Universidade de Brasília
Departamento de Ciência da Computação

Aula 16
Implementação MIPS
MIPS Pipeline - Conceitos

Objetivo

- Apresentar técnicas para aumento de desempenho de uma única tarefa em único processador usando ILP (*Instruction Level Parallelism*):
  - ***Pipelining*** : Técnica de projeto onde o *hardware* processa mais de uma instrução de cada vez, sem esperar que uma instrução termine antes de começar a próxima.
  - **Superpipeline** : Técnicas para aumento da eficiência do pipeline.
  - **Superescalar**: Uso eficiente de múltiplas unidades funcionais.

- Outros níveis de paralelismo: Aumenta a eficiência em máquinas multi task.
  - Multithread : Várias tarefas independentes simultaneamente, aproveitando-se das "falhas" do pipeline e/ou de estrutura superescalar.
  - Multicore: Vários núcleos em 1 chip, compartilhando memórias caches.
  - Multiprocessor: Sistema multiprocessado/distribuído. Diferentes computadores.

Pipelining: Conceito Básico

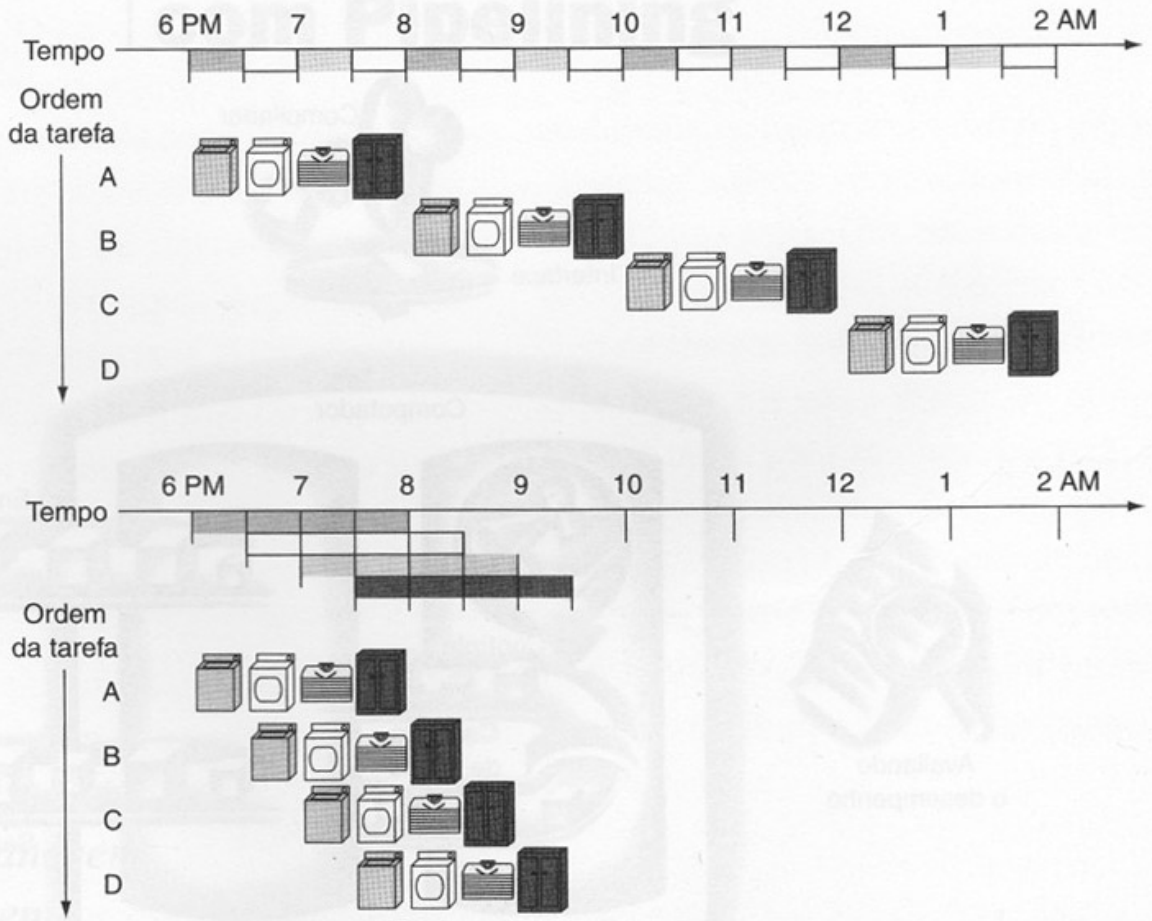
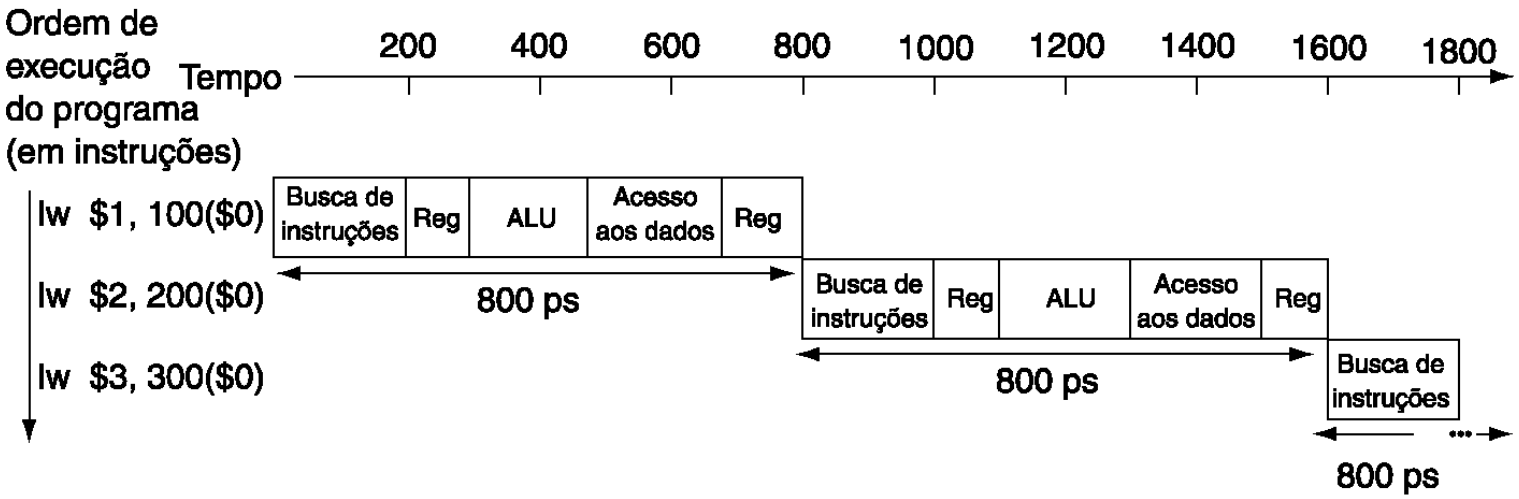
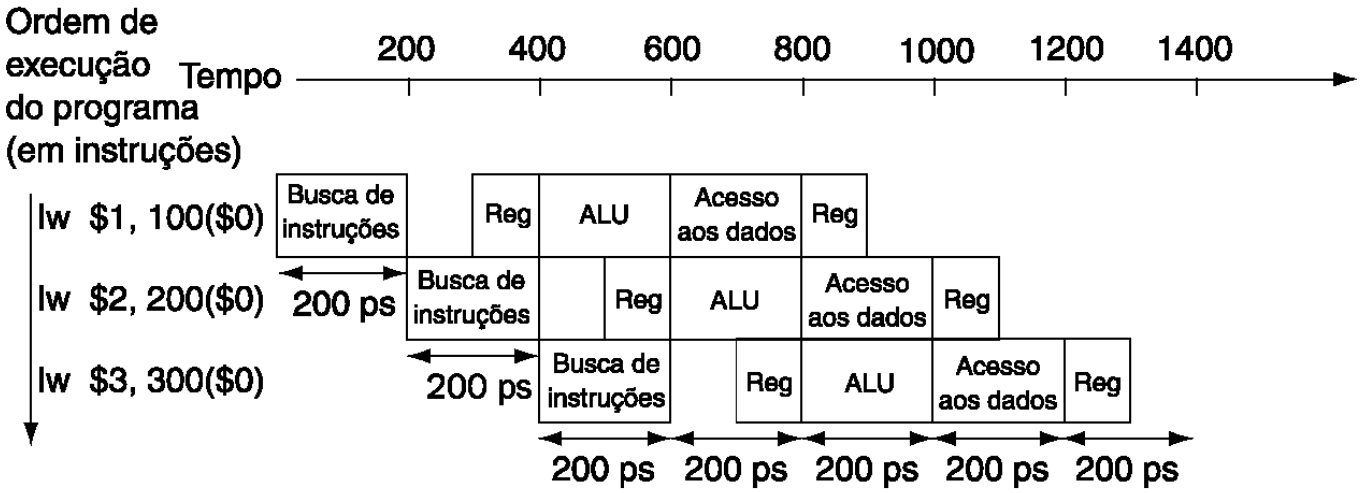


