Arquitectura de Computadores

Y86
Encadeamento
(parte I)

Créditos

Randal E. Bryant

http://csapp.cs.cmu.edu

Vista Geral

Princípios Gerais do Encadeamento

- Ambições
- Dificuldades

Um Processador Y86 com Encadeamento

- Rearranjo do SEQ
- Inserção de registos de encadeamento
- Problemas com anomalias de dados e de controlo

Mundo Real: Lavagem de Automóveis

Sequencial

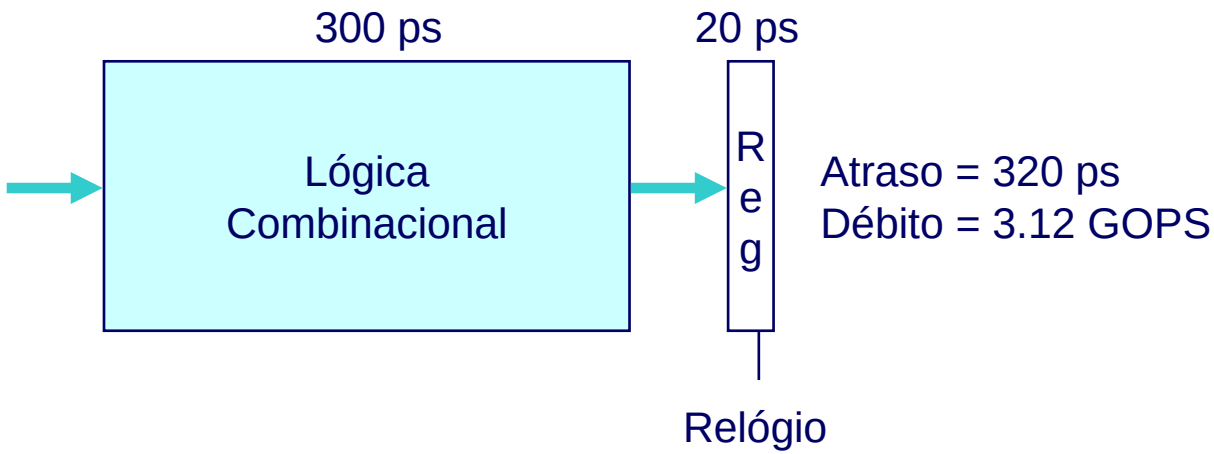
Paralelo

Encadeamento

Ideia

- Dividir o trabalho em estágios independentes
- Mover os objectos através de uma sequência de estágios
- Num dado momento múltiplos objectos a serem processados

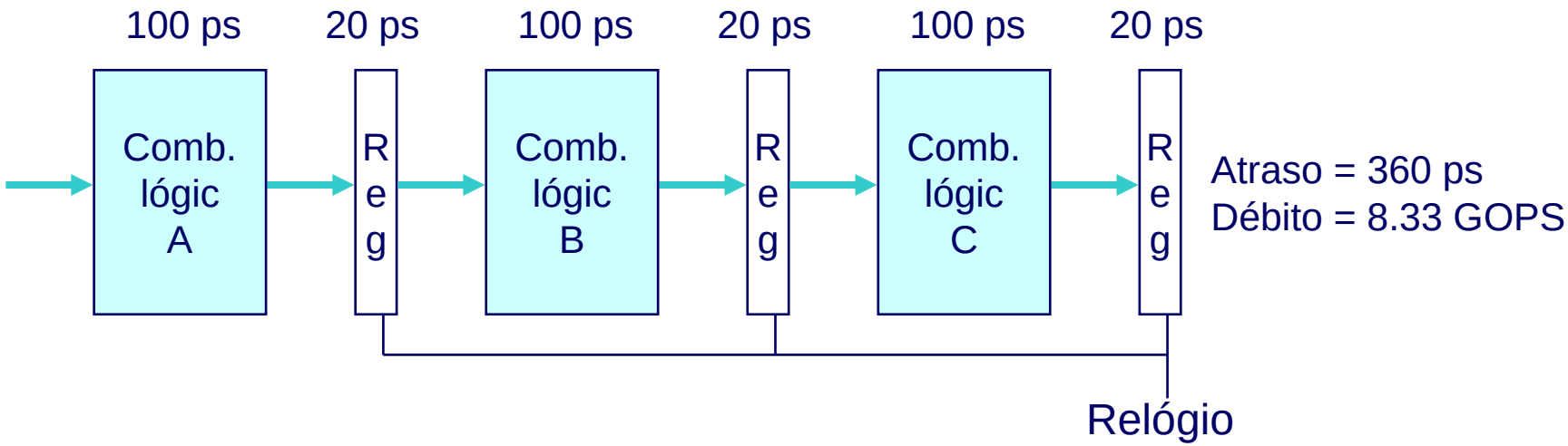
Exemplo Computacional



Sistema

- **Computação requer um total de 300 pico segundos**
- **20 pico segundos para guardar o resultado num registo**
- **O ciclo de relógio**
- **tem de ser pelo menos 320 ps**
- **3,12 GOPS =1 operação/ (20+300) ps * 1000ps / 1 ns**

Encadeamento Versão 3-Vias

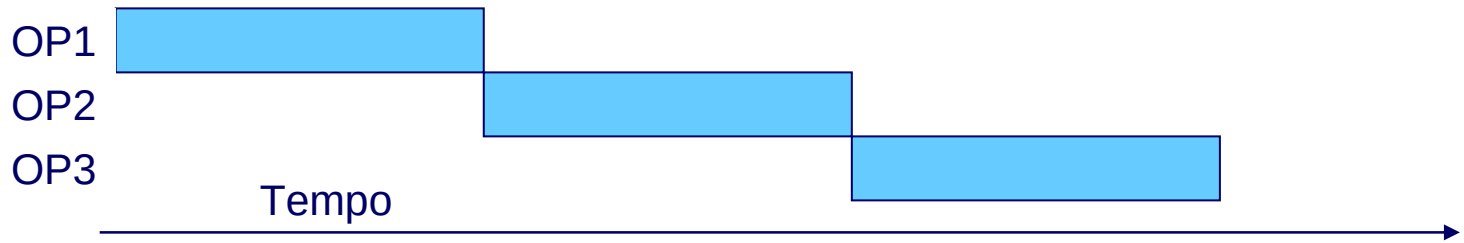


Sistema

- **Lógica Combinacional dividida em 3 blocos de 100 ps cada**
- **Uma nova operação arranca sempre que a anterior atravessa o estágio A.**
  - **Cada 120 ps**
- **Atraso cresce**
  - **360 ps desde o início ao fim**
- **Comparação**
  - **Tempo cresce 1,12 = 360/320; Débito cresce 2,67 = 8,33/3,12**

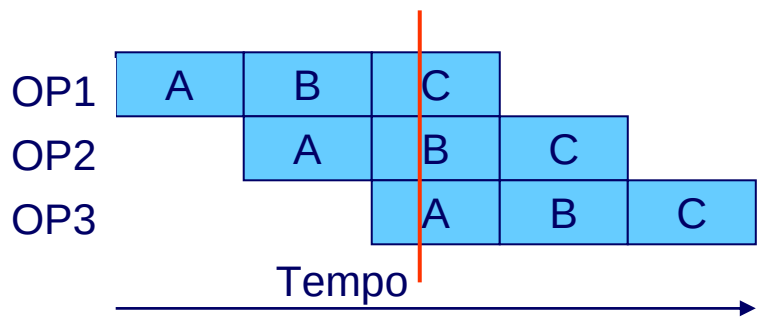
Encadeamento - Diagramas

Sem Encadeamento



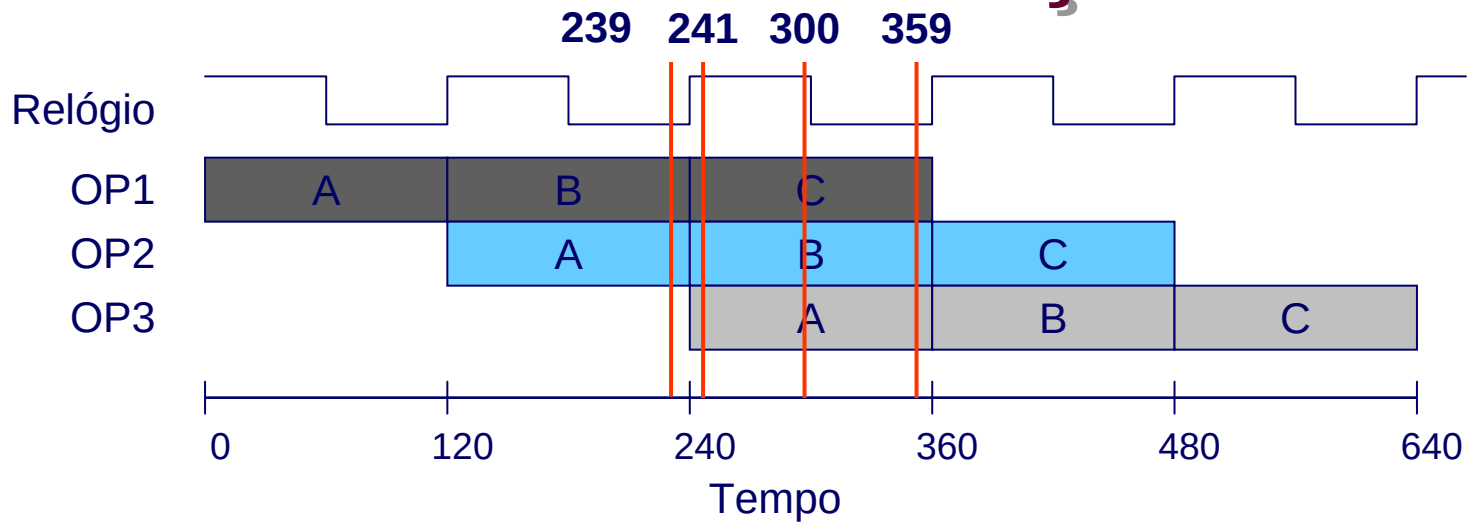
- **Nova operação só quando se completa a anterior**

