Universidade Federal do Rio de Janeiro
Informática DCC/IM

Arquitetura de Computadores II

Pipeline

Gabriel P. Silva

Introdução

- **Pipeline é uma técnica de implementação de processadores que permite a sobreposição temporal das diversas fases de execução das instruções.**

- **Aumenta o número de instruções executadas simultaneamente e a taxa de instruções iniciadas e terminadas por unidade de tempo.**

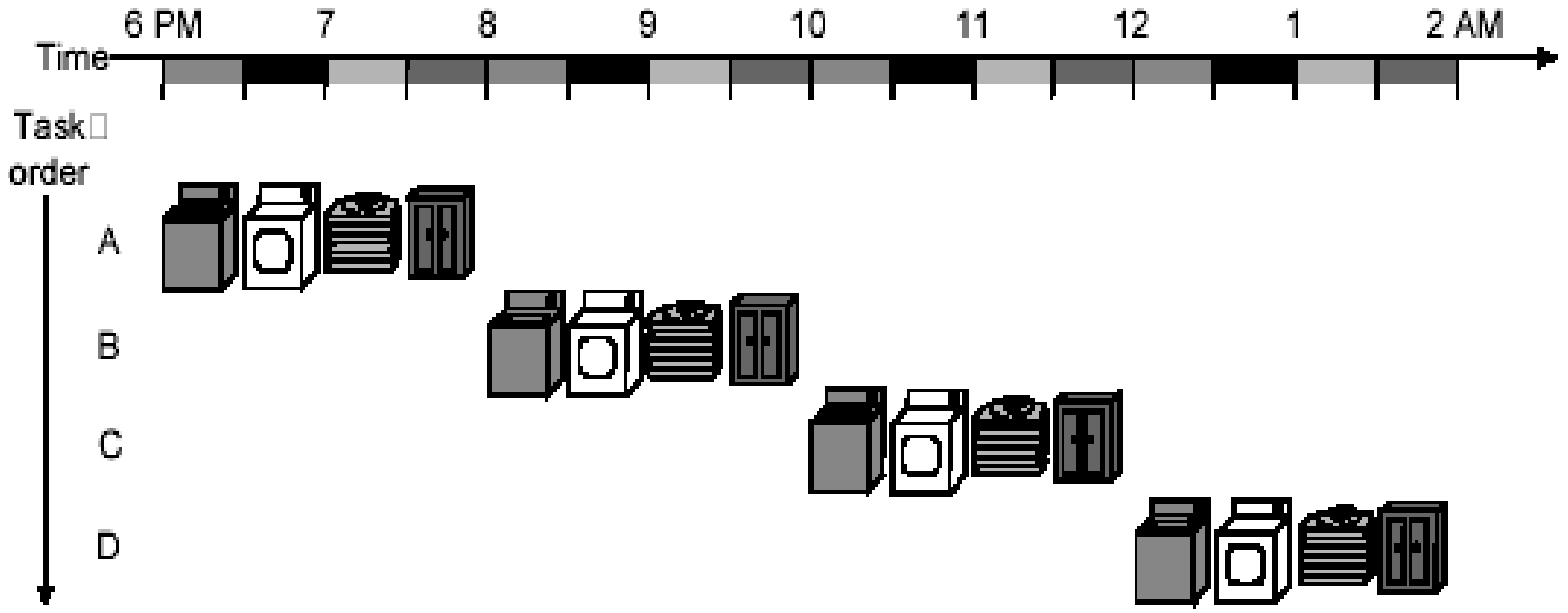
- **O pipeline não reduz o tempo gasto para completar cada instrução individualmente.**

Exemplo

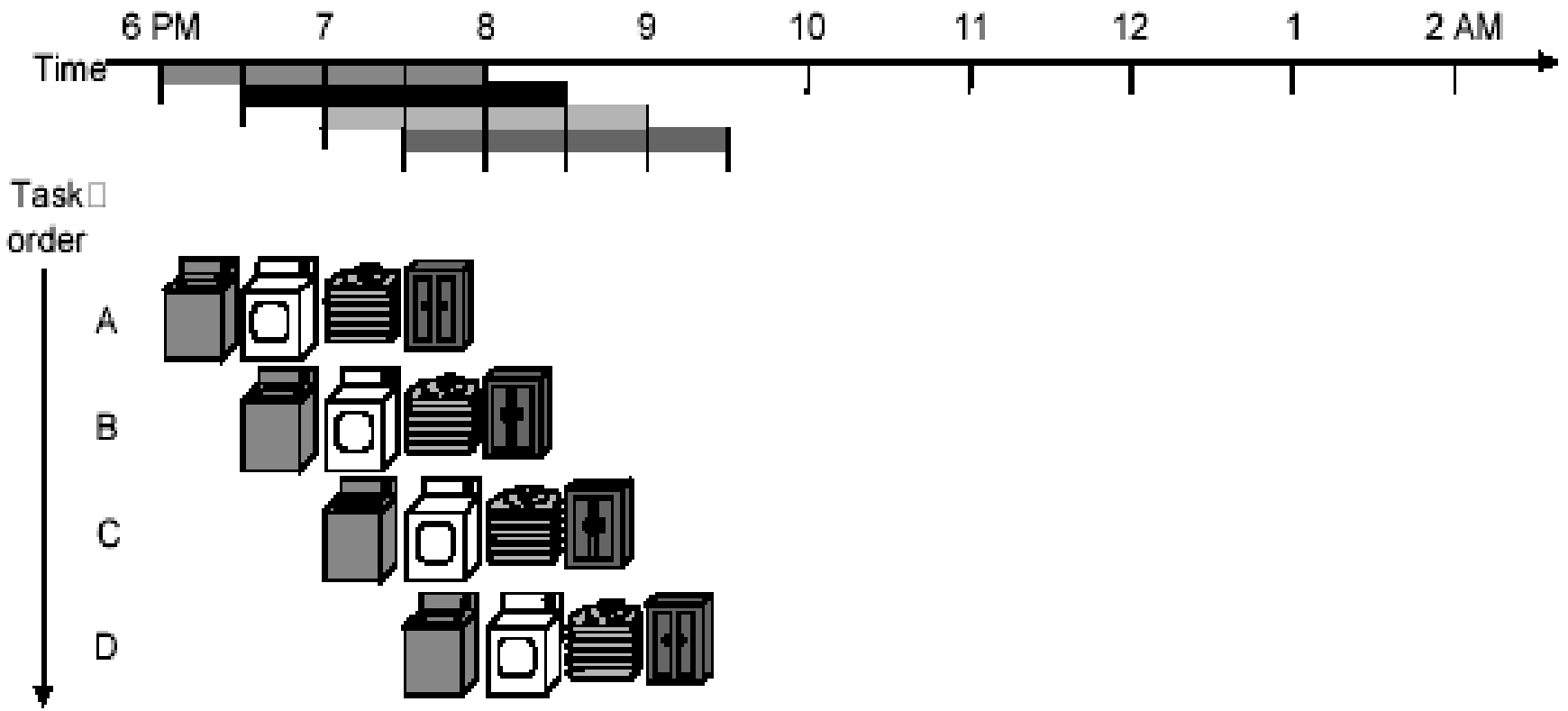
Vamos supor uma lavanderia, em que cada etapa possa ser realizada em 30 minutos:

1) **Colocar a roupa na máquina de lavar**
2) **Depois de lavada, colocá-la na máquina de secar roupa**
3) **Depois de seca, passar a ferro**
4) **Depois de passada, arrumá-la no armário**

Exemplo sem Pipeline



Exemplo com Pipeline



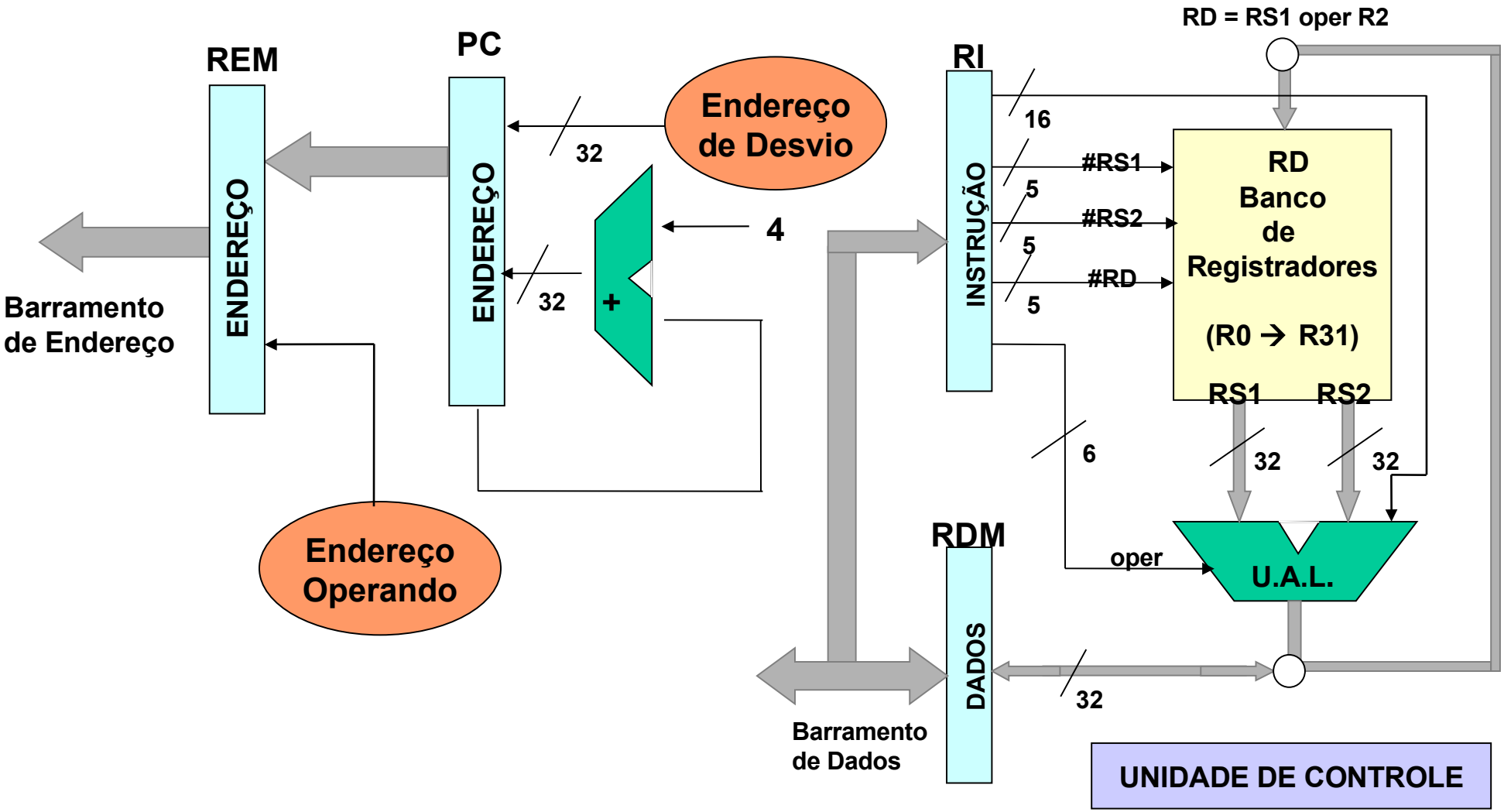
Exemplo

- **Supondo-se que cada uma destas etapas leve 30 minutos para ser realizada, a lavagem de um cesto de roupas continuará levando 2 horas para ser realizada.**
- **Entretanto, podemos iniciar a lavagem de um cesto de roupas a cada 30 minutos, até que tenhamos 4 cestos sendo lavados simultaneamente, um em cada etapa do "pipeline".**
- **Depois das primeiras 2 horas, teremos um cesto de roupa lavada a cada 30 minutos. Ao final do dia teremos lavado muito mais cestos de roupa do que sem o uso de pipeline.**

