Pinheiro

Departamento de Física
Universidade da Beira Interior
Covilhã - Portugal

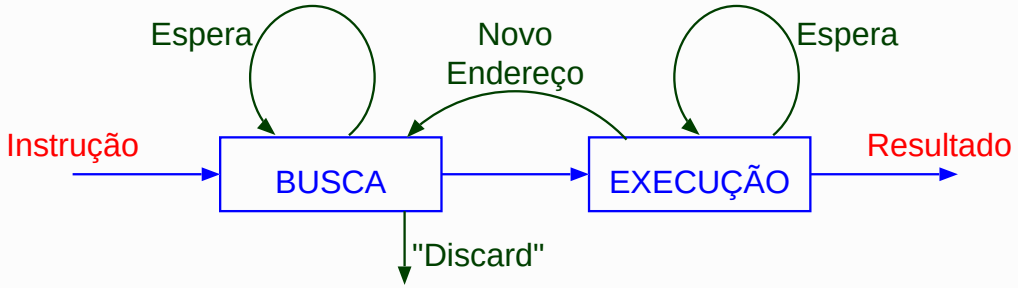
pinheiro@ubi.pt

"Pipelines"

Pipelines de Instrucções

Instrucções são divididas em diferentes Estágios

Exemplo de dois estágios: **BUSCA** e **EXECUÇÃO**



- Quando uma instrucção está a ser executada a seguinte já está no ciclo de busca.
- Saltos condicionais podem levar a que a instrução que está no ciclo de busca, não seja a executada ("Discard").
- Muitas instrucções resultam em tempos de execução maiores do que a busca, logo optou-se por partir a execução em múltiplos estágios.

"Pipelines"

Pipelines de 6 estágios

FI - Fetch Instruction
Lê a próxima Instrucção.

DI - Decode Instruction
Determina o Código e o Operando.

CO - Calculate Operands
Calcula o endereço de cada operando.

FO - Fetch Operands
Busca cada Operando da memória.

EI - Execute Instruction
Executa a operação

WO - Write Operand
Armazena o resultado na memória

Nota: Nem todas as instrucções usam todos os estágios. No entanto, o espaço temporal tem que ser mantido para manter sincronismo.

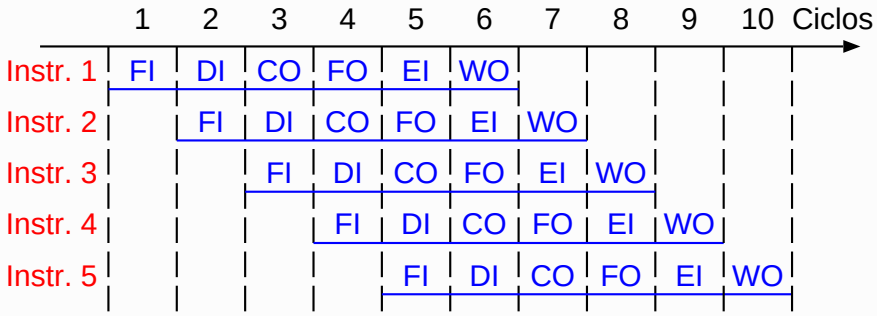
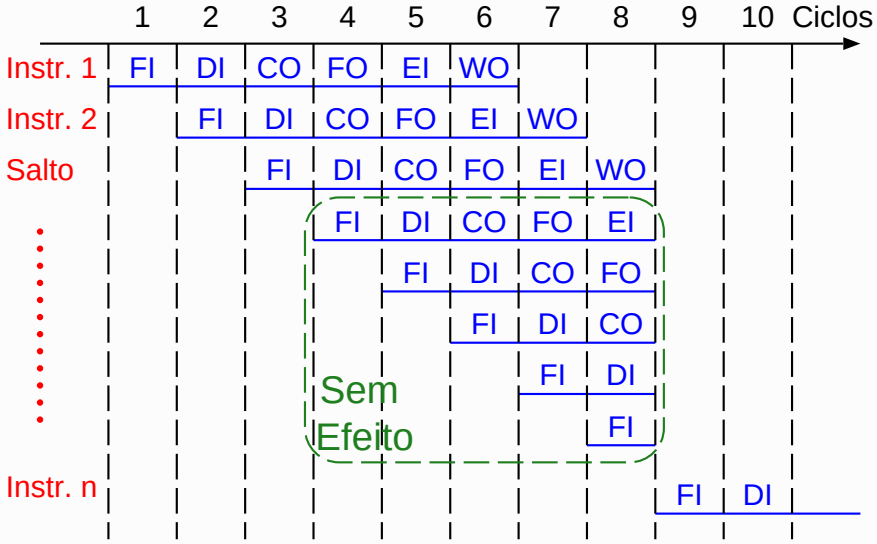