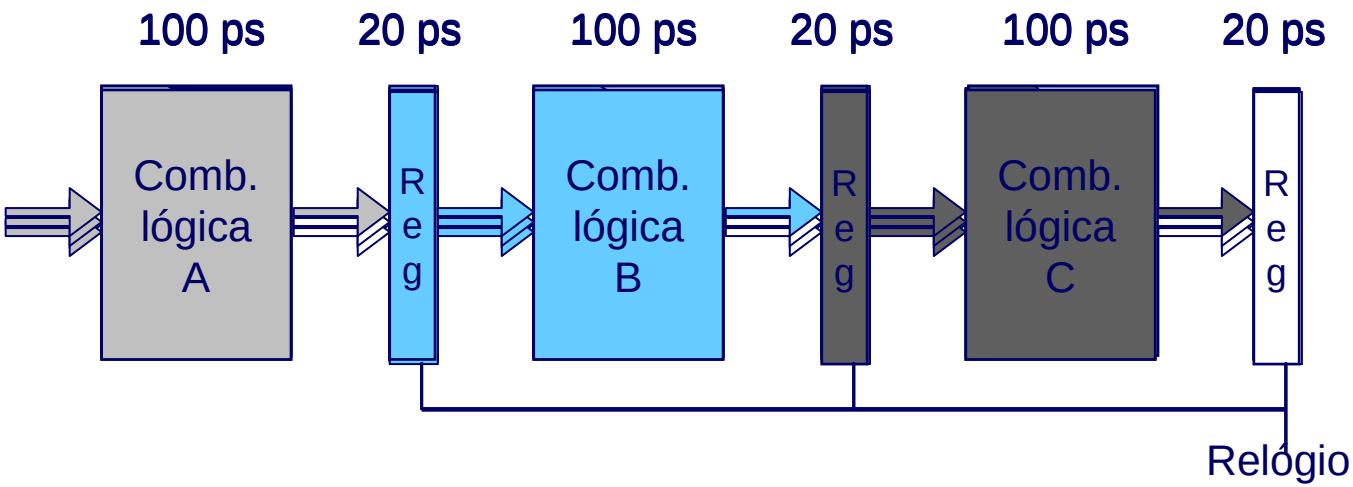
Encadeamento 3-Vias



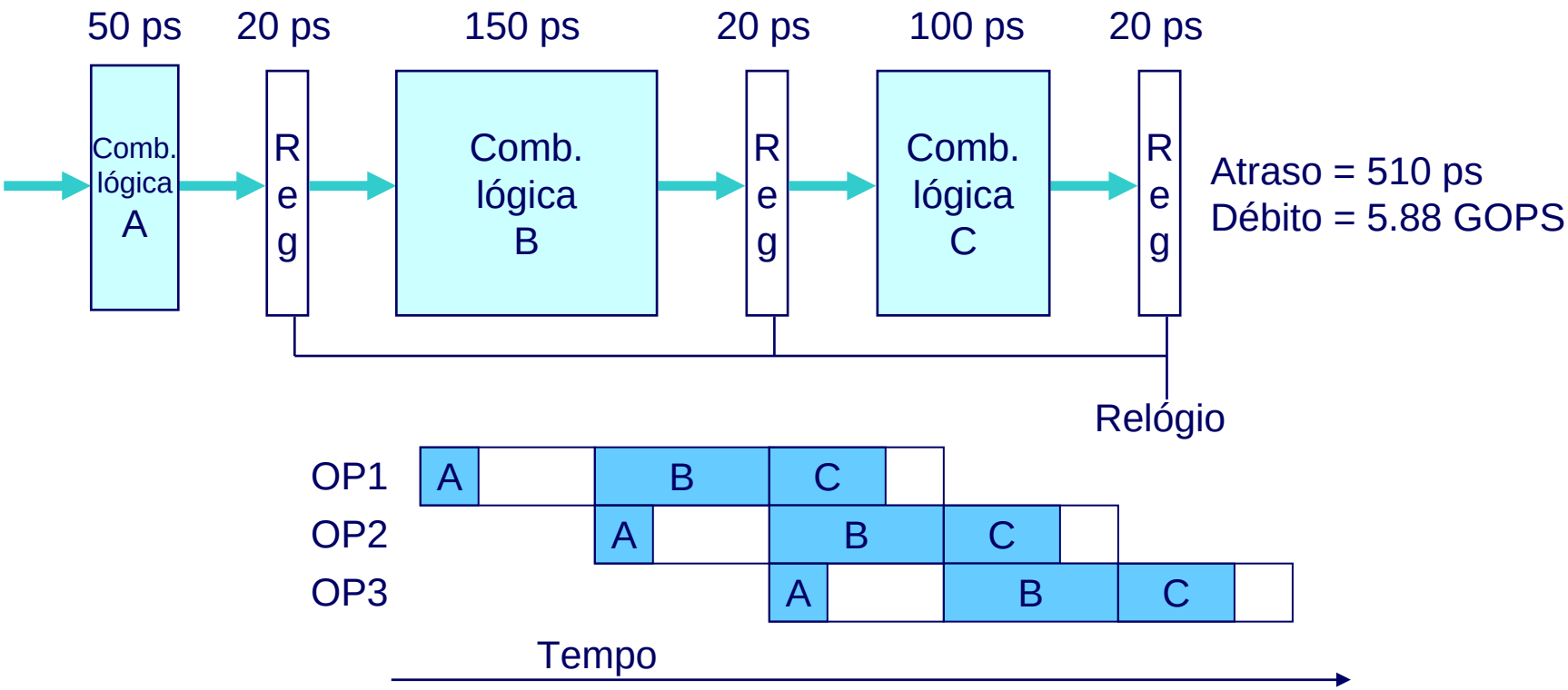
- **Até 3 operações em processamento simultâneo**

Encadeamento - Operação



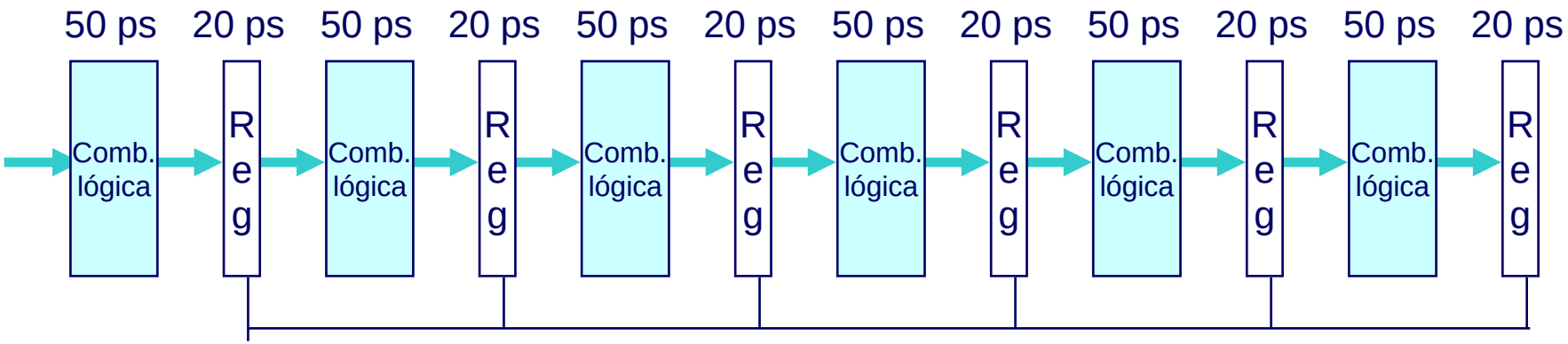


Limitações: Tarefas não-uniformes



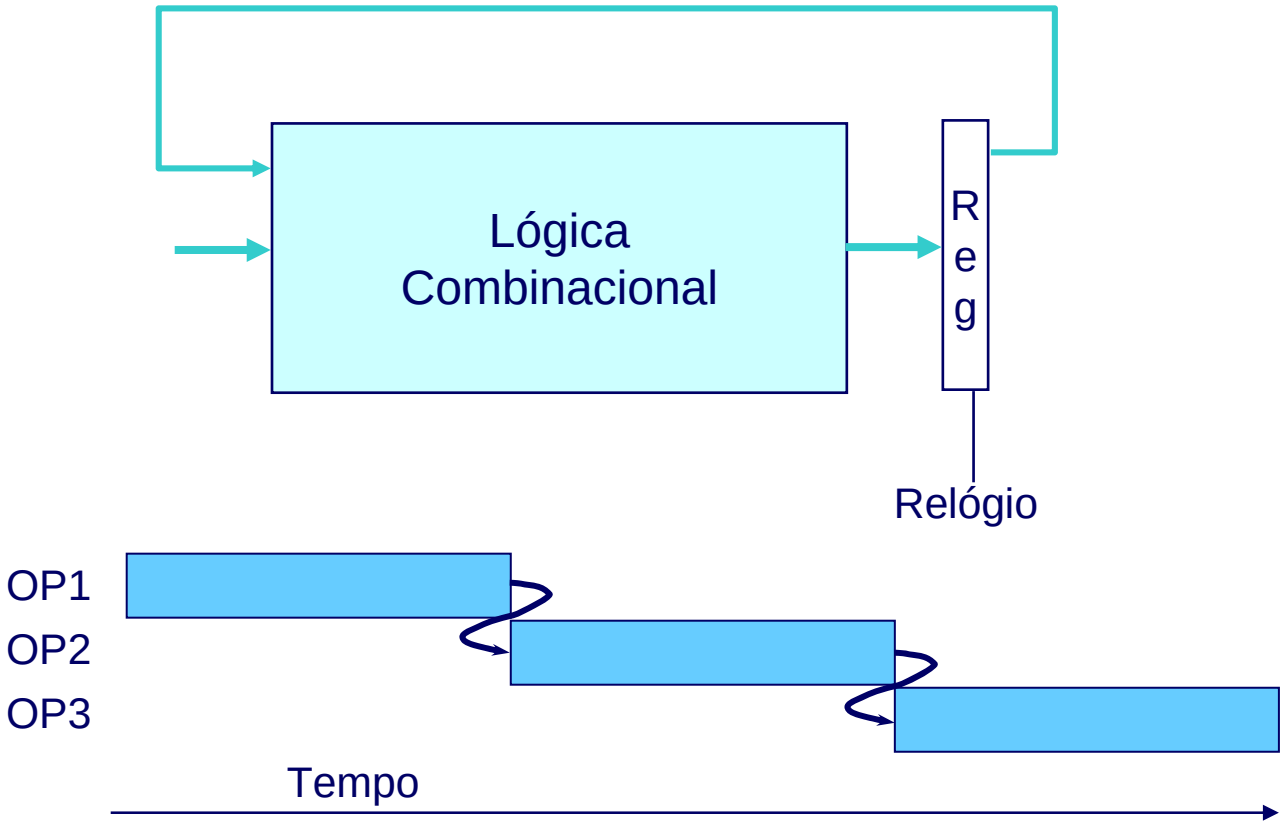
- **Débito limitado pelo estágio mais longo**
  - **Ciclo relógio 170 ps = 150ps + 20ps; atraso 510ps**
- **Há estágios desocupados grande parte do tempo**
  - **Estágio A - ocioso 100ps em cada ciclo**
- **O desafio é dividir o sistema em estágios balanceados**