Pipeline

- Não melhora a latência de cada tarefa individualmente.
- Melhora o *throughput* de todo o trabalho.
- Várias tarefas executam simultaneamente usando recursos diferentes.
- *Speedup* potencial = número de estágios do pipeline.

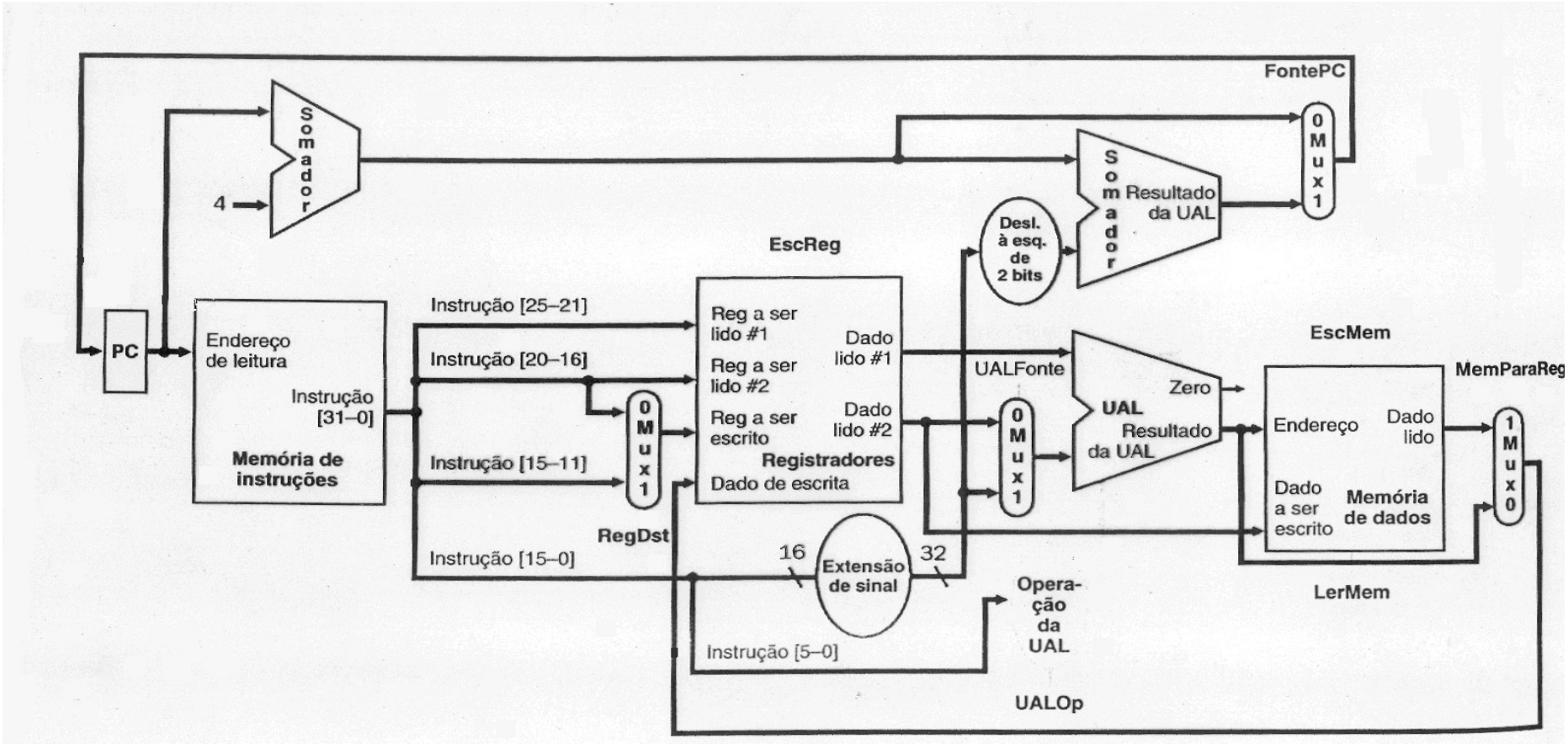
Arquitetura Básica



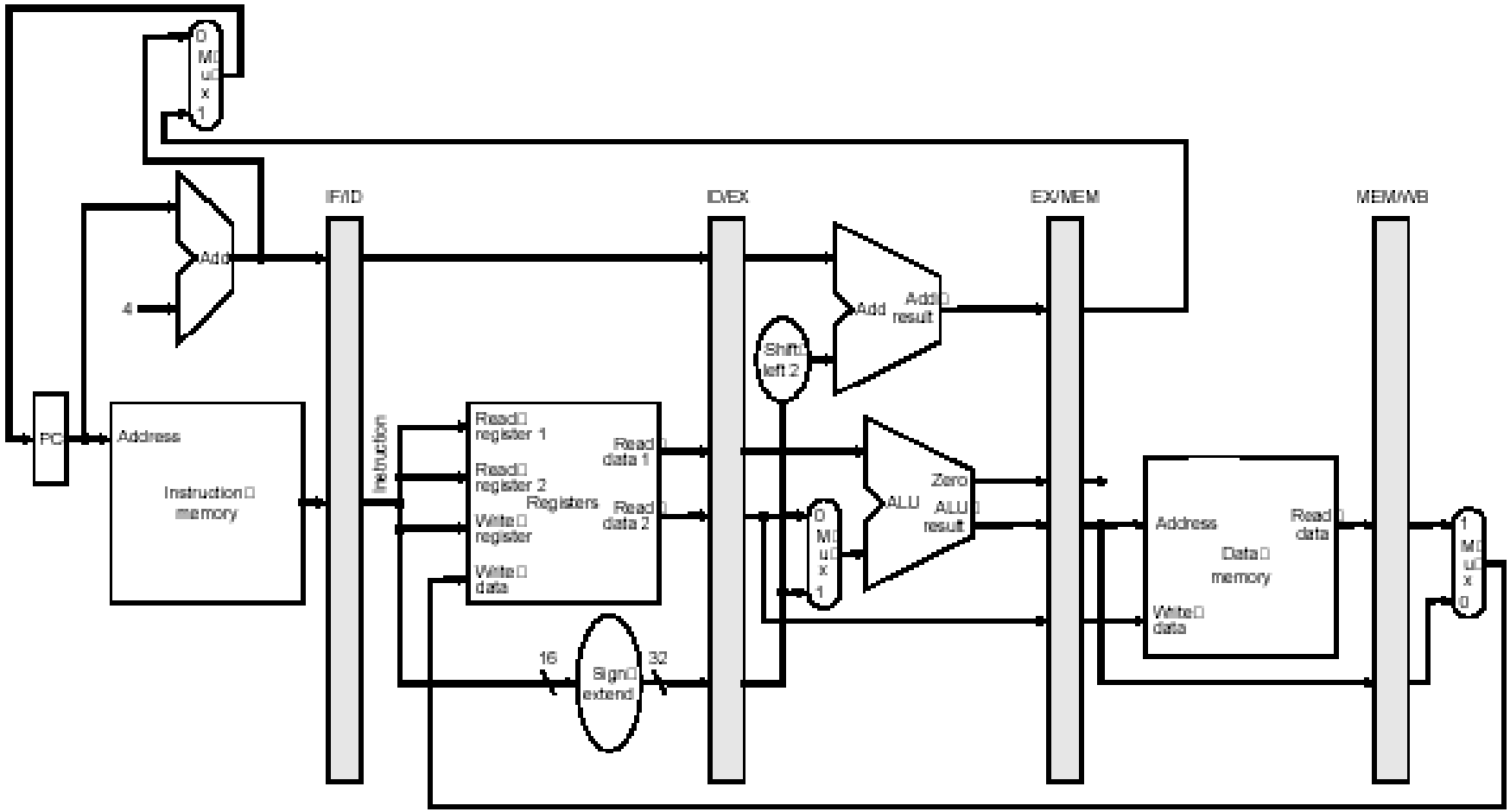
Exemplo de Pipeline de Instruções

- **Divisão da Execução da Instrução em 5 estágios:**
  - **Busca da Instrução na Memória (B)**
  - **Leitura dos Registradores e Decodificação da Instrução (D)**