FIGURA 6.1 A analogia da lavagem de roupas para pipelining. Ana, Beto, Catarina e Davi possuem roupas sujas para serem lavadas, secadas, passadas e guardadas. O lavador, o secador, o passador e o guardador levam 30 minutos para sua tarefa. A lavagem seqüencial levaria 8 horas para quatro trouxas de roupas, enquanto a lavagem com pipeline levaria apenas 3,5 horas. Mostramos o estágio do pipeline de diferentes trouxas com o passar do tempo mostrando cópias dos quatro recursos nessa linha de tempo bidimensional, mas na realidade temos apenas um de cada recurso.

Uniciclo vs Pipeline





Análise

- Latência: 5 ciclos. Vazão: 1 instrução/ciclo
- Qual a aceleração ideal?
  - Para estágios balanceados, condições ideais e grande número de instruções:

$$Tempo\ entre\ instruções_{com\ pipeline} = \frac{Tempo\ entre\ instruções_{sem\ pipeline}}{Número\ de\ estágios\ do\ pipeline}$$

  - Porque estágios balanceados? (lacunas)
  - Por que condições ideais? (hazards)
  - Porque grande número de instruções? (overhead final)

Ex.: 3 instruções: 2400 vs 1400

1.000.000 instruções: 1.000.000x800 vs 999.999x200+1000

Pipeline

O que facilita o pipeline:

- ***Todas instruções têm o mesmo comprimento.***

  Obs.: IA-32 instruções de 1 a 17 bytes, dificulta o cálculo do endereço da próxima instrução e a leitura da própria instrução da memória.

- ***Poucos formatos de instruções.***

  Obs: Tipo-R, I , J operandos “quase” nas mesmas posições

- ***Operandos em memória só aparecem em loads e stores.***

  Obs.: Pode-se usar a ULA para cálculo de endereço.

O mesmo não vale para busca de operandos da ULA da memória (IA-32)

O que complica (hazards):

- ***Riscos Estruturais***
- ***Riscos de Controle***
- ***Riscos de Dados***

Hazard Estruturais

- Hardware pode não admitir a combinação de instruções em um mesmo ciclo de clock.
- Ou a unidade funcional está ocupada no momento.

1) O que aconteceria se a só houvesse 1 pessoa na lavanderia?

2) Considerando apenas 1 memória para dados e instruções, como ficaria uma quarta instrução no exemplo dado?

Hazard de Dados

- Pipeline precisa ser interrompido por que uma etapa precisa esperar até que outra seja concluída.

1) O que fazer se 1 pé de meia seu foi lavado junto com as roupas de seu colega? (é claro que é exagero!)