Limitações: Sobrecarga de Registos



- **Quanto mais profundo o encadeamento mais significativo o peso da carga dos registos**
  - **Ciclo relógio 70 ps >= 50ps + 20ps;**
- **Percentagem do ciclo de relógio gasto a carregar registos :**
  - **1-estágio : 6.25%**
  - **3-estágios : 16.67%**
  - **6-estágios : 28.57%**
- **Processadores modernos projectados com encadeamento muito profundo (>15 estágios)**

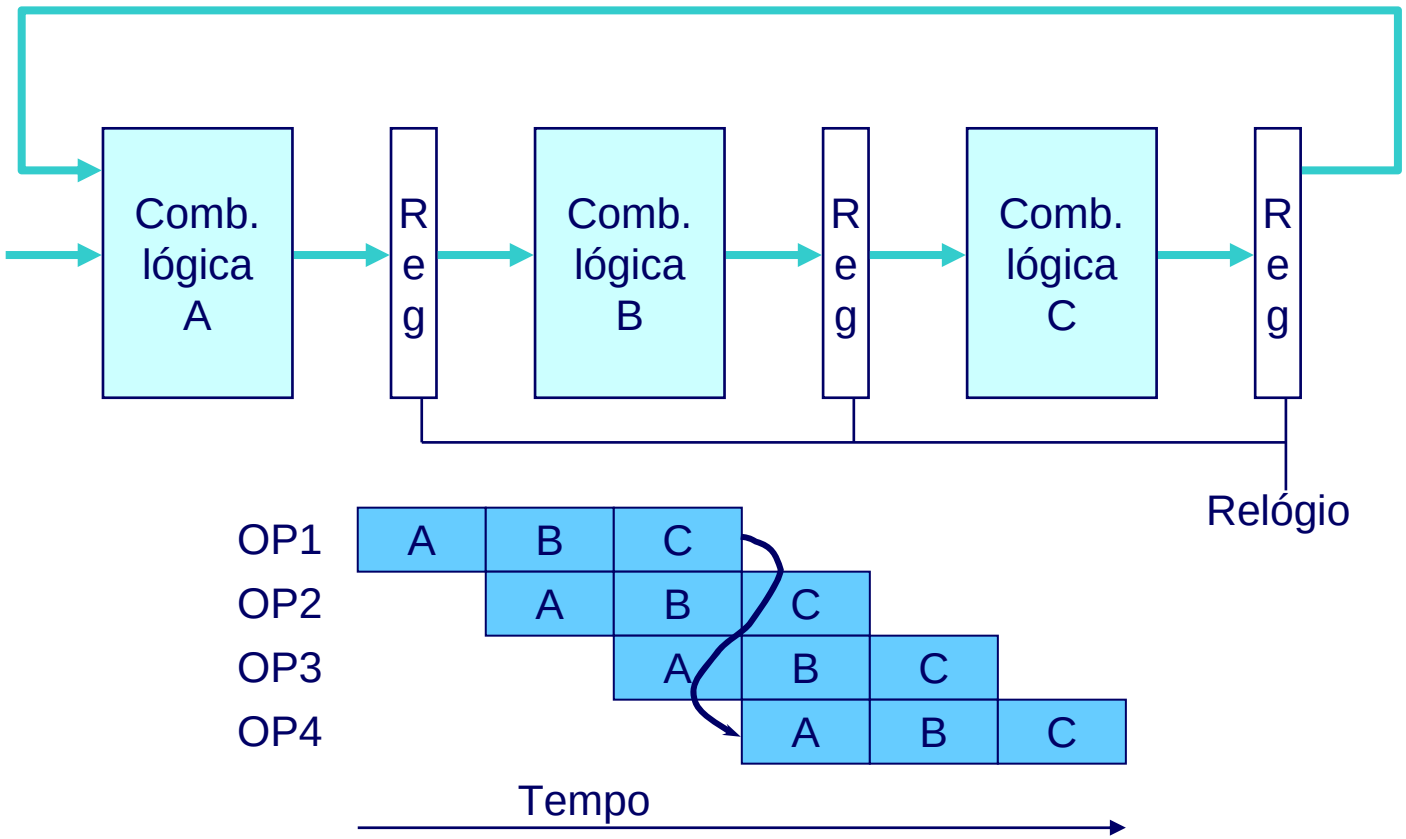
Dependências de Dados



Sistema

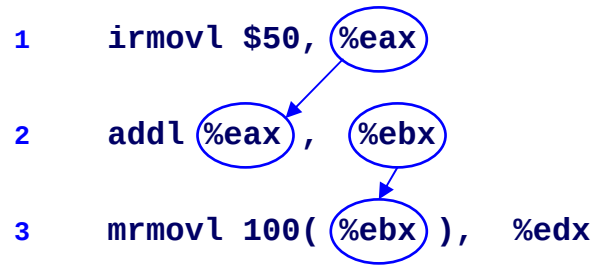
- **Cada operação depende da operação precedente**

Anomalias de Dados



- **Realimentação da próxima operação não é conveniente**
- **Encadeamento modificou o comportamento do sistema**

Dependências de dados



- Resultado de uma instrução usado como operando de outra
  - Dependência de leitura-após-escrita (RAW)
- Muito habitual em programas
- Necessário garantir que o encadeamento opera de forma conveniente
  - Obter resultados correctos
  - Minimizar o impacto no desempenho

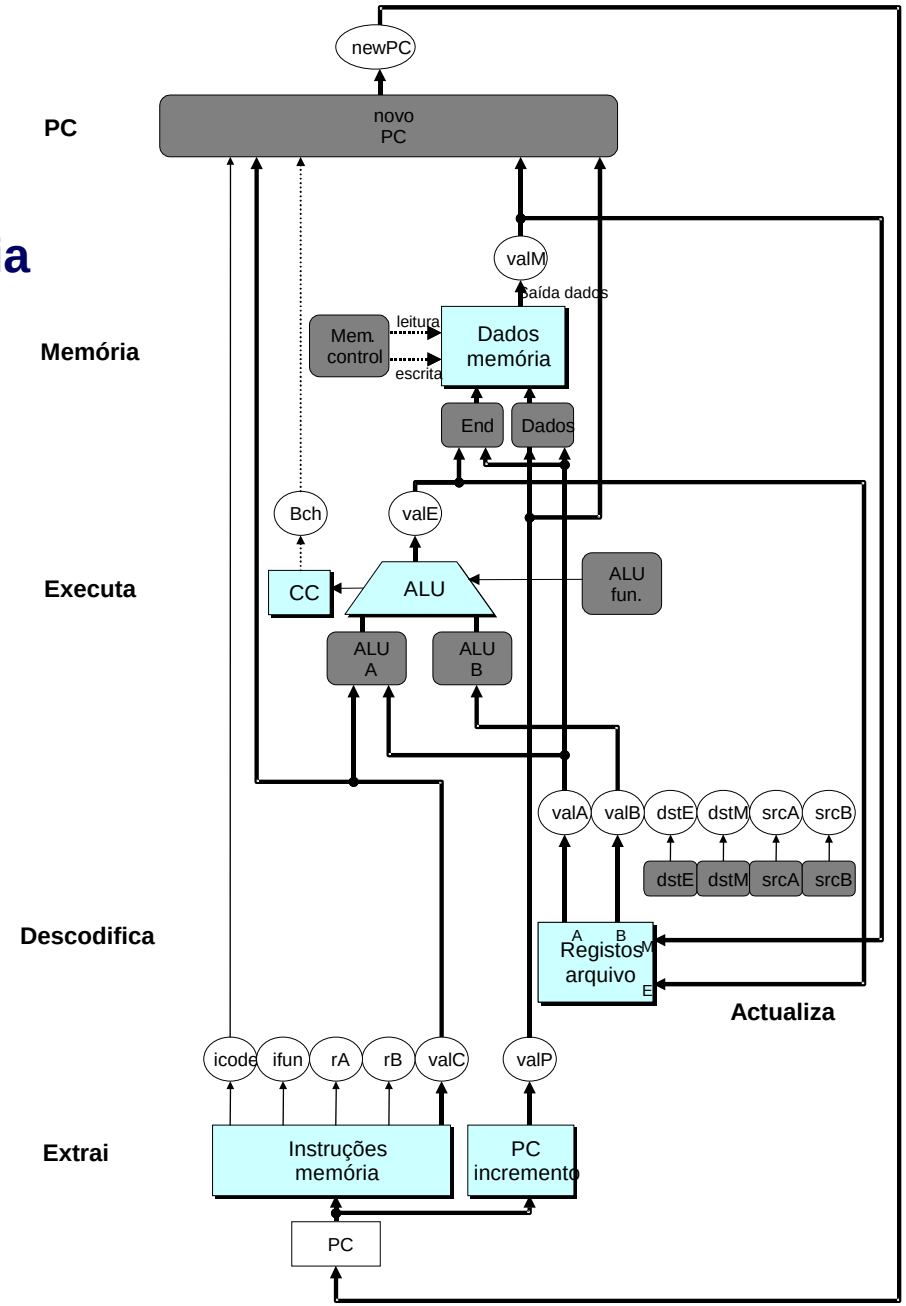
Dependências de controlo

```
1:laço  sub %edx,%ebx
2:      jne  t
3:      irmovl $10, %edx
4:      jmp  laço
5: t    halt
```