  - **Execução da Instrução / Cálculo do Endereço de Desvio (E)**
  - **Acesso a um Operando na Memória (M)**
  - **Escrita de um Resultado em um Registrador (W)**

Arquitetura Sem Pipeline



Arquitetura com Pipeline



Exemplo de Instruções

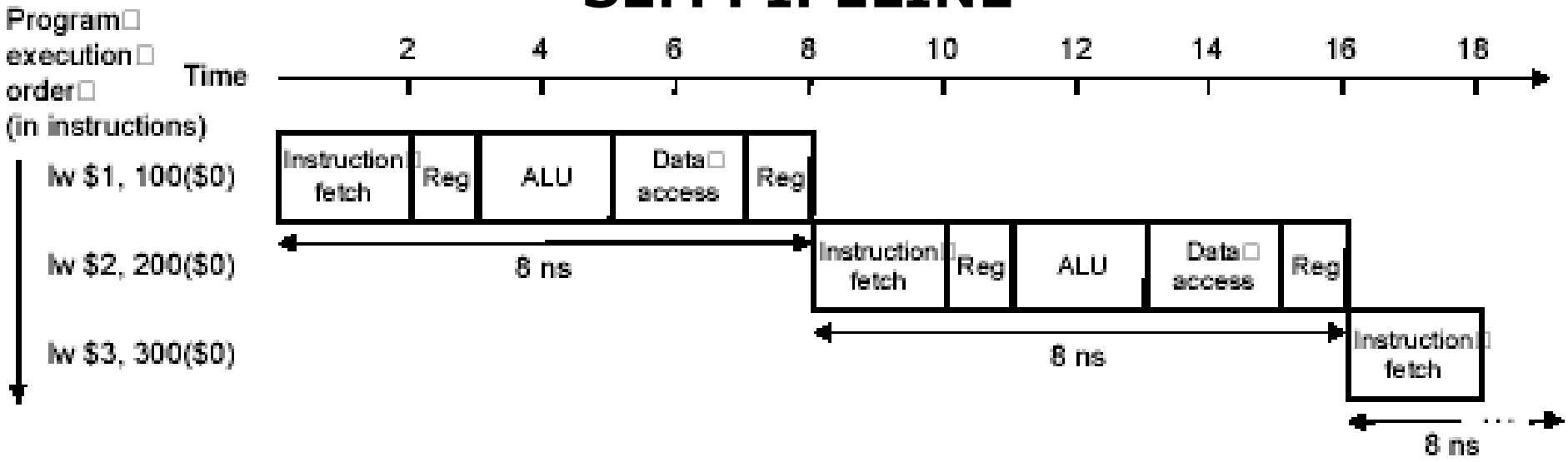
Classe da Instrução	Busca da Instrução	Leitura Operando	Operação da ULA	Acesso à Memória	Escrita do Resultado	Total
Load Word (lw)	2 ns	1 ns	2 ns	2 ns	1 ns	8 ns
Store Word (sw)	2 ns	1 ns	2 ns	2 ns	-	7 ns
Aritméti-cas (add, sub, and)	2 ns	1 ns	2 ns	-	1ns	6 ns
Branch (beq)	2 ns	1 ns	2 ns	-	-	5 ns