Diagrama Temporal



Efeito de Salto Condicional

"Pipelines"

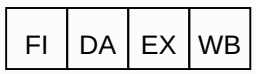
Atrasos nas Pipelines

Sem atrasos, as instrucções levam um período de relógio em média a serem executadas.
Atrasos podem ser provocados por:

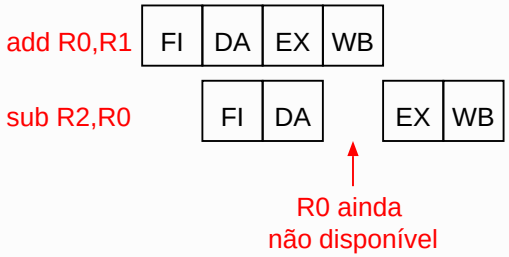
- **Problemas Estruturais**
  Quando é impossível executar dois estágios em paralelo porque provoca conflitos no hardware.
- **Atraso nos Dados**
  Existe um dado que é necessário mas ainda não está disponível.

Exemplo de Pipeline de 4 estágios

FI - Busca de Instrução
DA - Descodifica a Instrucção e Calcula Endereço
EX- Executa Instrução
WB - "Write Back"



Pipeline



Atraso

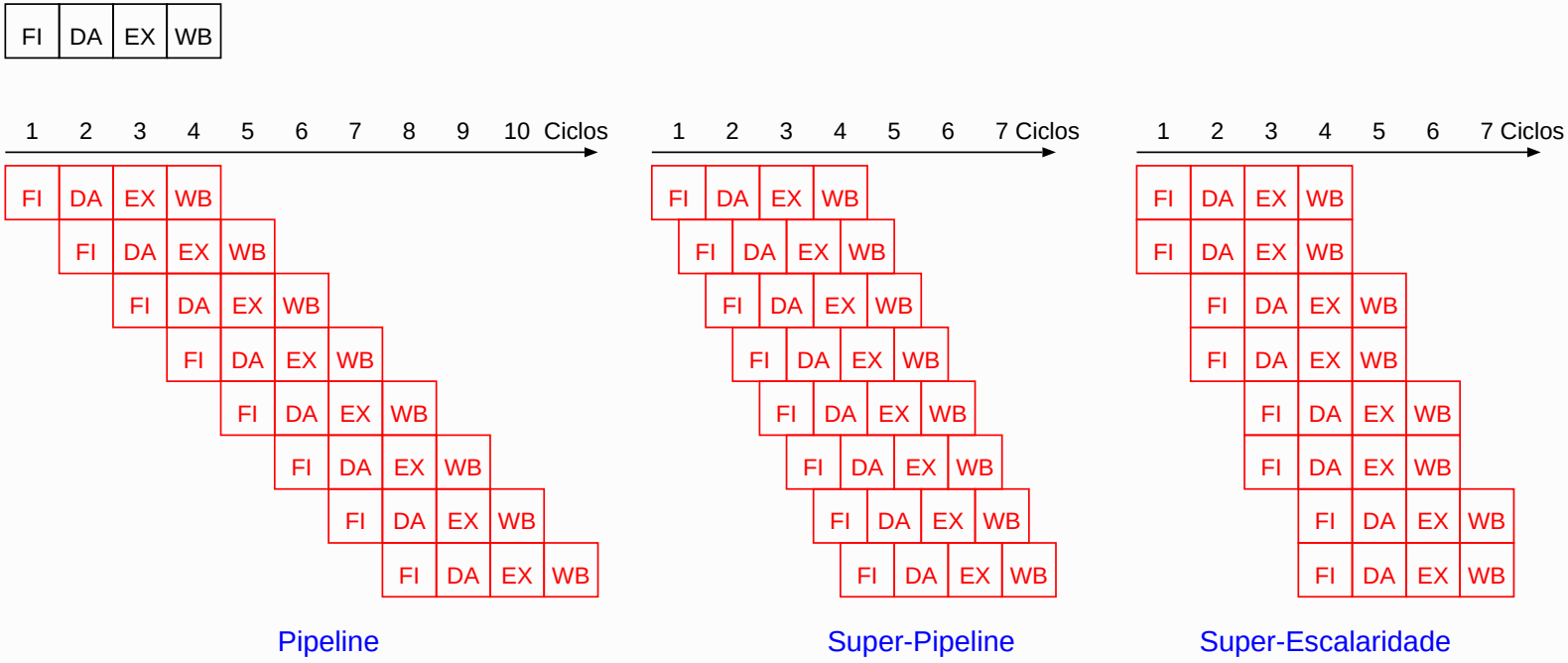
- **Saltos Condicionais**

"Pipelines"

Redução do atraso provocado pelos Saltos Condicionais nas Pipelines

- "Multiple Streams"
  Duas pipelines executam as duas possibilidades de salto. Exige duplicação de recursos.
- "Prefetch branch target"
  Alvo do salto é Pre-fetched e se a decisão é fazê-lo, já está disponível para o controlador.
- "Loop buffer"
  Um "buffer" (memória de alta velocidade) contem as instrucções executadas mais recentes.
  - reduz tempo de acesso à memória caso as instruções estejam no "buffer";
  - se o alvo do salto condicional estiver próximo, deverá estar no "buffer";
  - lida especialmente bem com ciclos e iterações.
- Predição do salto
  - Dependendo do tipo de salto podemos optar por **Salta sempre/Nunca Salta**
  - "Switch" que comuta entre **Salto/Não Salto** dependendo do passado recente.