- A instrução na linha 2 cria uma dependência de controlo
  - Próxima instrução linha 3 ou linha 5.
- Necessário garantir que o encadeamento opera de forma conveniente para que
  - O valor de PC seja correcto
  - Minimizar o impacto no desempenho

SEQ Hardware

- Estágios ocorrem em sequência
- Uma operação processada em cada ciclo



SEQ+ Hardware

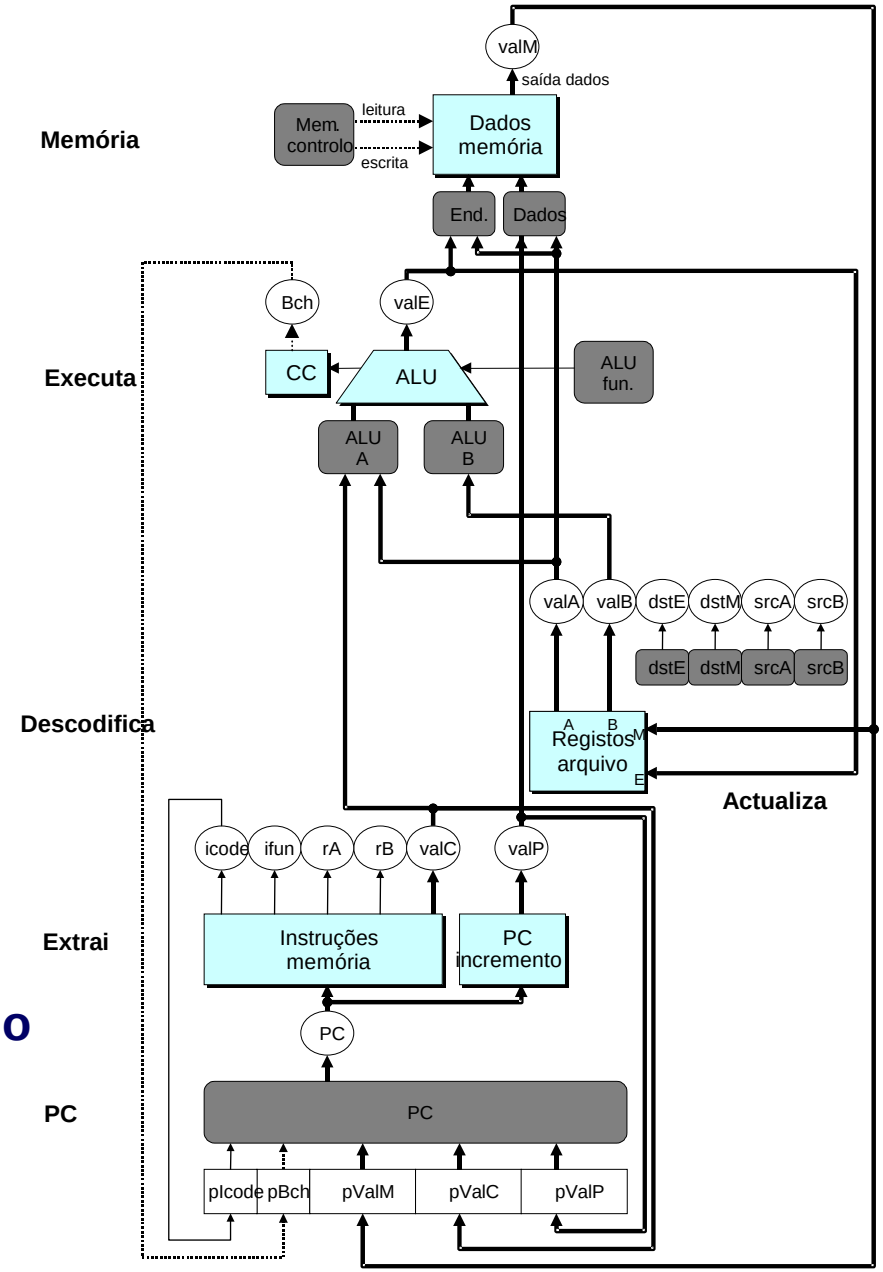
- Realização sequencial
- Estágio PC mudado para o início

Estágio PC

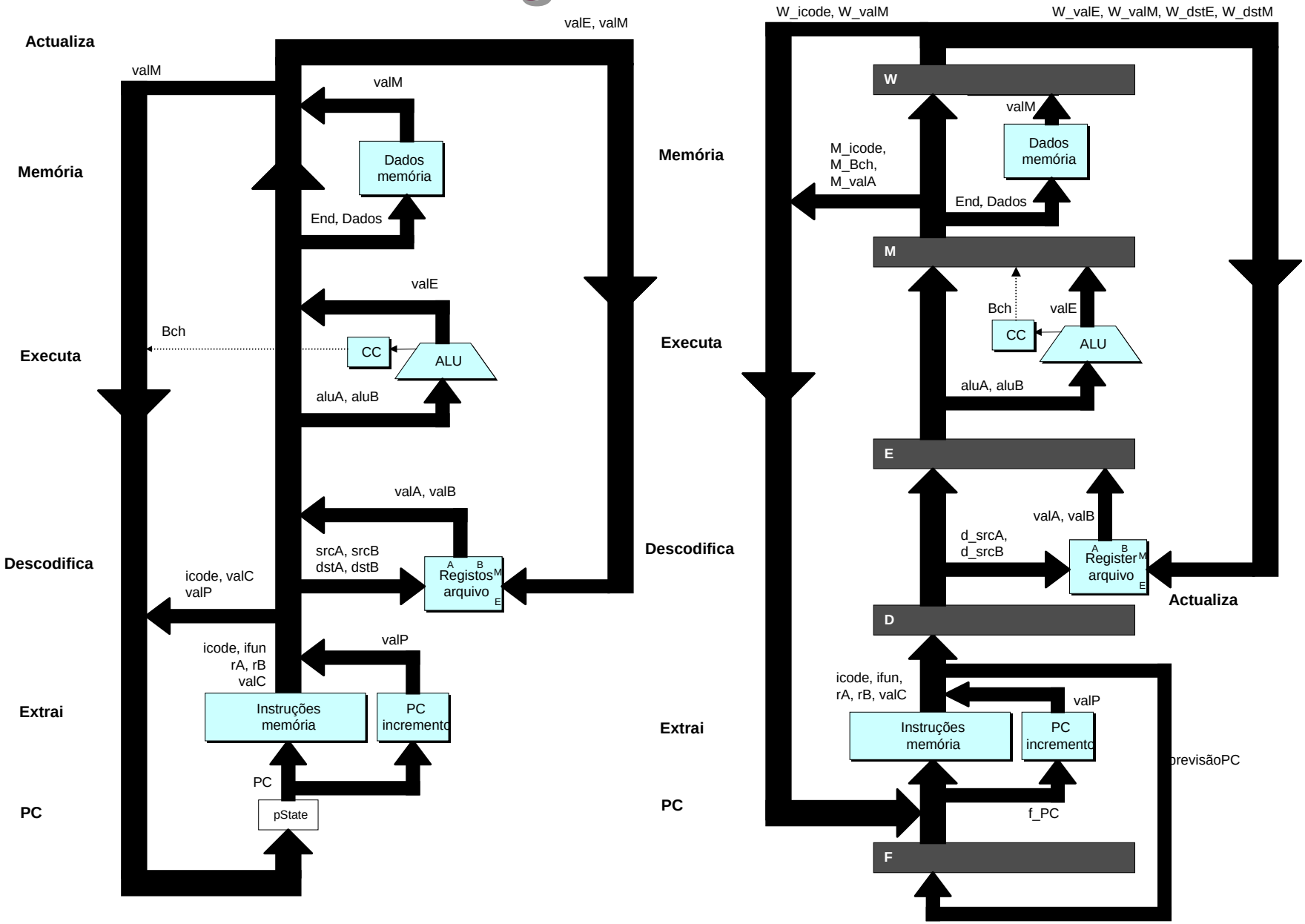
- Responsável por seleccionar o PC para a instrução corrente
- Baseado nos resultados produzidos na instrução prévia

Estado do Processador

- Não existe um registo para o valor de PC
- O valor PC é obtido a partir da informação disponível