Exemplo de Instruções comPipeline

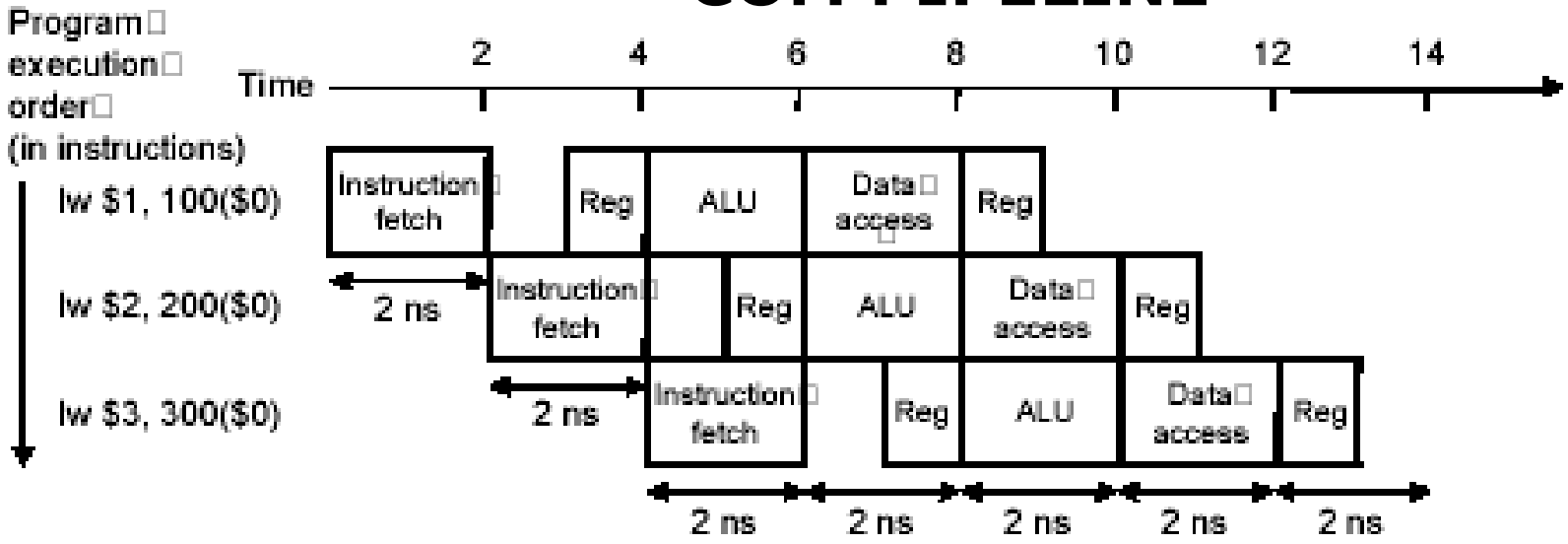
Classe da Instrução	Busca da Instrução	Leitura Operando	Operação da ULA	Acesso à Memória	Escrita do Resultado	Total
Load Word (ld)	2 ns	1 ns	2 ns	2 ns	1 ns	10 ns
Store Word (sd)	2 ns	1 ns	2 ns	2 ns		10 ns
Aritméticas (dadd, dsub, and)	2 ns	1 ns	2 ns		1ns	10 ns
Branch (beq)	2 ns	1 ns	2 ns			10 ns

O tempo do ciclo do relógio é igual a 2 ns.

SEM PIPELINE



COM PIPELINE



Características dos Pipelines de Instrução

- **O tempo do ciclo do relógio do processador deve ser igual ou maior que o tempo de execução do estágio mais lento do "pipeline".**
- **Deve-se procurar dividir a execução da instrução em estágios com o mesmo tempo.**
- **O pipeline deve ser mantido sempre "cheio" para que o desempenho máximo seja alcançado.**
- **De um modo geral, com o uso do pipeline, cada instrução ainda leva o mesmo tempo para ser executada.**
- **Algumas instruções contudo podem ter o seu tempo de execução aumentado, pois atravessam estágios em que não realizam nenhuma operação útil.**

Características dos Pipelines de Instrução

- **O tempo gasto no processamento de M instruções em um pipeline com K estágios e ciclo de máquina igual a t é dado por:**

$$T = [K + (M - 1)] * t$$

- **Se M >> K (caso comum), T é aproximadamente M * t**

Características dos Pipelines de Instrução

- **Se um programa tem 10.001 instruções. Quanto tempo leva para ser executado em um processador com pipeline de 5 estágios e relógio de 100 ns?**

T= (5 + (10.000))*100x10^{-9} =~ 1 ms
(com pipeline)

T = 500 ns * 10.000 =~ 5ms
(sem pipeline)

Problemas no Uso de Pipelines

- **Estágios podem ter tempos de execução diferentes:**
  - **Solução 1: Implementar esses estágios como um pipeline onde cada sub-estágio possui tempo de execução semelhante aos demais estágios do pipeline principal.**
  - **Solução 2: Replicar esse estágio, colocando réplicas em paralelo no estágio principal. O número de réplicas é dado pela razão entre o tempo do estágio mais lento e os demais.**
- **O sistema de memória é incapaz de manter o fluxo de instruções no pipeline**
  - **O uso de memória cache com alta taxa de acerto e tempo de acesso compatível com o tempo de ciclo do pipeline.**