  - Tabela histórica de saltos (que assim são mantidos em cache)
- Rearranjo de instrucções (Delayed Branch)
  Instrucções são reordenadas de forma a que operações que deviam ser anteriores sejam executadas em paralelo com a execução do salto.

Arquitecturas Superescalares



Notas:

- Fazem **reordenamento** de instrucções na tentativa de limitar efeitos das depêndencias
- Usam execução de instruções **especulativa** para conseguirem ser eficientes.
- Requerem múltiplas capacidades na CPU, como sejam múltiplas ALU, FPU, etc.

Arquitecturas Superescalares

Paralelismo ao Nível da Instrucção

Pode ser aumentado de duas formas

1. Aumenta-se o comprimento da pipeline, subdividindo níveis.
   O número de isntrucções na pipeline aumenta, mas como em cada nível se pode fazer menos, pode-se aumentar a velocidade.
2. Replicando-se as unidades internas da CPU (buses, ALUs, FPUs)
   "Multiple Issue":
   - *Estática*: As decisões são tomadas no compilador/escrita do programa.
   - *Dinâmica*: As decisões são tomadas no Microprocessador durante a execução do programa.

"Pipeline Multiple Issue"

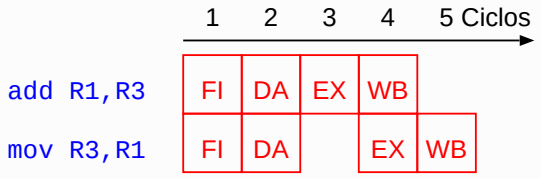
1. Microprocessador determina quantas instrucções podem ser colocadas em cada ciclo de relógio (slot temporal).
2. Lidar com "Hazards" de dados e controlo:
   - nos "Multiple Issue" estáticos, têm que ser manipulados pelo compilador.
   - nos "Multiple Issue" dinâmicos, o controlador têm que alivia-los ao máximo.

Arquitecturas Superescalares

Limitações da Superescalaridade

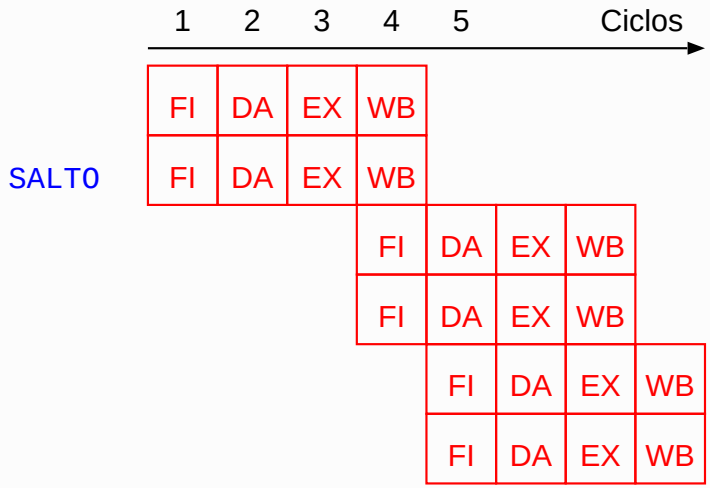
"True Data Dependency"

