Arquitetura de Computadores
- Processadores Superescalares

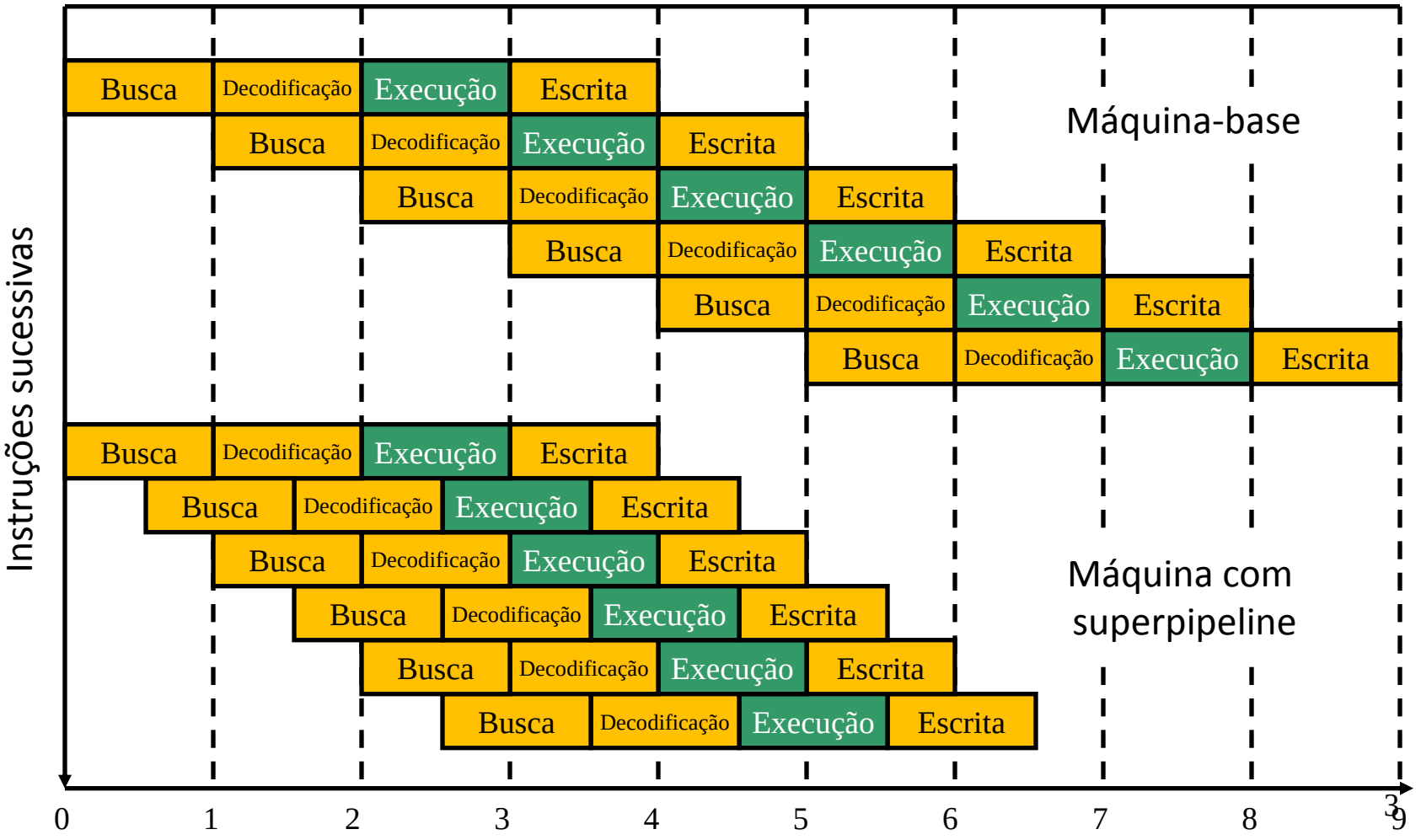
por

Helcio Wagner da Silva

Introdução

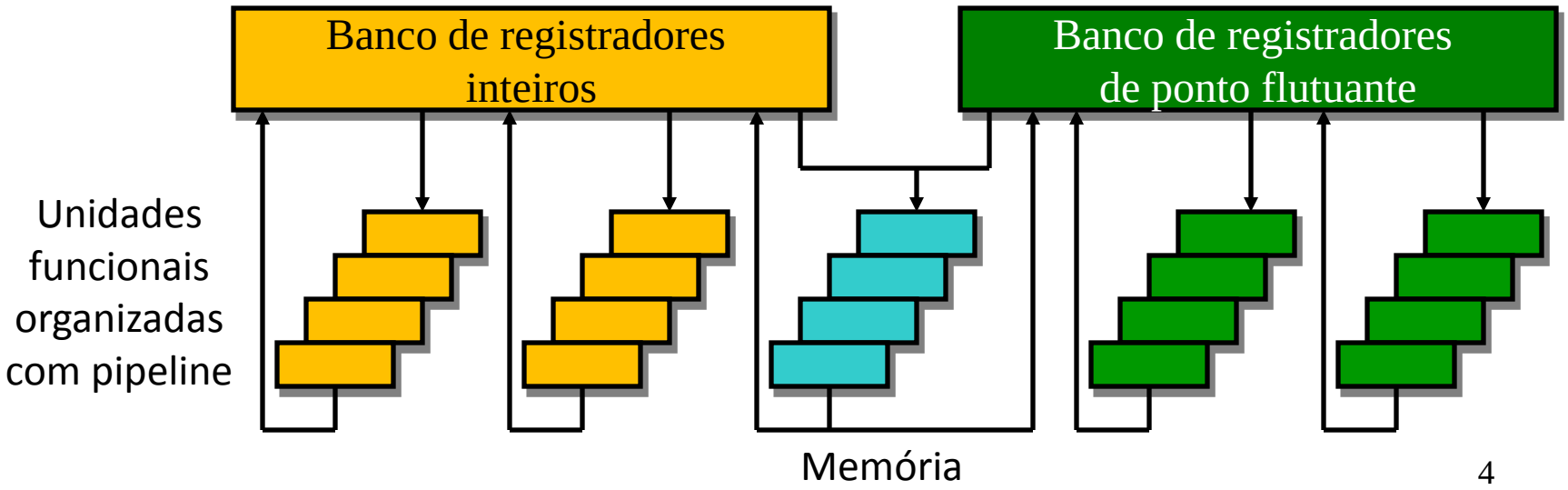
- O Pipeline é uma técnica desenvolvida para a melhoria do desempenho frente à execução seqüencial de instruções
- Uma evolução natural ao Pipeline é o Superpipeline
- O Superpipeline explora o fato de que muitos estágios de um pipeline requerem um tempo menor que a metade de um ciclo de relógio