Inclusão de Registos de Encadeamento



Estágios - Encadeamento

Extrai

- Selecciona o PC corrente
- Lê instruções
- Calcula novo valor PC

Descodifica

- Lê registos de programas

Executa

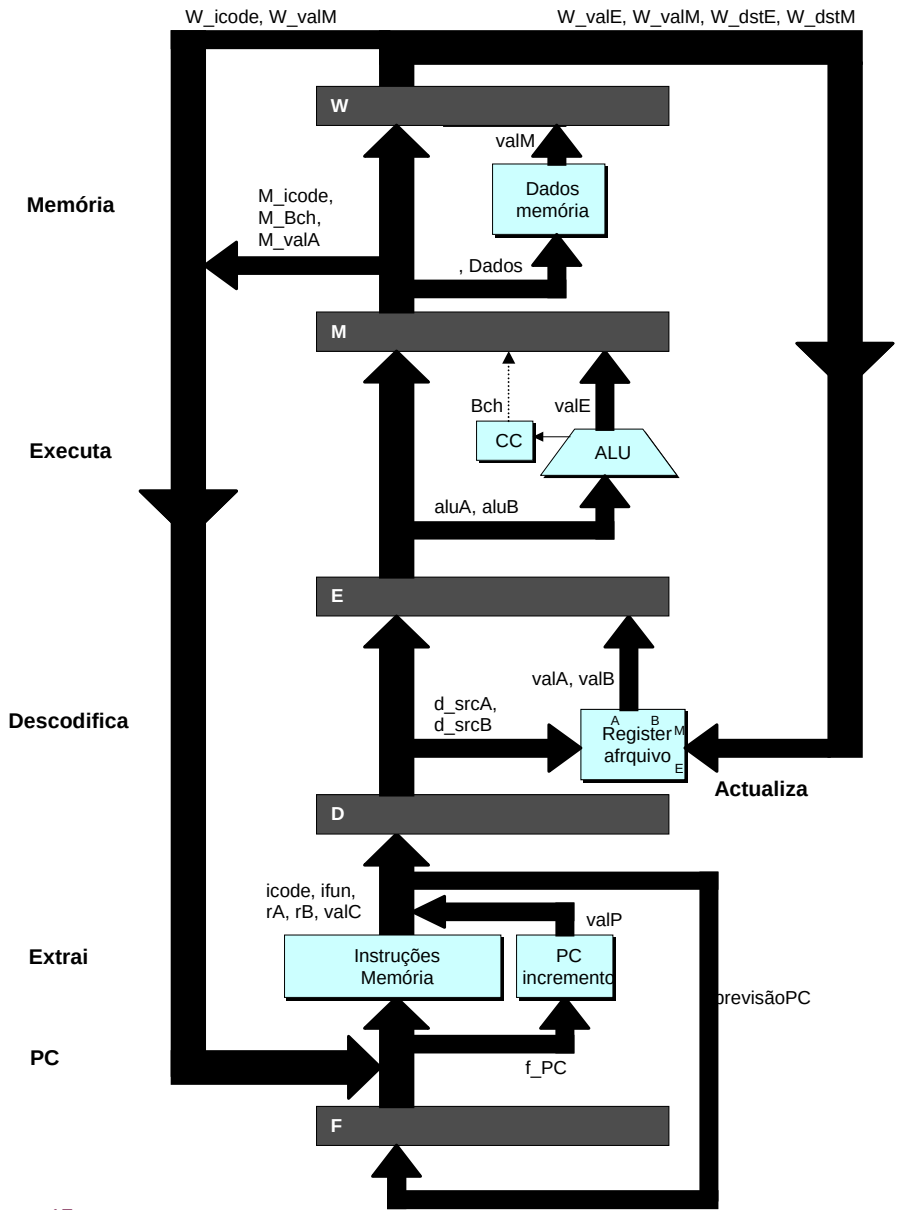
- Opera a ALU

Memória

- Lê/escreve na memória

Actualiza

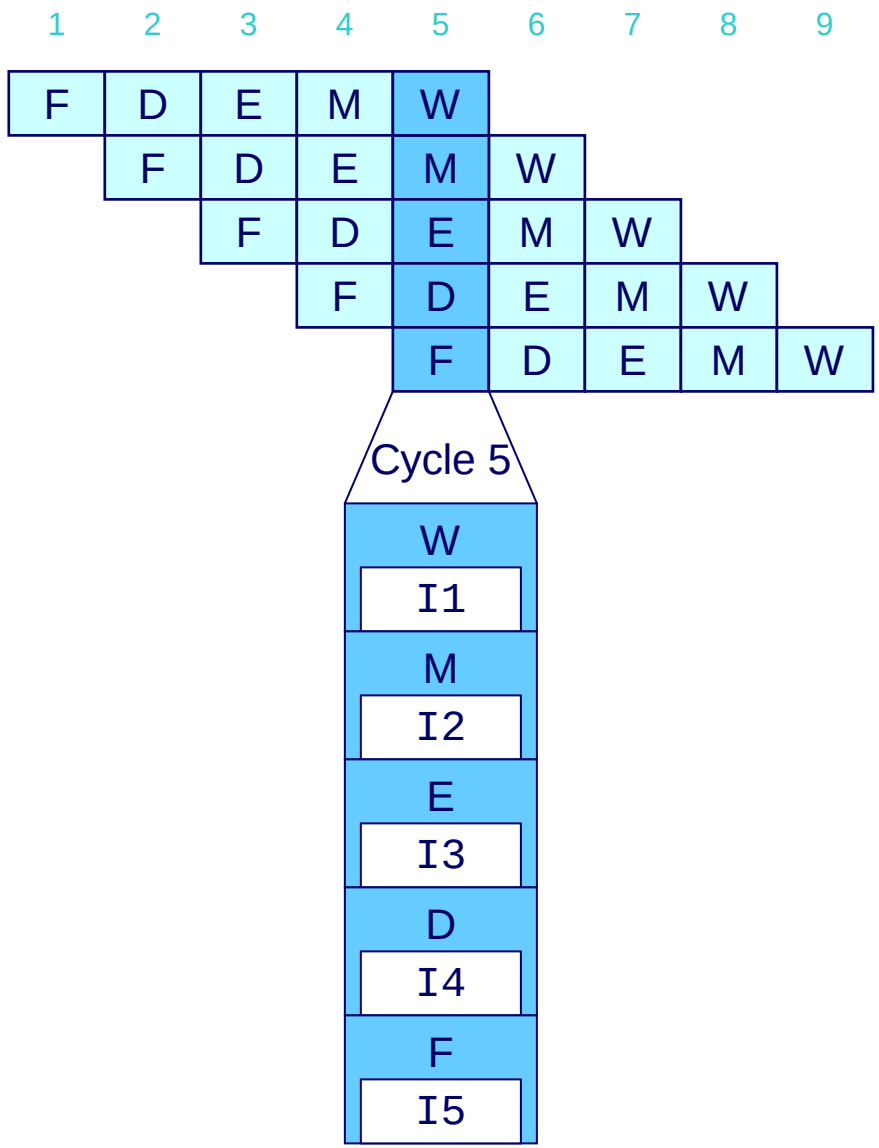
- Modifica o arquivo de registos



Encadeamento - Demonstração

demo-basic.ys

```
irmovl   $1,%eax   #I1
irmovl   $2,%ecx   #I2
irmovl   $3,%edx   #I3
irmovl   $4,%ebx   #I4
halt               #I5
```



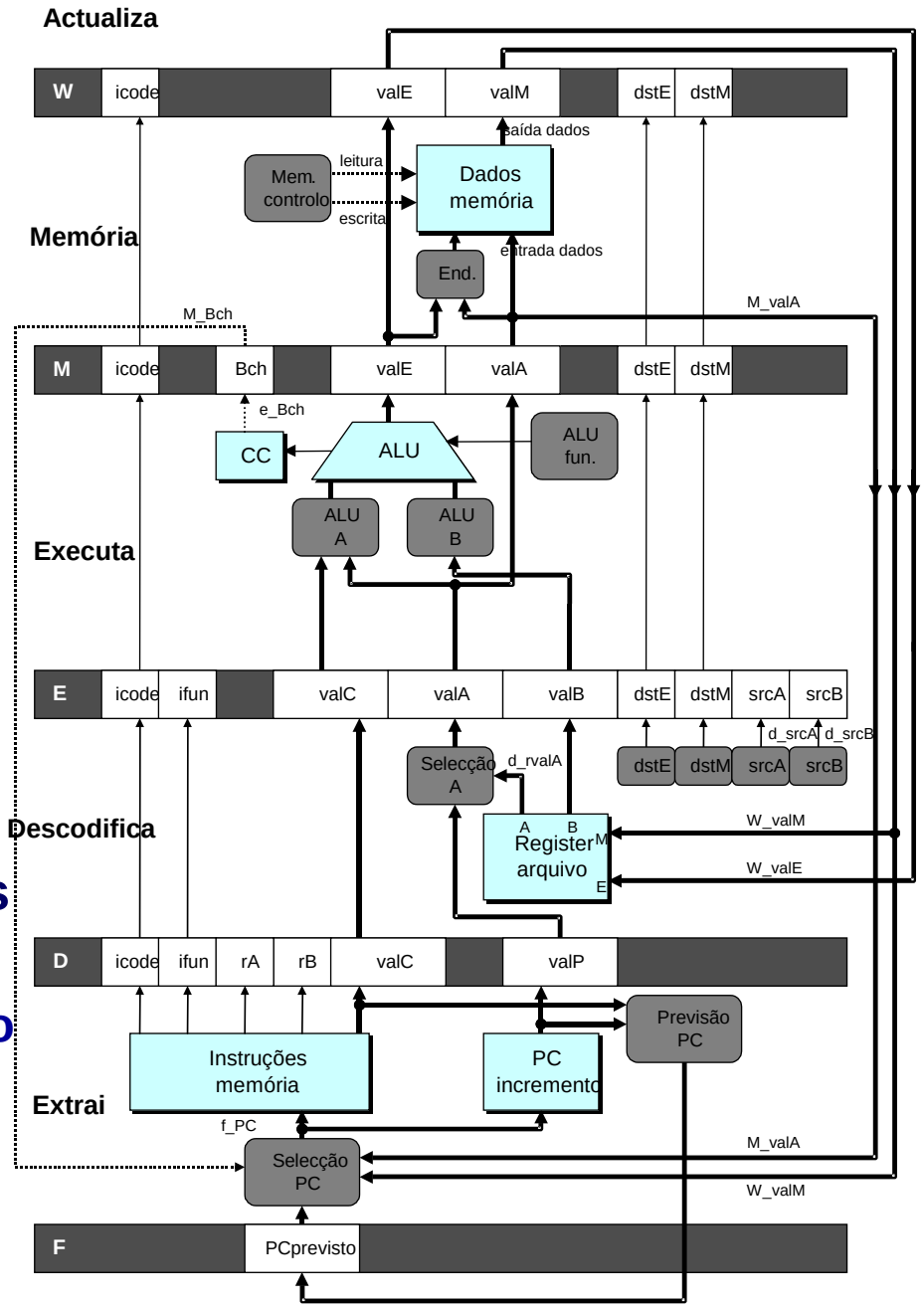
Encadeamento - *Hardware*

Registos de encadeamento

- guardam valores intermédios, resultantes da execução de instruções
- F, D, E, M, W

Encaminhamento

- Valores passados de um estágio para o próximo
- Não há saltos para estágios anteriores
  - e.g., valC passa através do estágio de descodificação