Tem que se esperar o resultado de uma operação para se poder executar a outra.



"Procedural Dependency"

Saltos são sempre um problema!
Nota: Nos RISC, instrucções têm dimensão fixa.
Se for CISC ainda se acrescenta o problema de controlar a dimensão da instrucção.



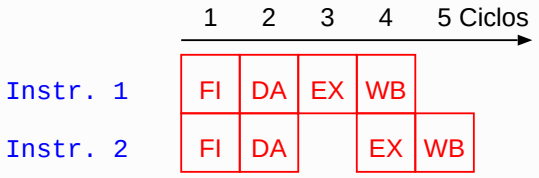
Arquitecturas Superescalares

Limitações da Superescalaridade

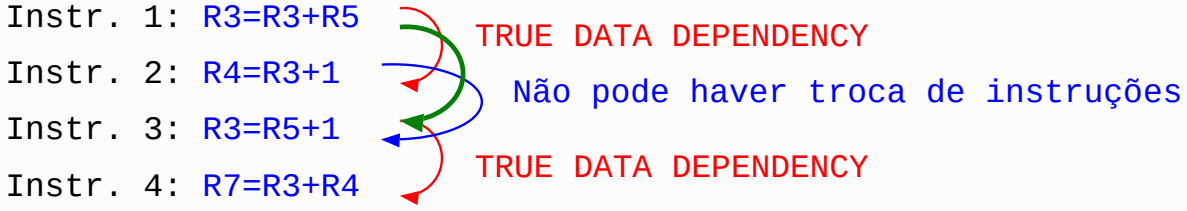
Conflito de recursos

Acontece quando duas instrucções requerem o mesmo recurso ao mesmo tempo (tal como o mesmo barramento, memória, cache, unidades funcionais, ALU, FPU).

Este problema é semelhante ao "True Data Dpendency" mas pode ser facilmente eliminado através da duplicação de recursos.

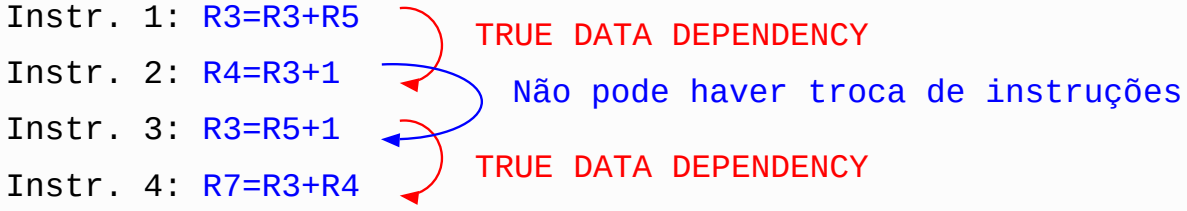


"Output Dependency" ou "Read-Write Dependency"



Dependencia da Saída refere-se que a instrucção 1 3 alteram o mesmo registo (R3), impedindo troca de instrucções.

"Anti Dependency" ou "Write-Write Dependency"



Anti Dependencia refere-se que a instrucção 3 altera o que a instrucção 2 necessita (R3).

Arquitecturas Superescalares - Técnicas para Melhorar Eficiência

Especulação

O compilador ou o processador tentam adivinhar a saída de uma instrucção para retirar qualquer tipo de dependência que resultam da execução de outras instrucções.

Por exemplo, num salto tenta adivinhar as instrucções que deverão ser executadas depois, executando-as!

Renomeação de registos

De forma remover anti-dependências.

Execução fora de ordem - *"Out of Order Execution"*

Executar instrucções seguintes independentemente de uma anterior que esteja bloqueada por uma dependência. Na prática trata-se de reordenar as instrucções.