Problemas no Uso de Pipelines

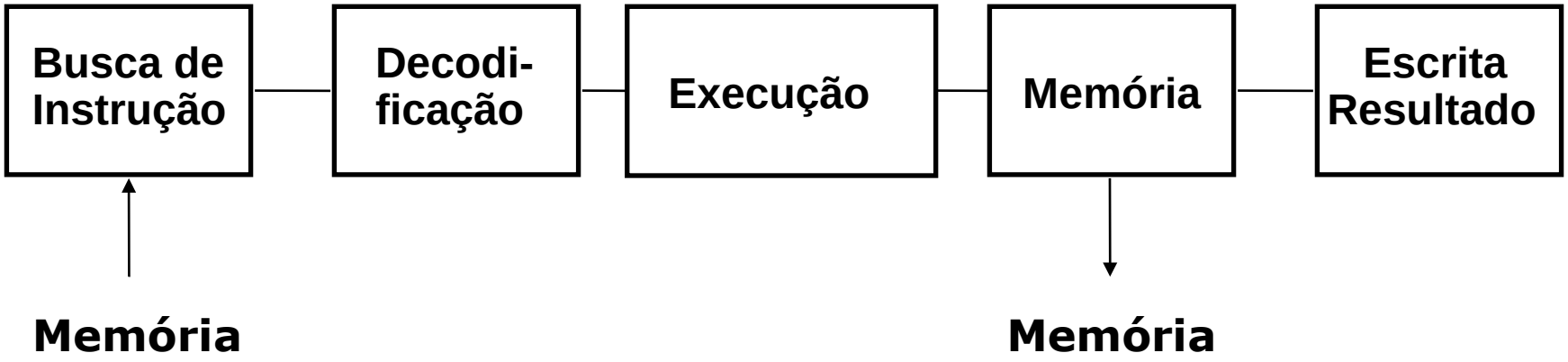
- **Dependências ou Conflitos ("Hazards")**
  - **Conflitos Estruturais**
    - **Pode haver acessos simultâneos à memória feitos por 2 ou mais estágios.**
  - **Dependências de Dados**
    - **As instruções dependem de resultados de instruções anteriores, ainda não completadas.**
  - **Dependências de Controle**
    - **A próxima instrução não está no endereço subseqüente ao da instrução anterior.**
- **Tratamento de Exceções**

Conflitos Estruturais

- **Acessos Concorrentes à Memória**
  - **Uso de memórias multi-portas ou com múltiplos bancos com acessos independentes.**
- **Leitura de instrução e leitura/escrita de dados simultâneos à memória.**
  - **Uso de arquitetura "Harvard" com caches de dados e instrução separados.**
- **Acesso simultâneo ao banco de registradores**
  - **Uso de banco de registradores com múltiplas portas.**
- **Uso simultâneo de uma mesma unidade funcional**
  - **Replicação da unidade funcional ou implementação "pipelined" dessa unidade.**

Conflitos Estruturais

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



- **Soluções**
  - **Caches separadas de dados e instruções.**
  - **Memória com múltiplos bancos com acessos independentes.**

Conflitos por Dados

- **Problema: uma instrução faz uso de um operando que vai ser produzido por uma outra instrução que ainda está no pipeline.**
- **A execução da instrução seguinte depende de operando calculado pela instrução anterior.**
- **Tipos de dependências de dados:**
  - **Dependência verdadeiras**
  - **Dependências falsas**
    - **antidependência**
    - **dependência de saída**

Conflitos por Dados

- **Tipos de dependências de dados:**
  - **Dependências verdadeiras (diretas) ou RAW:**
    - **Uma instrução utiliza um operando que é produzido por uma instrução anterior.**
  - **Dependências falsas:**
    - **Antidependência ou WAR:**
      - **Uma instrução lê um operando que é escrito por uma instrução sucessora.**
    - **dependência de saída ou WAW:**
      - **Uma instrução escreve em um operando que é também escrito por uma instrução sucessora.**