Caminhos de Realimentação

Previsão PC

- Adivinhar o próximo valor de PC

Informação de Salto

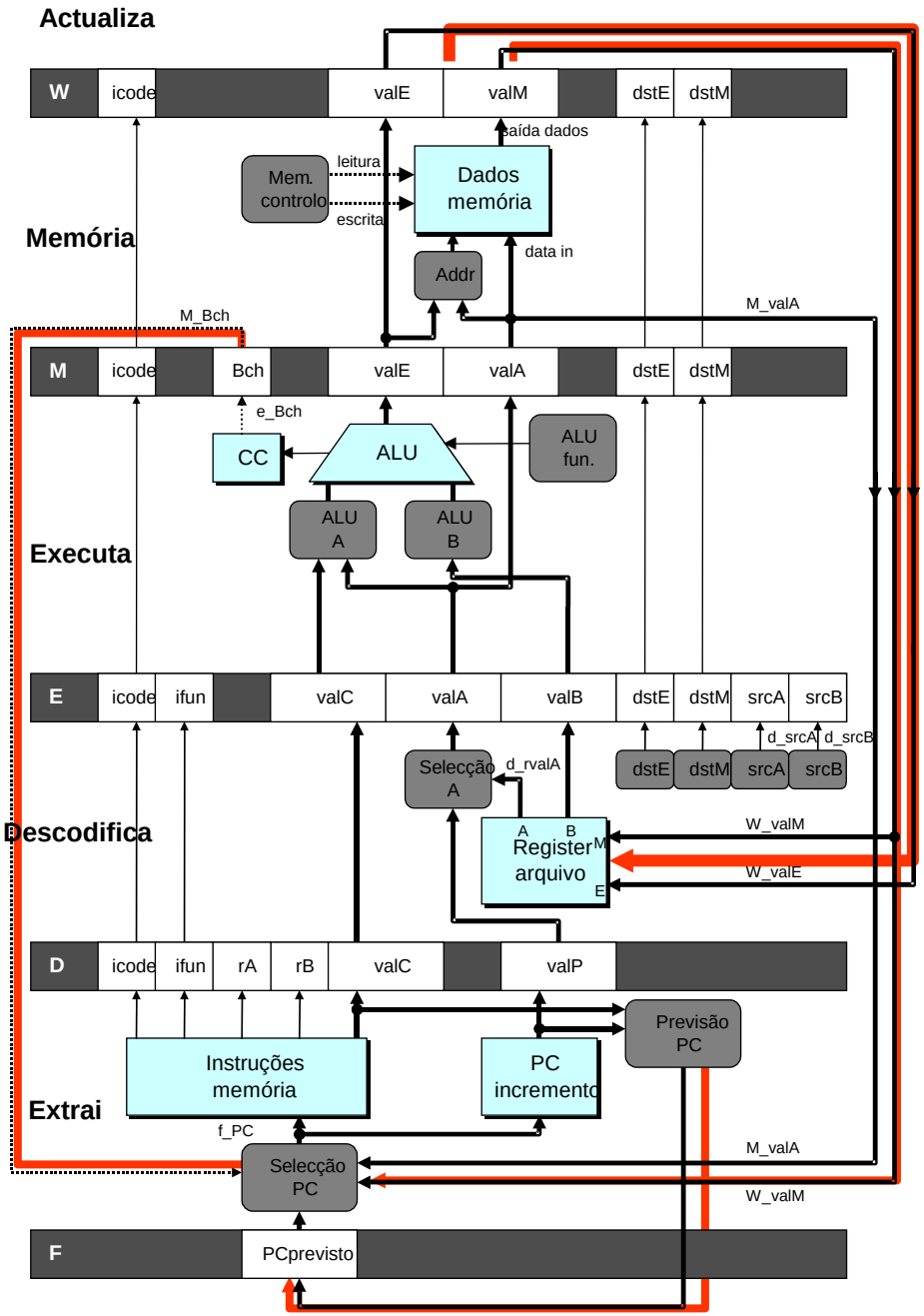
- *Jmp xxx*
- Segue/não-segue
- Destino/continua

Ponto de Retorno

- *Ret*
- Lido da memória

Actualização de registos

- Via portas de escrita do arquivo de registos



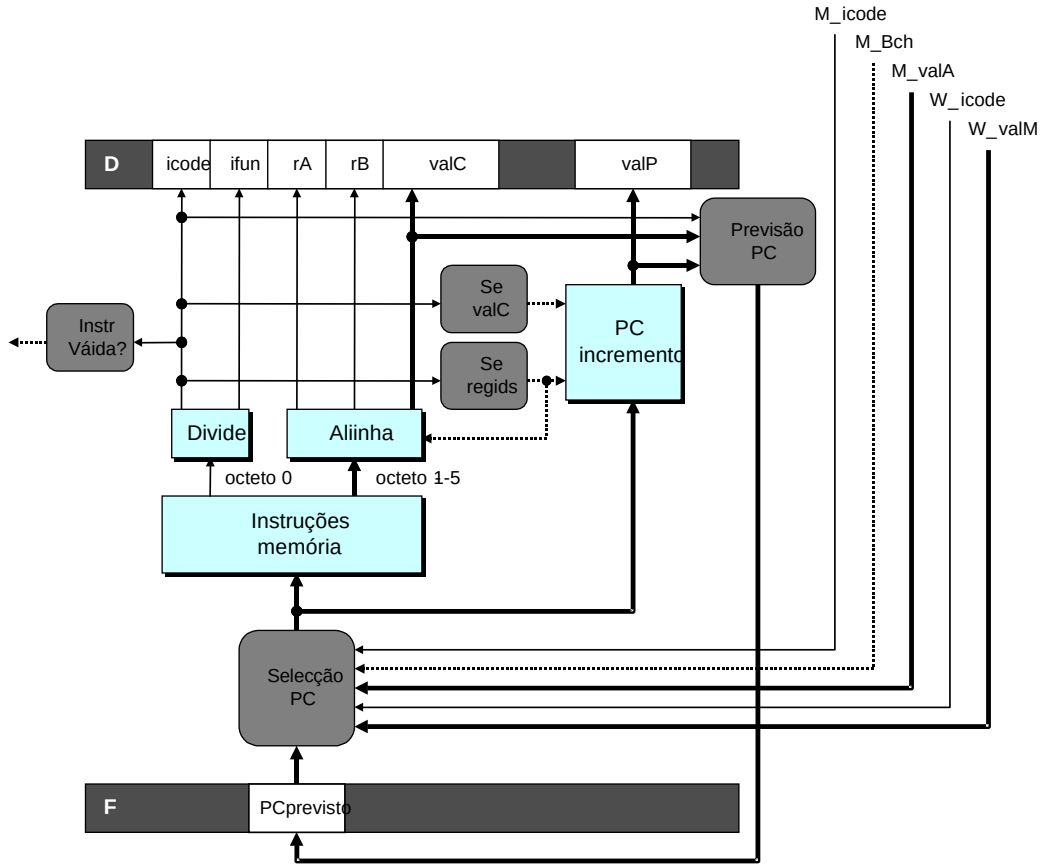
PC - Previsão

Objectivos

- Iniciar uma instrução em cada ciclo de relógio
- Débito - uma instrução em cada ciclo

Desafio

- Determinar próxima instrução



Nova instrução

- Imediatamente a seguir à conclusão do estágio de extracção anterior
- Tempo insuficiente para saber, com segurança, qual a próxima instrução

Adivinhar a próxima instrução

- Recuperar se a previsão falhar

Previsão – Acerta sempre

Instruções

- As que não transferem o controlo
- A previsão é o próximo PC ser valP
- Acerta sempre!

Chamadas a rotinas e saltos incondicionais

- Previsão:
  - próximo PC (destino) é valC
- Acerta sempre!

```
int new_F_predpc = [
        f_icode in {IJXX, ICALL} : f_valC;
        1 : f_valP;
];
```

Previsão - Com Falha

Saltos condicionais

- Previsão:
  - próximo PC (destino) é valC
- Acerta quando a condição se verifica!
  - Tipicamente 60% das vezes

Retorno

- A opção é não fazer previsão

```
int new_F_predPC = [
        # Falha na Previsão. Extrair do PC incrementado
        M_icode == IJXX && !M_Bch : M_ valA;
        # Finaliza instrução RET
        W_icode == IRET : W_valM;
        # Omissão: Usa valor previsto para PC;
        1 : f_valP;
];
```