Conceito de Superpipeline

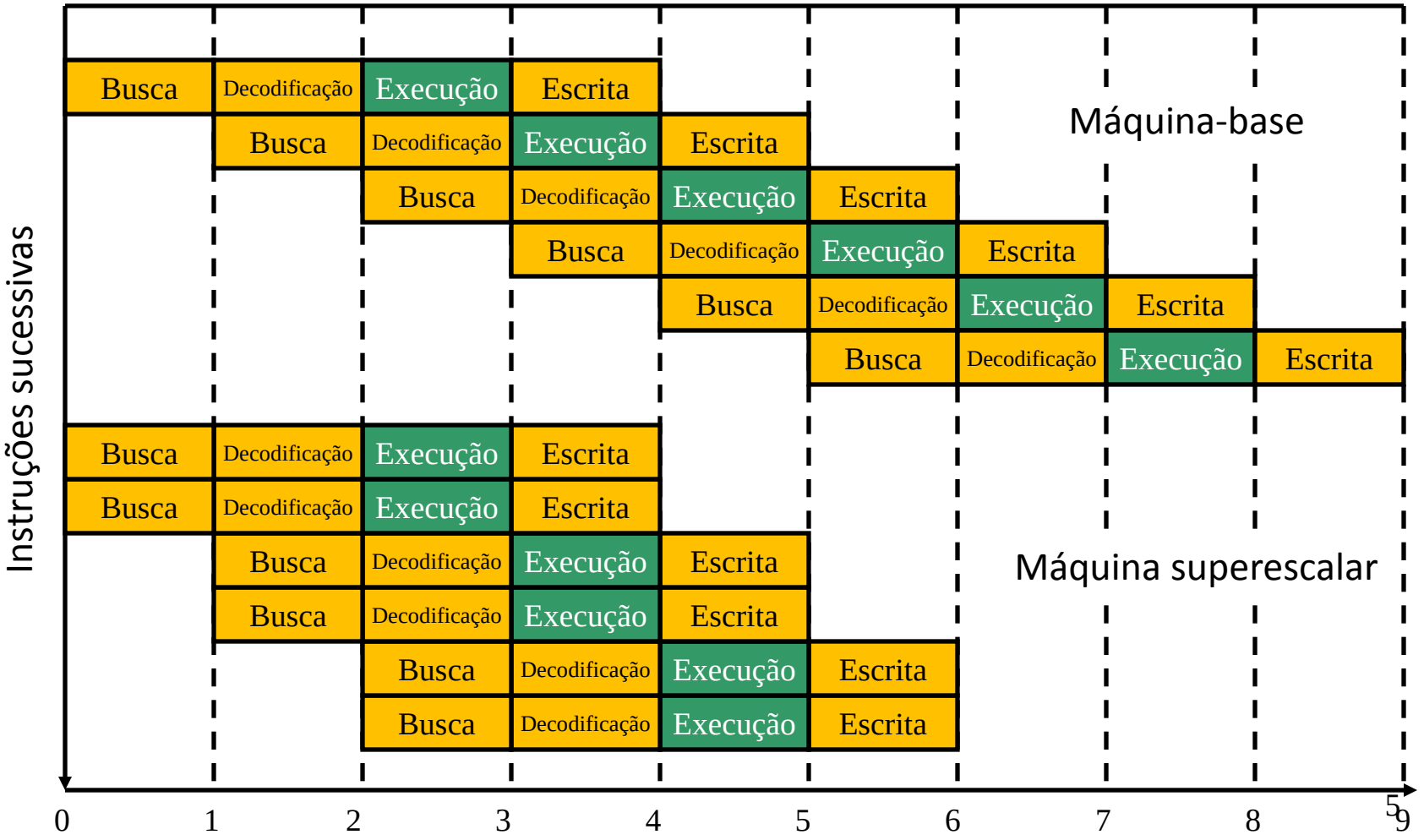


Conceito de Superescalar

- Essência da abordagem superescalar é a execução de instruções em pipelines paralelos



Conceito de Superescalar



Conceito de Superescalar

- A abordagem superescalar depende da habilidade de executar várias instruções em paralelo
- O **paralelismo no nível de instruções** diz respeito ao nível no qual as instruções e um programa podem ser executadas em paralelo
- Para maximizar esse paralelismo, pode ser usada uma combinação de otimização baseada em compilador e técnicas de hardware

Limitações

- As seguintes limitações ao paralelismo no nível de instruções podem ser listadas
  - Dependência de dados verdadeira
  - Dependência de desvio
  - Conflito de recurso
  - Dependência de saída
  - Antidependência