2) Exemplo:

   add $s0,$t0,$t1

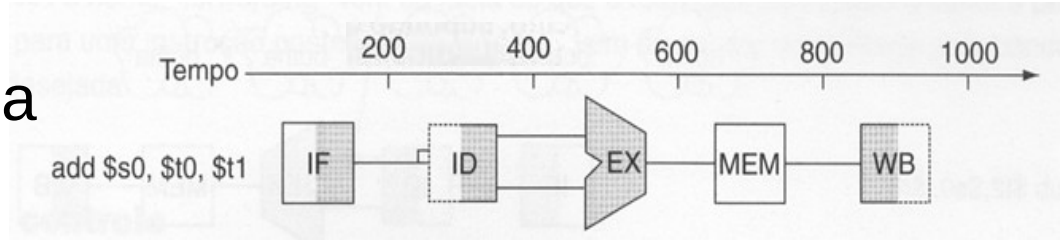
   sub $t2,$s0,$t3

Soluções:

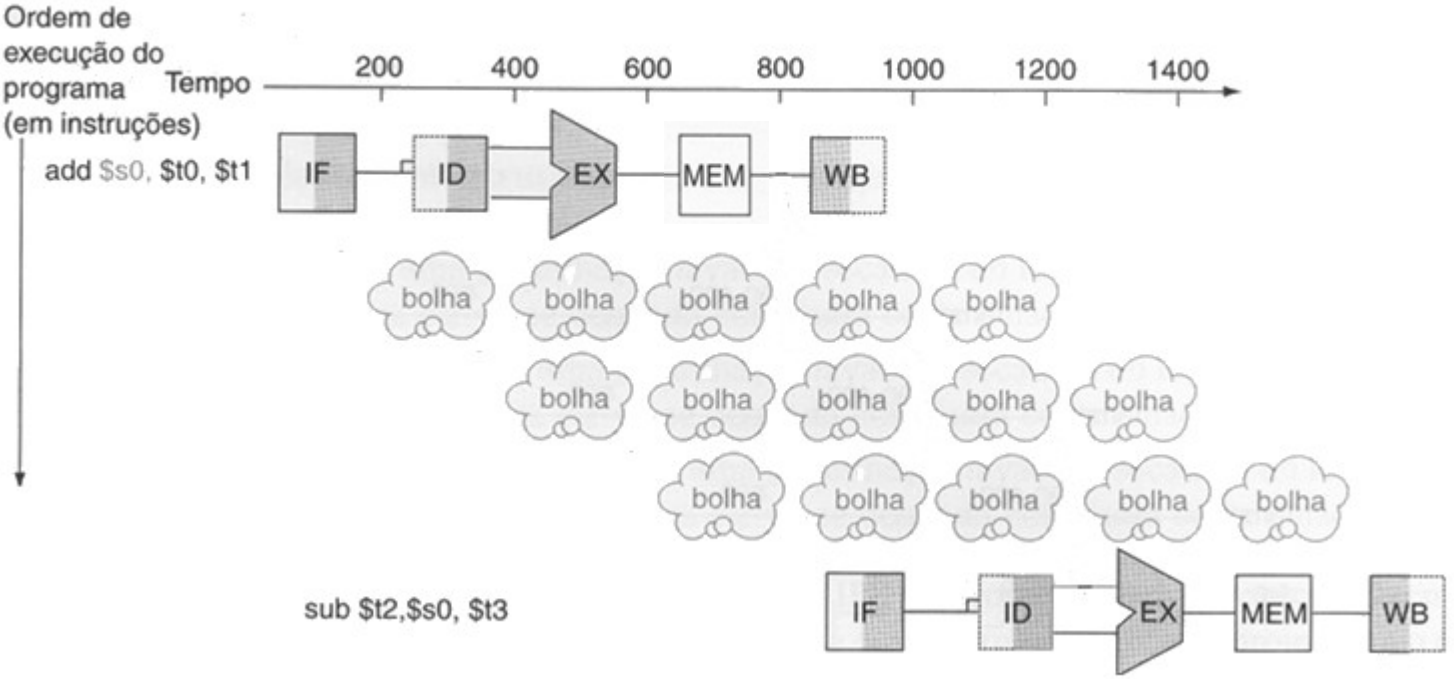
- Inserção de Bolhas
- Execução fora de ordem (compilador e processador)
- Forwarding ou Bypassing

Inserção de Bolhas

- Representação gráfica



- Bolhas
  - Sabendo que o registrador $s0 é escrito no 5º estágio



Execução Fora de Ordem

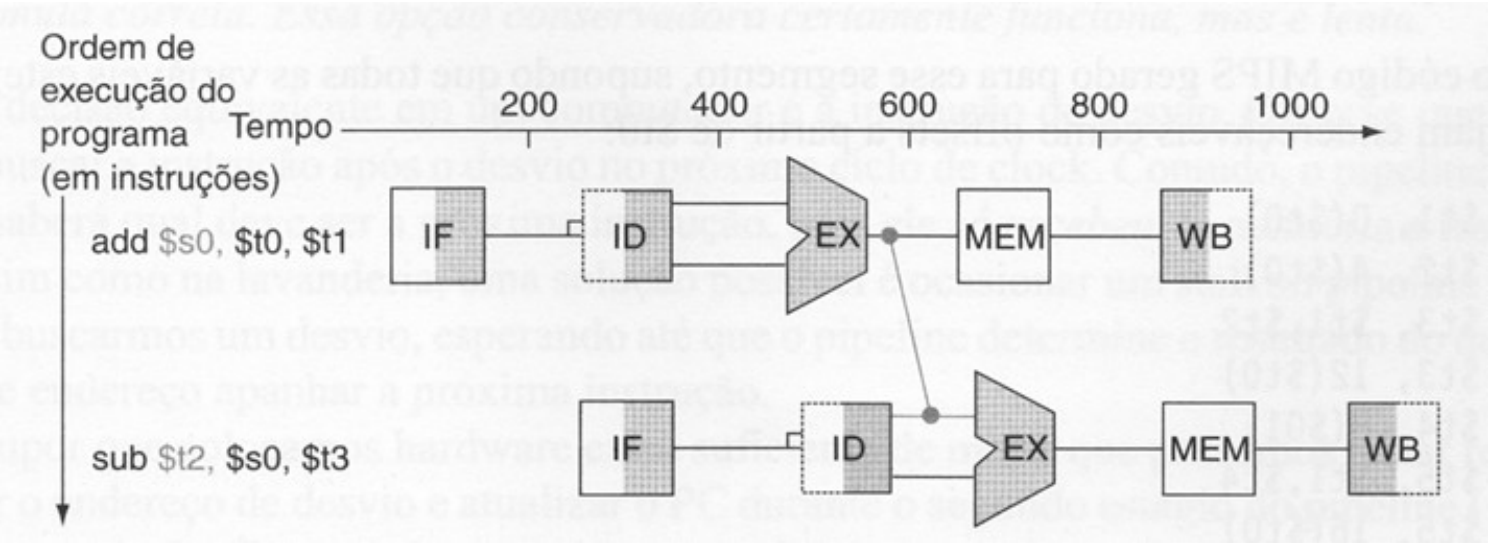
- Aproveitar os espaços das bolhas para tarefas úteis que sejam independentes.
- Alterar o algoritmo sem alterar o resultado!