Conflitos por Dados

- **Dependência direta:**

```
dadd R1, R2, R3
dsub R4, R1, R6
```

- **Antidependência:**

```
dsub R4, R1, R6
dadd R1, R2, R3
```

- **Dependência de Saída:**

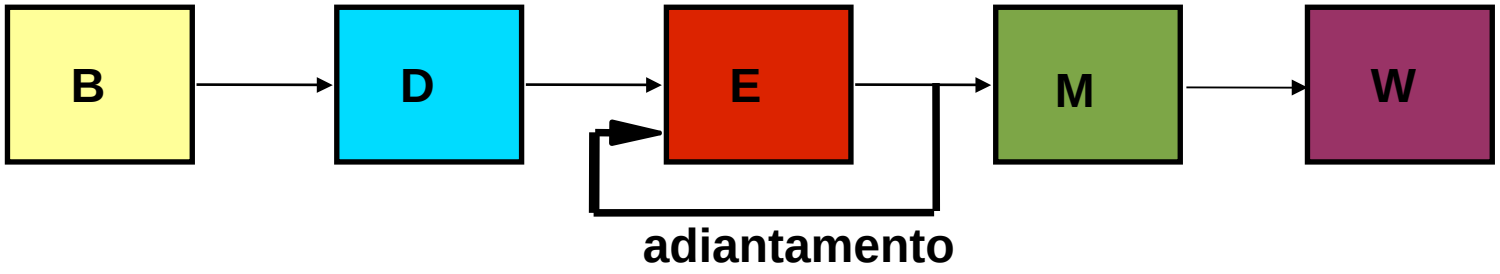
```
dadd R4, R2, R3
dsub R4, R1, R6
```

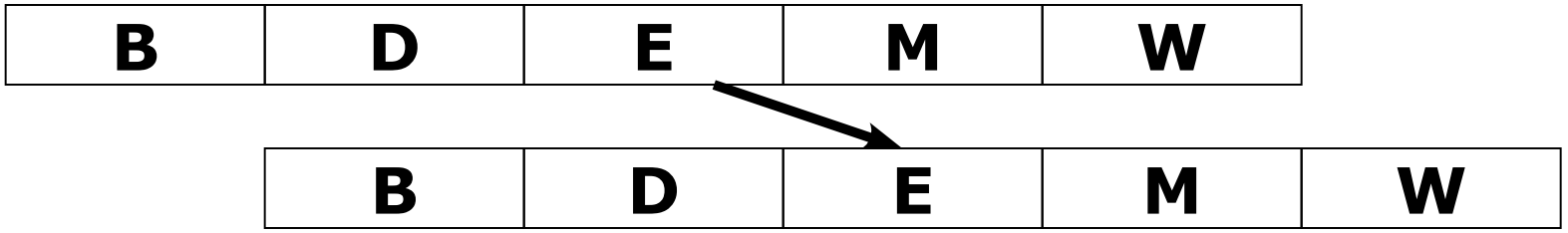
Dependências de Dados

- **Dependências Verdadeiras (Direta):**
  - **O pipeline precisa ser parado durante certo número de ciclos (interlock);**
  - **Colocação de instruções de "nop" ou escalonamento adequado das instruções pelo compilador;**
  - **O adiantamento dos dados pode resolver em alguns casos.**
- **Dependências Falsas:**
  - **Não é um problema em pipelines onde a ordem de execução das instruções é mantida;**
  - **Problema em processadores superescalares;**
  - **A renomeação dos registradores é uma solução usual para este problema.**

Adiantamento dos Dados

- **Caminho interno dentro do pipeline entre a saída da ALU e a entrada da ALU**
  - **Evita a *parada* do pipeline**





Escalonamento das Instruções

- **Exemplo:**

  X = Y - W

  Z = K + L

- **Código Gerado:**

```
ld      R1, mem[Y]
ld      R2, mem[W]
sub     R3, R1, R2      ➔ Situação de Interlock
sd      R3, mem[X]      ➔ Situação de Interlock
ld      R4, mem[K]
ld      R5, mem[L]
add     R6, R4, R5      ➔ Situação de Interlock
sd      R6, mem[Z]      ➔ Situação de Interlock
```

Escalonamento das Instruções

- **Código executado com nops inseridos pelo Interlock:**

```
ld      R1, mem[Y]
ld      R2, mem[W]
nop
nop
sub     R3, R1, R2 ➔ Situação de Interlock
nop
nop
sd      R3, mem[X] ➔ Situação de Interlock
ld      R4, mem[K]
ld      R5, mem[L]
nop
nop
add     R6, R4, R5     ➔ Situação de Interlock
nop
nop
sd      R6, mem[Z]     ➔ Situação de Interlock
```

Escalonamento das Instruções

- **Código Otimizado:**

```
ld      R1, mem[Y]
ld      R2, mem[W]
ld      R4, mem[K]
sub     R3, R1, R2
ld      R5, mem[L]
sd      R3, mem[X]
add     R6, R4, R5
sd      R6, mem[Z] ➔ Adiantamento
```