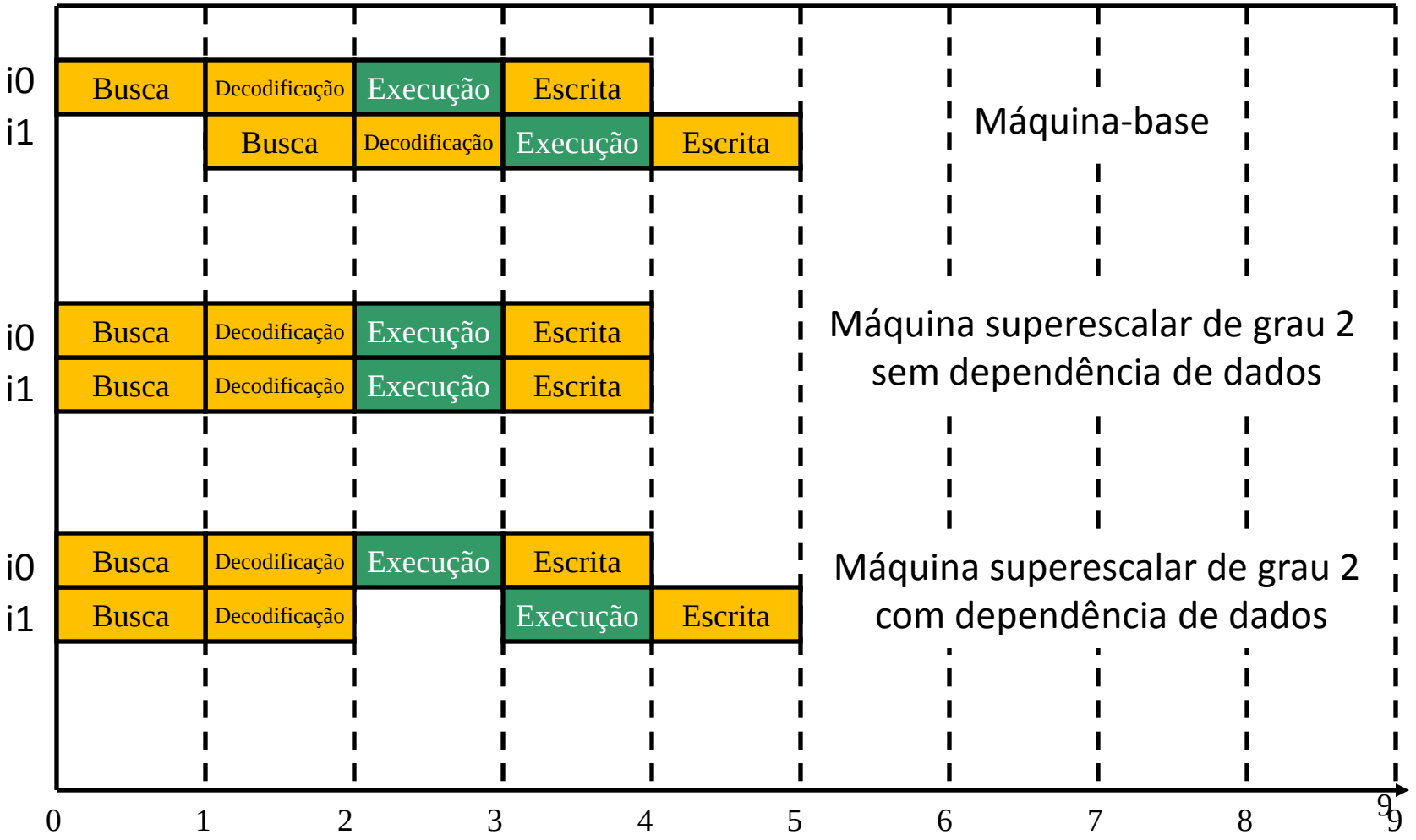
Dependência de Dados Verdadeira

- Exemplo:

```
add r1, r2 ;carregar registrador
           r1 com a soma dos
           conteúdos de r1 e r2
mov r3, r1 ;carregar registrador
           r3 com conteúdo de r1
```