  - Ex.:

```
add $s0,$t0,$t1
ori $t7,$t7,100
add $s1,$zero,$zero
addi $t5,$t4,256
sub $t2,$s0,$t3
```

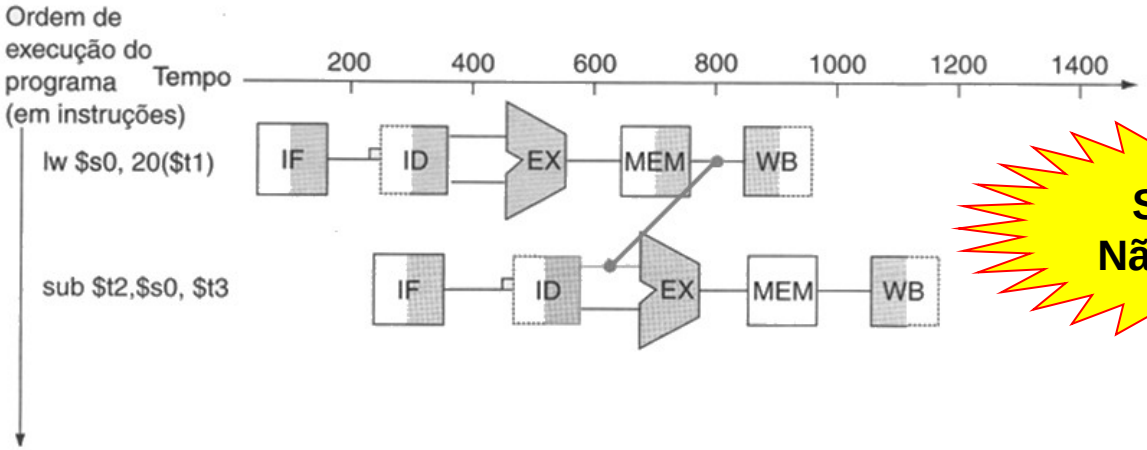
Forwarding

- Observação: não é necessário esperar que a instrução termine antes de tentar resolver o hazard de dados.

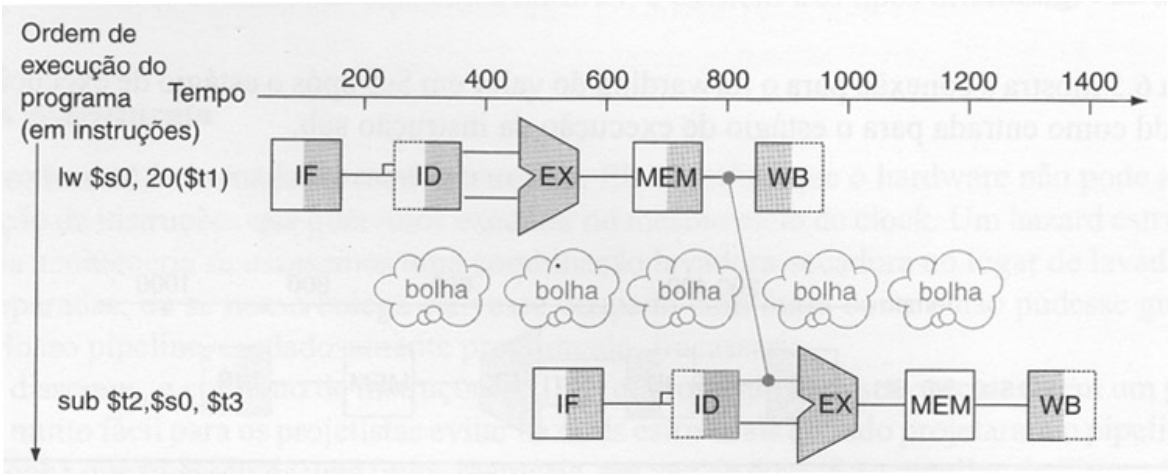


Forwarding

- E se fosse um load?



Solução



- Exemplo:

A=B+E;

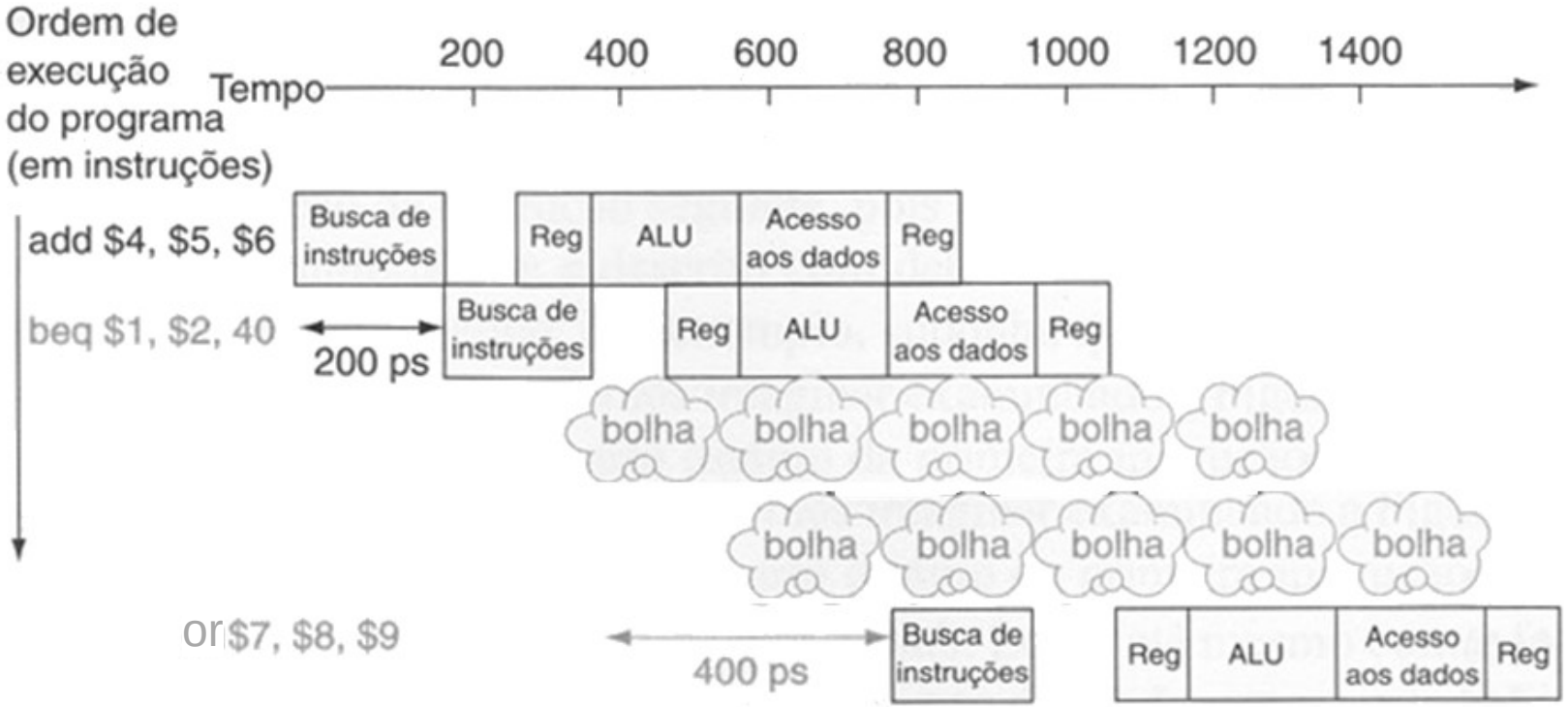
C=B+F;

```
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)
```