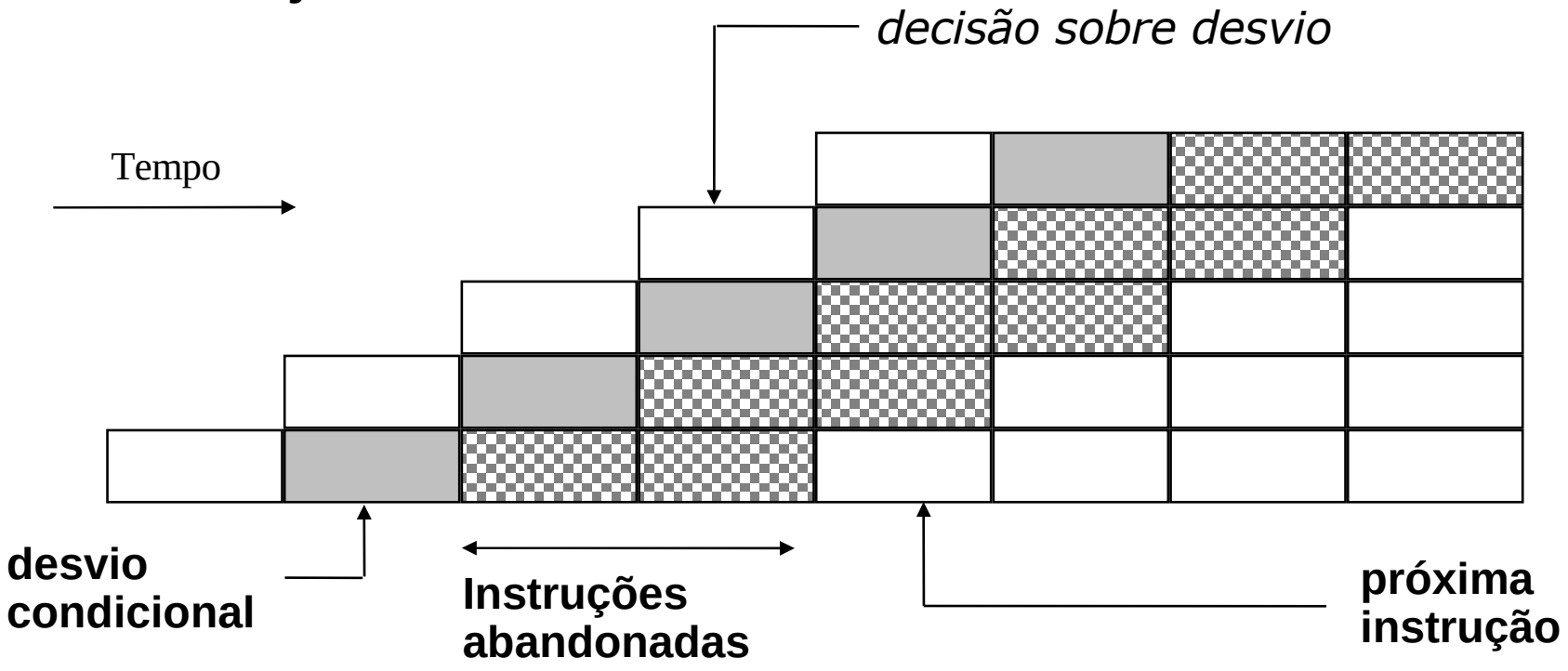
Escalonamento das Instruções

- **Código executado com nops inseridos pelo Interlock:**

```
ld      R1, mem[Y]
ld      R2, mem[W]
ld      R4, mem[K]
nop
sub     R3, R1, R2    → Situação de Interlock
ld      R5, mem[L]
sd      R3, mem[X]
nop
add     R6, R4, R5 → Situação de Interlock
nop
nop
sd      R6, mem[Z] → Situação de Interlock
```

Conflitos de Controle

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



Soluções para os Conflitos de Controle

- **Uso do Desvio Atrasado**
  - **A instrução após o desvio é sempre executada → preenchimento útil do "delay slot" nem sempre é possível**
- **Congelar o pipeline até que o resultado do desvio seja conhecido**
  - **Insere "bolhas" no pipeline → solução ruim quando o pipeline é muito longo**

Soluções para os Conflitos de Controle

- **Predição Estática de Desvios**
  - **O compilador faz uma predição se o desvio vai ser tomado ou não ➔ geração de "bolhas" quando a predição é errada, baixa taxa de acertos**
- **Predição Dinâmica de Desvios**
  - **Existem mecanismos em "hardware" que fazem a predição baseada no comportamento daquele desvio no passado ➔ idem, alta taxa de acertos**

Preenchimento do "delay slot"

- **Exemplo 1:**

dadd R1, R2, R3 beq R2, R0, label *delay slot* 3 ciclos	beq R2, R0, label **dadd R1, R2, R3** 2 ciclos

- **Exemplo 2:**

dsub R4, R5, R6 dadd R1, R2, R3 beq R1, R0, label *delay slot* 4 ciclos	dadd R1, R2, R3 beq R1, R0, label **dsub R4, R5, R6** 3 ciclos

Preenchimento do “delay slot”

- **Para facilitar o trabalho do compilador no preenchimento do “delay slot” muitas arquiteturas permitem o uso do “delay slot” com a opção de anulação automática dessa instrução se o desvio condicional não for tomado.**
- **Desse modo, uma instrução do endereço alvo pode ser movida para o “delay slot”, o que é muito útil no caso de “loops”. Nesse caso, está implícita uma previsão de desvio estática que diz que o desvio será sempre tomado.**

Predição Estática do Desvio

- **Três abordagens podem ser adotadas:**
  - **Assumir que todos os desvios são tomados (“branch taken”)**
  - **Os desvios para trás são assumidos como tomados (“branch taken”) e os desvios para frente são assumidos como não tomados (“branch not taken”)**
  - **Fazer a predição com base em resultados coletados de experiências de “profile” realizadas anteriormente**

Predição Dinâmica do Desvio

- **Pequena memória endereçada pela parte baixa do endereço das instruções de desvio;**
- **A memória contém 1 bit (bit de predição) que diz se o desvio foi tomado ou não da última vez;**
- **Se a predição for errada, o bit correspondente é invertido na memória**
- **Problemas:**
  - **Instruções de desvio diferentes podem mapear para uma mesma posição do buffer**
  - **O esquema pode falhar quando a decisão do desvio se alterna a cada execução**

Tratamento de Exceções

- **Exemplos de Exceções:**
  1. **Interrupção de dispositivos de E/S**
  2. **Chamadas ao Sistema Operacional**
  3. **Breakpoints**
  - **Operações Aritméticas (Overflow e Underflow)**
  - **Falha de página**
  - **Erros de endereçamento de memória**
  - **Violação de proteção de memória**
  - **Instrução inválida**
  - **Falha de alimentação**

Classificação das Exceções

Síncronas	2, 3, 4, 5, 6, 7, 8
Assíncronas	1, 9

Solicitadas pelo usuário	2, 3
Fora do controle do usuário	1, 4, 5, 6, 7, 8, 9

No meio da instrução	4, 5, 6, 7, 8, 9
Entre instruções	1, 2, 3

Encerram a execução do programa	6, 7, 8, 9
Permitem a continuação do programa	1, 2, 3, 4, 5

Modelo de Exceções Precisas

- **Definição:**
  - **Ao ser detectada uma exceção, todas as instruções anteriores à ocorrência da exceção podem ser completadas e as posteriores podem ser anuladas e reiniciadas após o tratamento da exceção**
- **Requisição Básico:**
  - **Instruções só mudam o estado da máquina quando há garantia de que elas concluirão sem exceões**
- **Conseqüências:**
  - **Difícil de implementar, sem perda de desempenho, em pipelines que implementam instruções complexas**
  - **Algumas arquiteturas possuem dois modos de operação: com ou sem exceções precisas**