Anomalias de Encadeamento

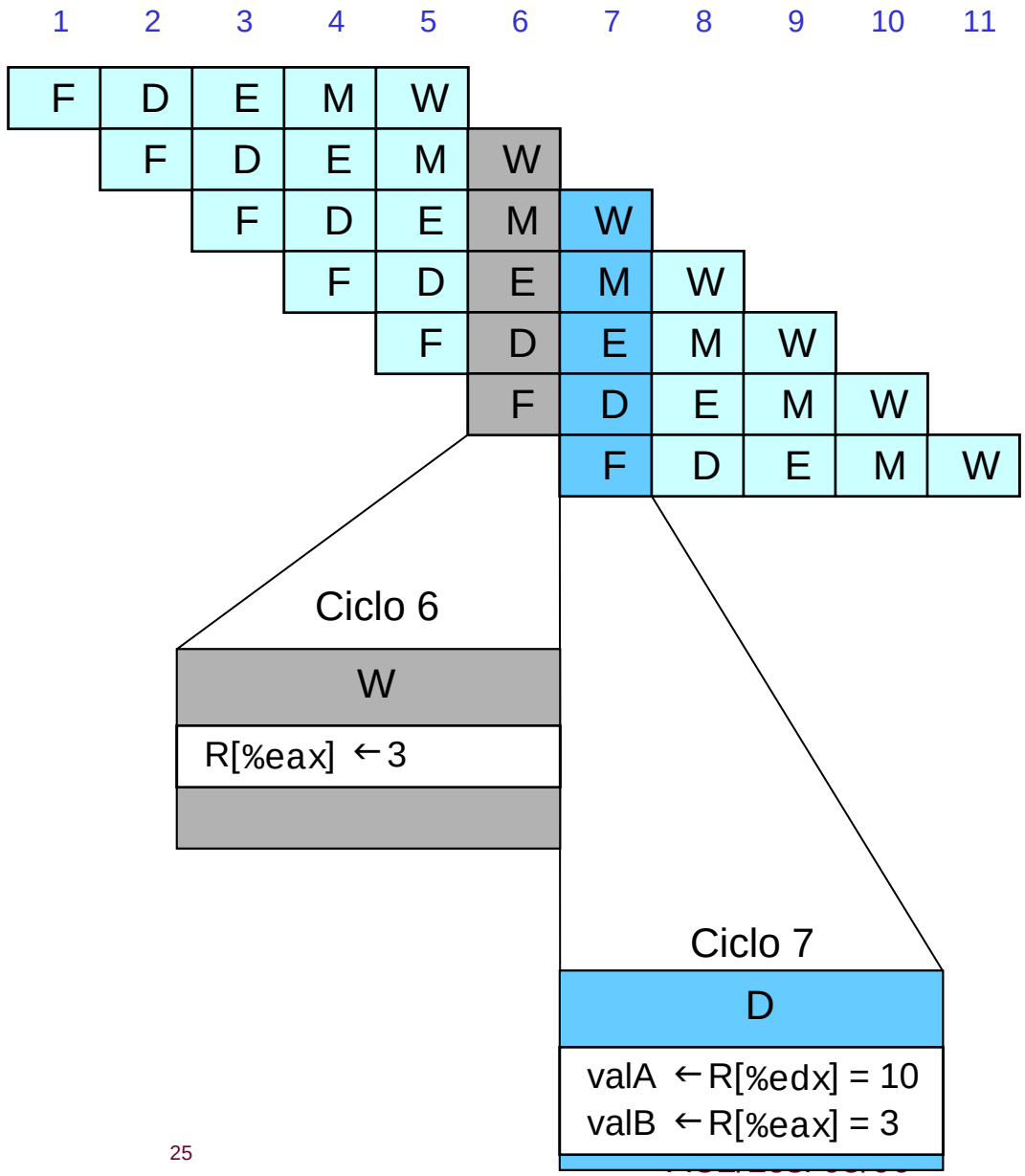
Dependências entre instruções

- Dados
  - Resultados calculados usados pela próxima instrução
- Controlo
  - Execução anterior determina a localização da próxima instrução
- Anomalias
  - Dados e controlo

```
1:   irmovl $10, %edx
2:   irmov  $3, %eax
3:   addl   %edx, %eax
```

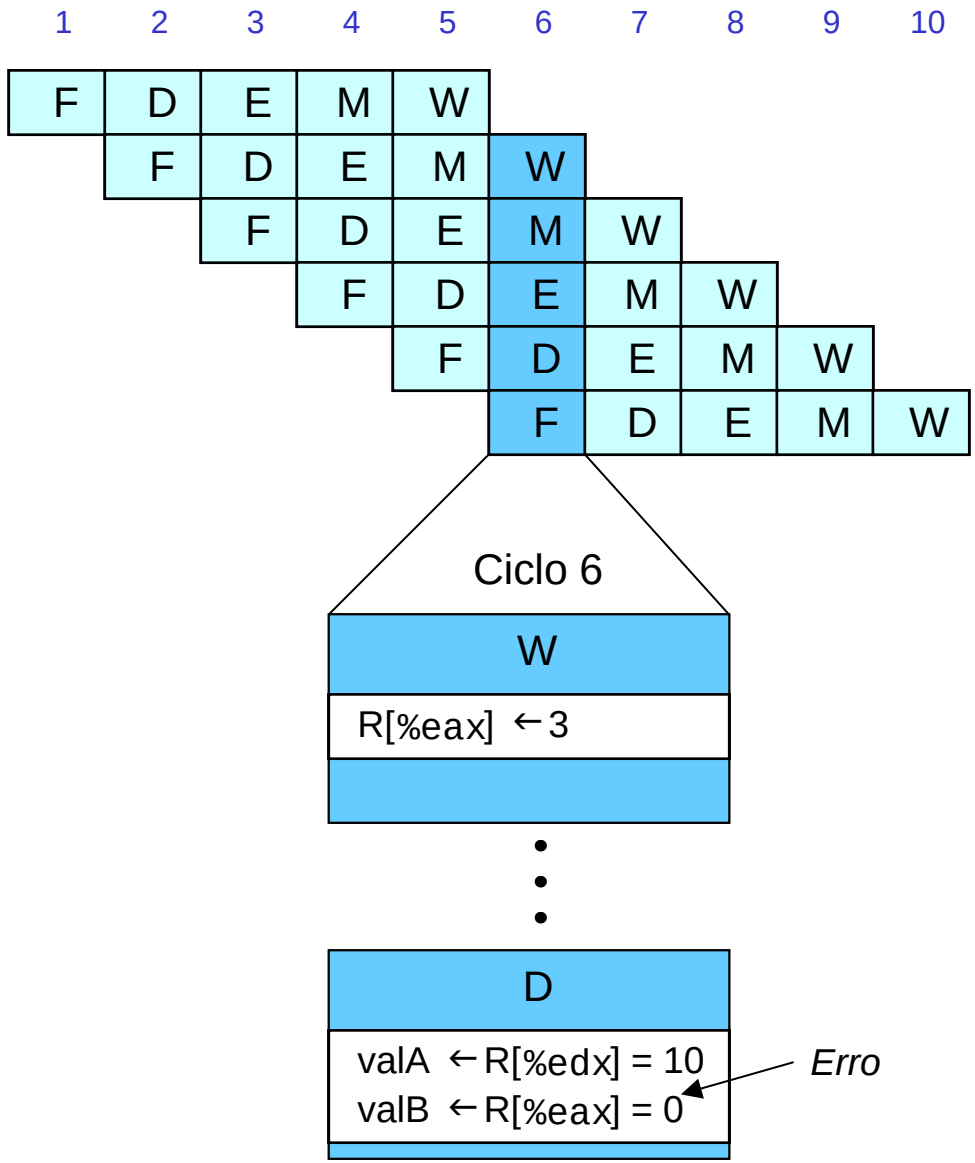
Dependências de Dados : 3 Nop's

```
# demo-h3.ys
 0x000: irmovl $10,%edx
 0x006: irmovl  $3,%eax
 0x00c: nop
 0x00d: nop
 0x00e: nop
 0x00f: addl %edx,%eax
 0x011: halt
```



Dependências de Dados: 2 Nop's

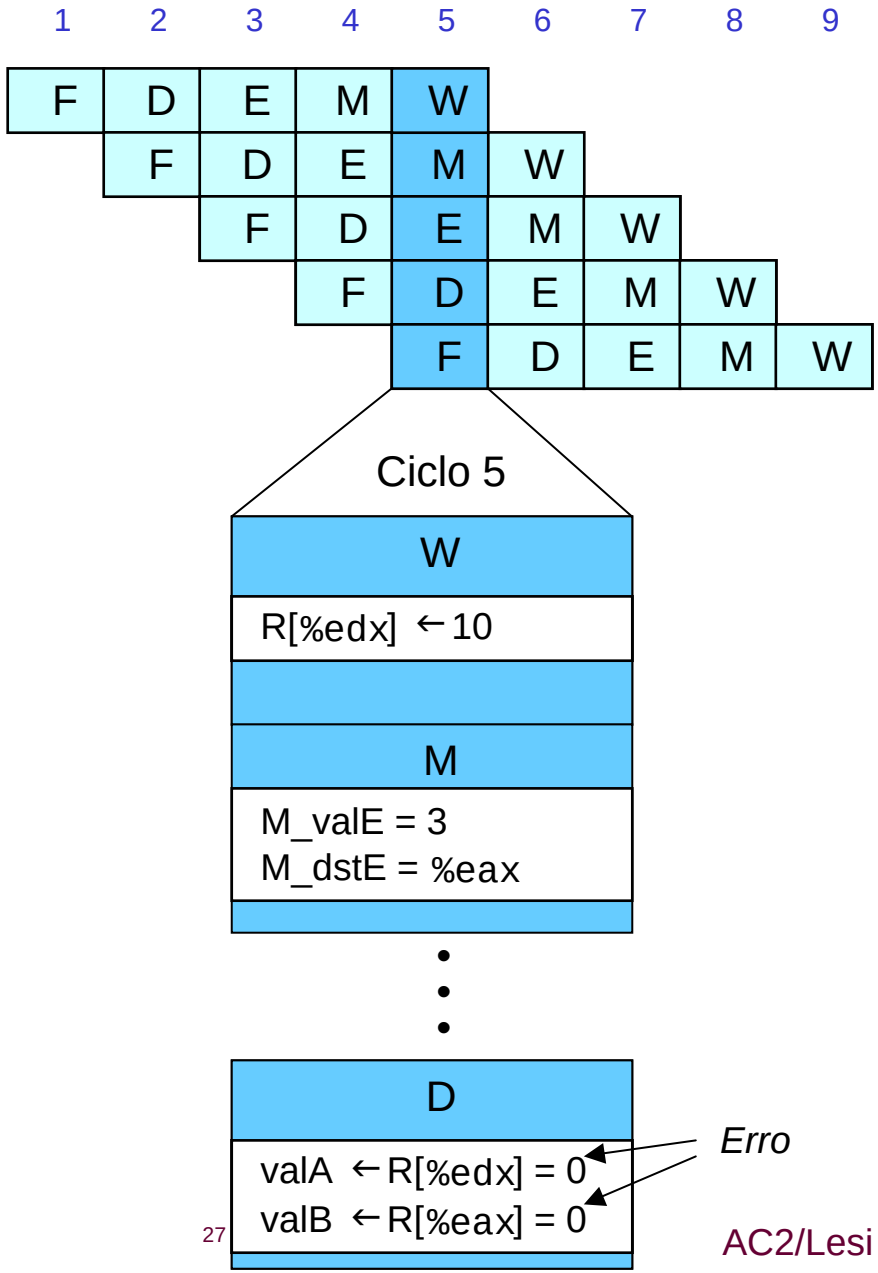
```
# demo-h2.ys
0x000: irmovl $10,%edx
0x006: irmovl  $3,%eax
0x00c: nop
0x00d: nop
0x00e: addl %edx,%eax
0x010: halt
```