Identifique os hazards.
Como solucionar utilizando apenas bolhas?
Quantos ciclos são necessários com latência?
Como solucionar de maneira mais eficiente?
Quantos ciclos são necessários sem latência?

Hazard de Controle

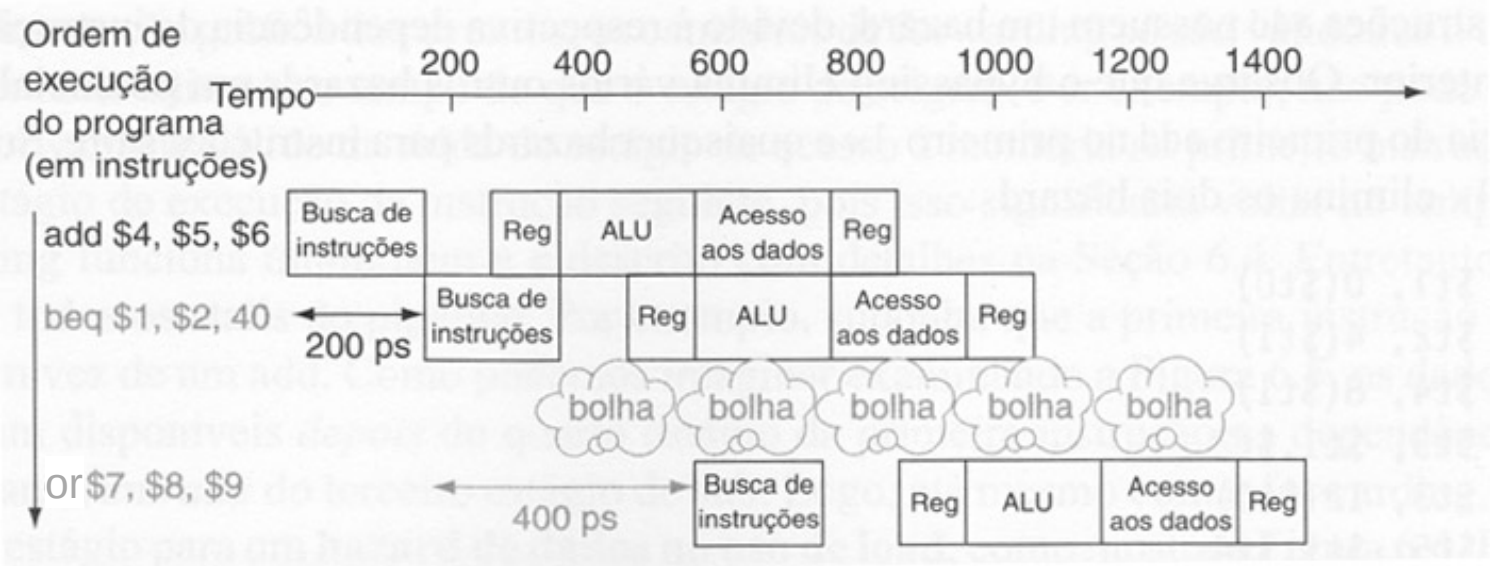
- Necessidade de tomar uma decisão com base nos resultados de uma instrução enquanto outras estão sendo executadas.



BEQ é avaliado pela ULA no 3º estágio => Necessita 2 bolhas!!!

Hazard de Controle

- Alterando o caminho de dados para que o BEQ seja avaliado no 2º Estágio => Economiza 1 Bolha!!



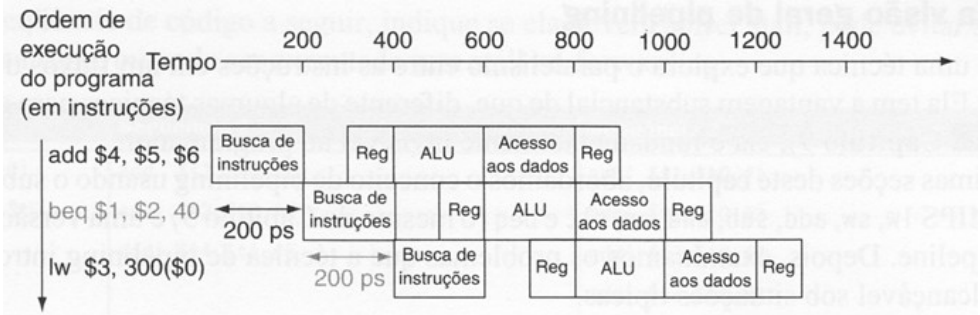
- *Delayed Branch:* ocupar a bolha com uma instrução útil, equivalente a atrasar a execução do Branch

  No exemplo: add $4,$5,$6 no lugar da bolha

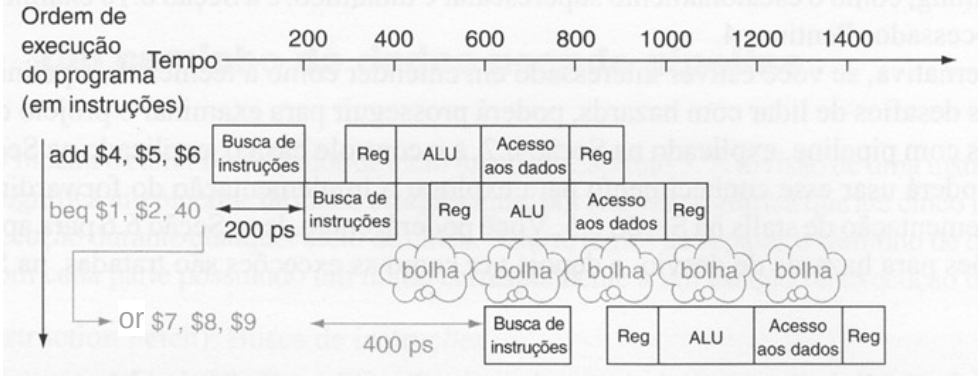
Hazard de Controle: Previsão de Desvios

- Prevendo estaticamente o desvio como não tomado

Caso de:
Desvio não tomado
Acerto!



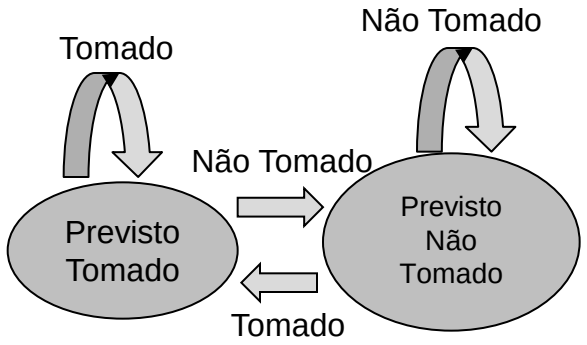
Caso de:
Desvio tomado
Erro! Necessita dar um *flush* na instrução seguinte



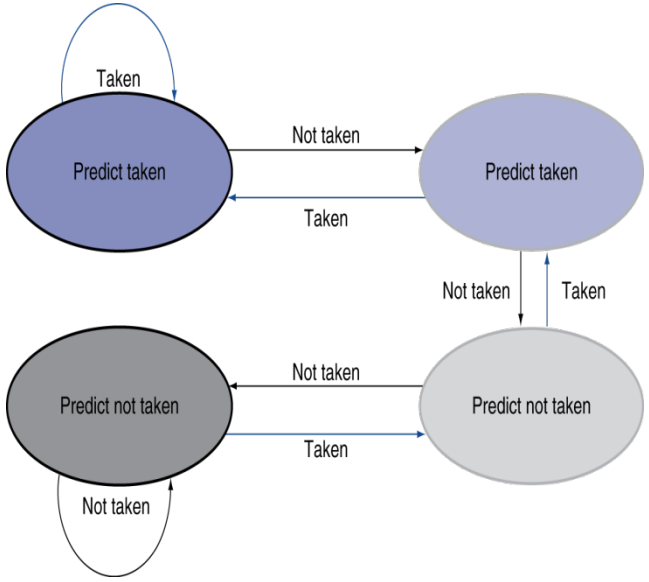
Hazard de Controle: Previsão de Desvios

- Prevendo Dinamicamente o desvio
  - •Ex.: Endereço é anterior? Prevê tomar o desvio

- 1 Bit
  - Considera que a previsão é a mesma da última escolha



- 2 Bits
  - Considera uma variação conforme:

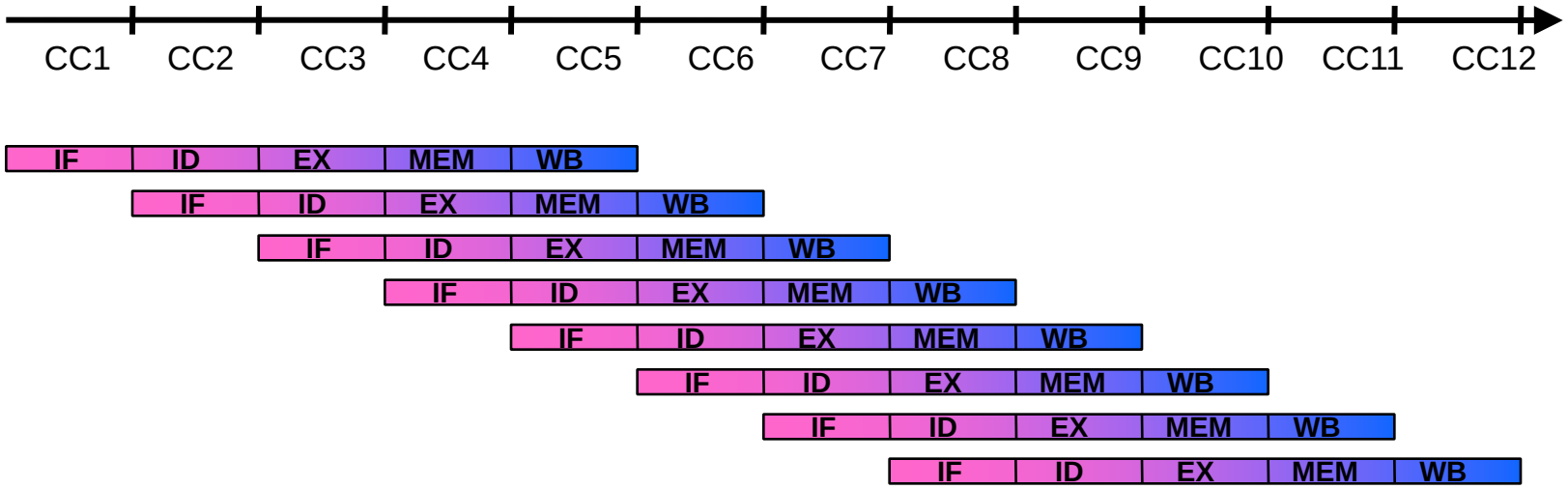


- Tabela
  - Considera uma tabela com contador do histórico

Mostrar Tool / BHT Simulator do Mars

Acelerando máquinas com *pipeline*

- A velocidade máxima de uma máquina com *pipeline* é de uma instrução por ciclo de *clock*. Correto?