Dependências de Dados: 1 Nop

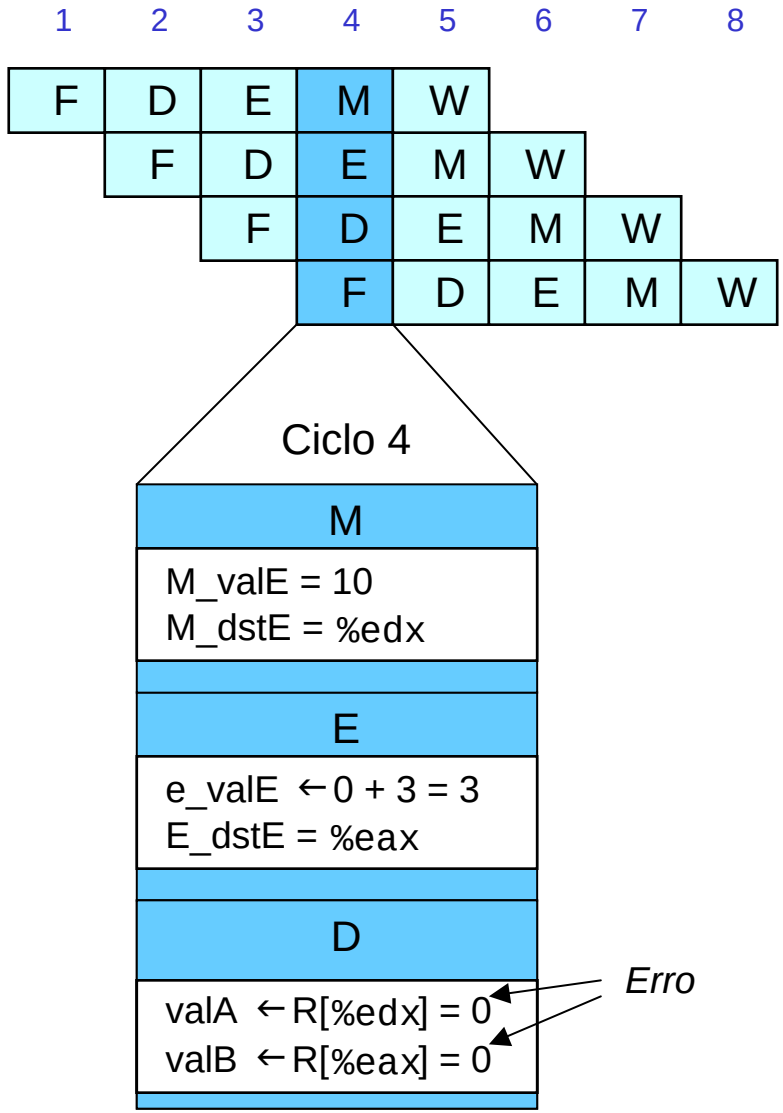
```
# demo-h1.ys

0x000: irmovl $10,%edx
0x006: irmovl  $3,%eax
0x00c: nop
0x00d: addl %edx,%eax
0x00f: halt
```



Dependências de Dados: Sem Nops

```
# demo-h0.ys
0x000: irmovl $10,%edx
0x006: irmovl  $3,%eax
0x00c: addl %edx,%eax
0x00e: halt
```



Previsão de Salto: Exemplo

Instruções a serem executadas

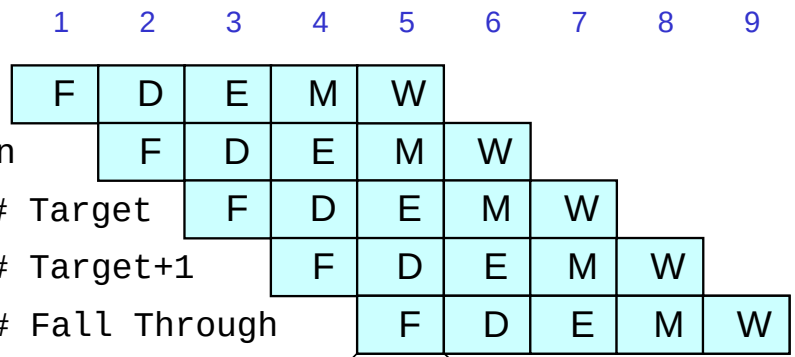
- Posição 0x000
  - %eax <- 0
- Posição 0x200
  - Condição falsa
- Posição 0x00d
  - Continua a execução até halt

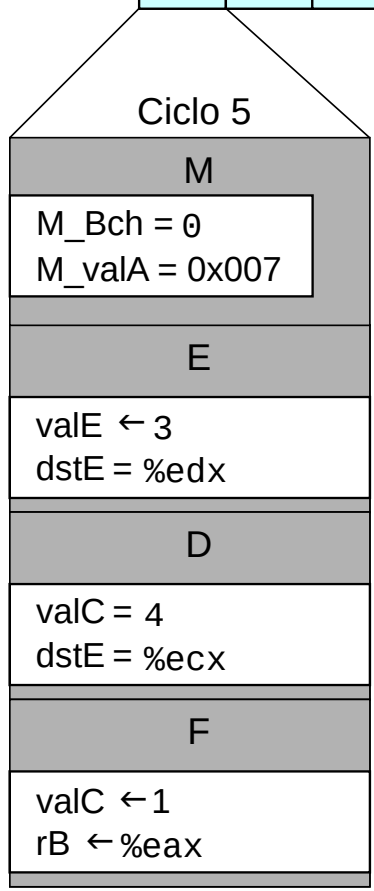
demo-j.ys

```
0x000:    xorl %eax,%eax      # %eax <- 0
0x002:    jne  t              # Não segue
0x007:    irmovl $1, %eax     # Continua
0x00d:    nop
0x00e:    nop
0x00f:    nop
0x010:    halt
0x011: t: irmovl $3, %edx     # Destino (Não executado)
0x017:    irmovl $4, %ecx     # Não executado
0x01d:    irmovl $5, %edx     # Não executado
```

Previsão de Salto: Traçado

```
# demo-j
0x000:    xorl %eax,%eax
0x002:    jne t # Not taken
0x011: t: irmovl $3, %edx # Target
0x017:    irmovl $4, %ecx # Target+1
0x007:    irmovl $1, %eax # Fall Through
```





Processamento da falha

- Executa incorrectamente
  - Posição 0x011
    - » *irmovl $3, %edx*
    - » *irmovl $4, %ecx*

Retorno: Exemplo

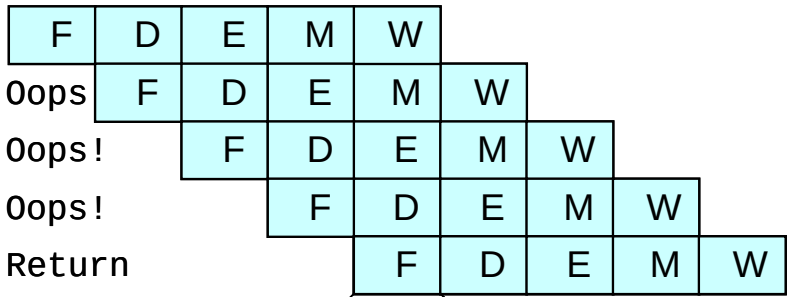
- Muitos nops para evitar anomalias

demo-ret.ys

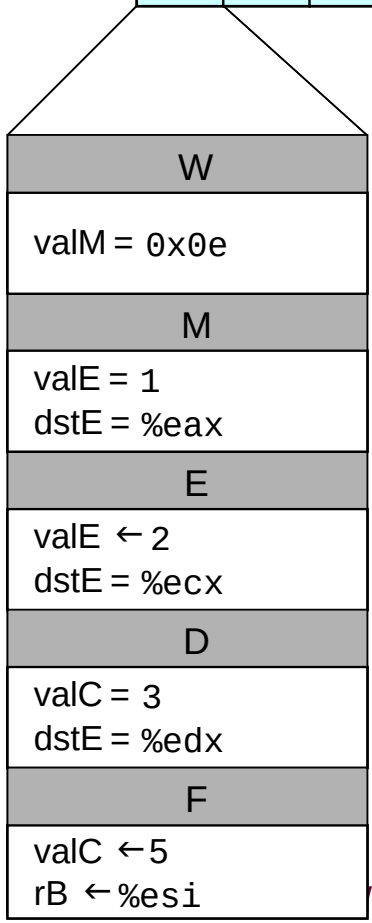
```
0x000:    irmovl Stack,%esp # Inicia apontador de pilha
0x006:    nop                         # Evita anomalia em %esp
0x007:    nop
0x008:    nop
0x009:    call p                      # Chama procedimento
0x00e:    irmovl $5,%esi              # Ponto de retorno
0x014:    halt
0x020: .pos 0x20
0x020: p: nop                         # Procedimento
0x021:    nop
0x022:    nop
0x023:    ret
0x024:    irmovl $1,%eax              # Não executado
0x02a:    irmovl $2,%ecx    # Não executado
0x030:    irmovl $3,%edx    # Não executado
0x036:    irmovl $4,%ebx              # Não executado
0x100: .pos 0x100
0x100: Stack:                         # Pilha: apontador de pilha
```

Retorno incorrecto: Exemplo

```
# demo-ret
0x023:    ret
0x024:    irmovl $1,%eax # Oops
0x02a:    irmovl $2,%ecx # Oops!
0x030:    irmovl $3,%edx # Oops!
0x00e:    irmovl $5,%esi # Return
```



- **Executa incorrectamente**
  - **Posição 0x024**
    - » ***irmovl $1, %eax***
    - » ***irmovl $2, %ecx***
    - » ***irmovl $3, %edx***



Encadeamento: Resumo

Conceito

- Dividir a execução de instruções em 5 estágios
- Executar instruções em cadeia

Limitações

- Não resolve dependências entre instruções com vizinhança apertada
- Dependências de Dados
  - Uma instrução escreve num registo, subsequentemente uma outra instrução lê-o
- Dependências de Controlo
  - A instrução ajusta o PC com um valor incorrectamente previsto pelo encadeamento
  - Falha de previsão e instrução de retorno

Soluções

- Nos próximos capítulos!