Técnicas para iniciar múltiplas instruções em 1 ciclo de clock:

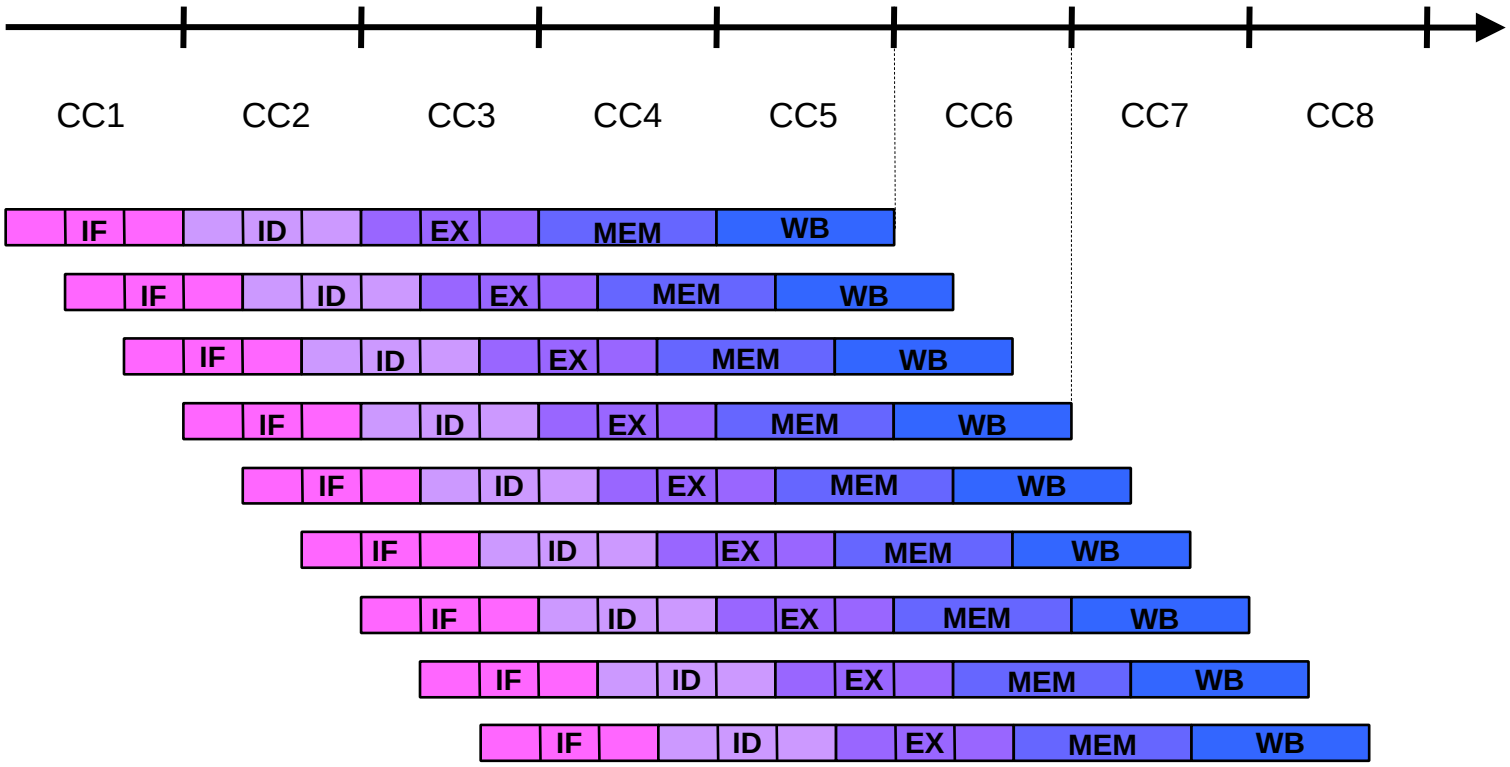
- Estruturas *superpipeline*
- Estruturas *superescalares.*

Acelerando máquinas: *Superpipeline*

- A estrutura *Superpipeline* subdivide cada estágio do *pipeline* em subestágios, e multiplica internamente a frequência de *clock*.

- Cada (sub)estágio continua executando uma instrução por *clock*. Mas como o *clock* interno é multiplicado, o *pipeline* pode aceitar duas ou mais instruções para cada *clock* externo.

Superpipeline

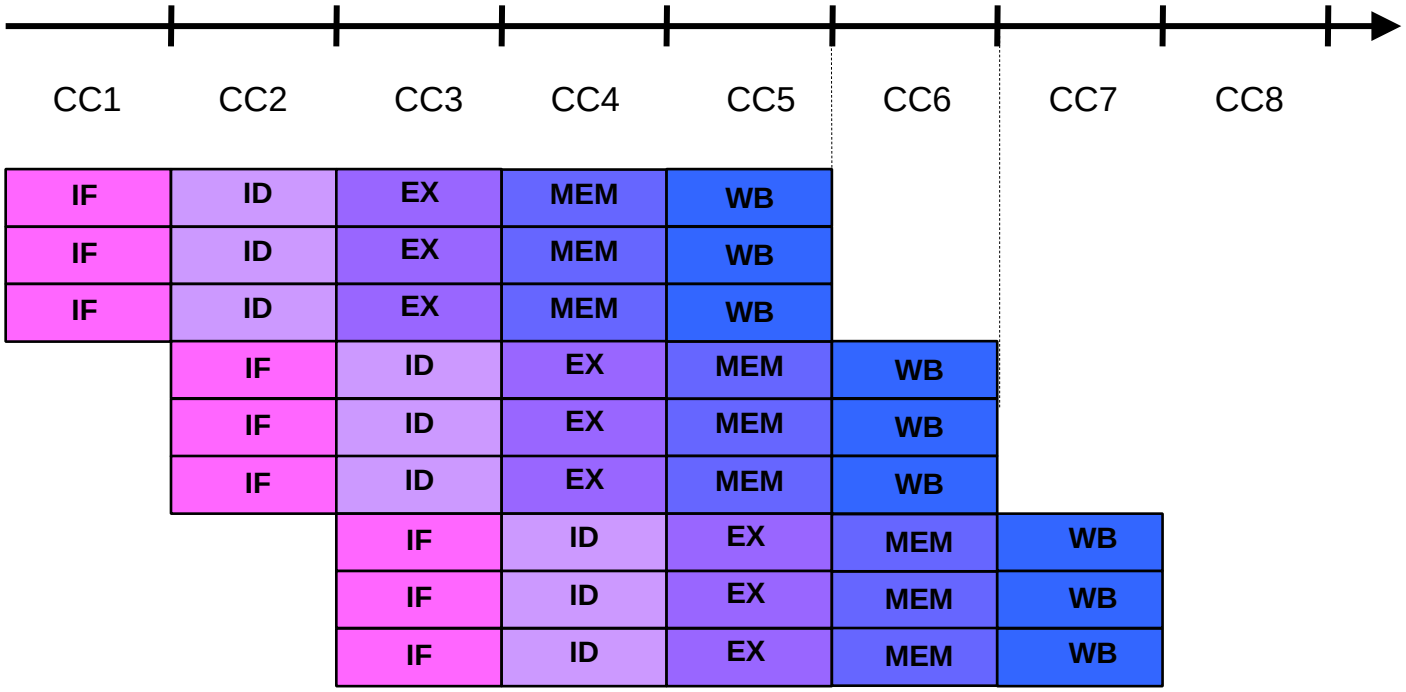


Acelerando máquinas: *Superescalar*

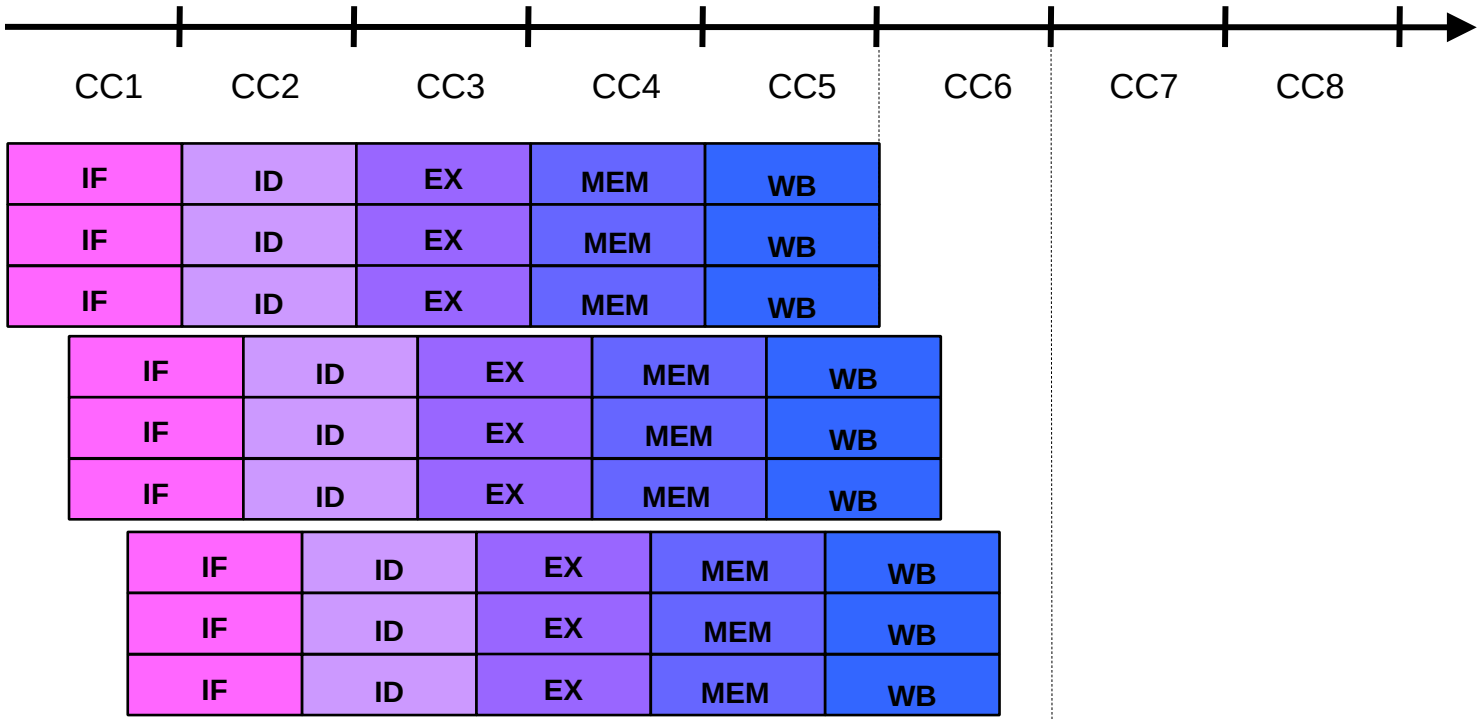
- A estrutura *superescalar* contém múltiplas unidades funcionais que são capazes de fazer a mesma tarefa.

- Isto permite ao processador executar várias instruções similares concorrentemente, pelo roteamento das instruções às unidades de execução disponíveis.

Pipeline Superescalar



Superpipeline Superescalar



Quantas instruções são iniciadas a cada ciclo de clock externo?

Conclusão

- Todos processadores modernos possuem pipeline complexos:

  80486 : 5 estágios (CISC , puro escalar)

  Pentium: 5 estágios + 2 ULAs inteiros

  Pentium II: 10 estágios + superescalar (núcleo RISC, nível de microinstruções)

  Pentium III : 10 estágios

  Pentium IV: 20 estágios de pipeline

  Pentium D: 31 estágios (Maior frequência, maior num. bolhas)

  Core2 : 14 estágios

  Core i3,i5,i7 : 16 estágios

  Athlon 64: 12 estágios

  - PowerPC e Pentium: tabela de histórico de desvios

  - Escalonamento de instruções: Que instrução executar agora?

    - Historicamente: tarefa do compilador

    - Hoje em dia: o hardware do processador já realiza.