  - A segunda instrução não pode ser executada até que seja completada a execução da primeira instrução

Dependência de Dados Verdadeira



Dependência de Dados Verdadeira

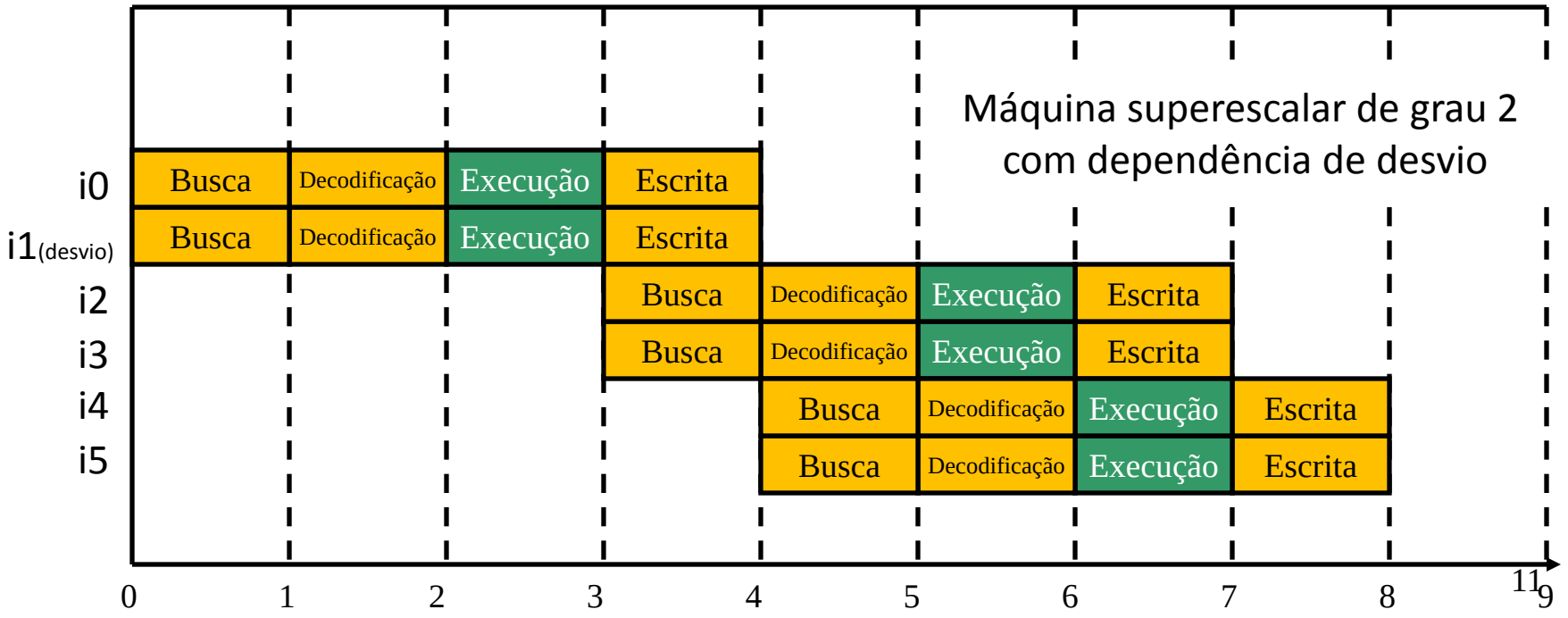
- Situação piora quando dados são carregados a partir da memória, e não de um registrador

```
load r1, eff ;carregar r1 com o
             conteúdo do endereço
             de memória eff
move r3, r1  ;carregar registrador r3
             com o conteúdo de r1
```

  - Um µP leva dois ou mais ciclos para transferir um valor da memória para um registrador

Dependência de Desvios

- Uma instrução seguinte a uma instrução de desvio condicional não pode ser executada até que seja completada a execução da instrução de desvio

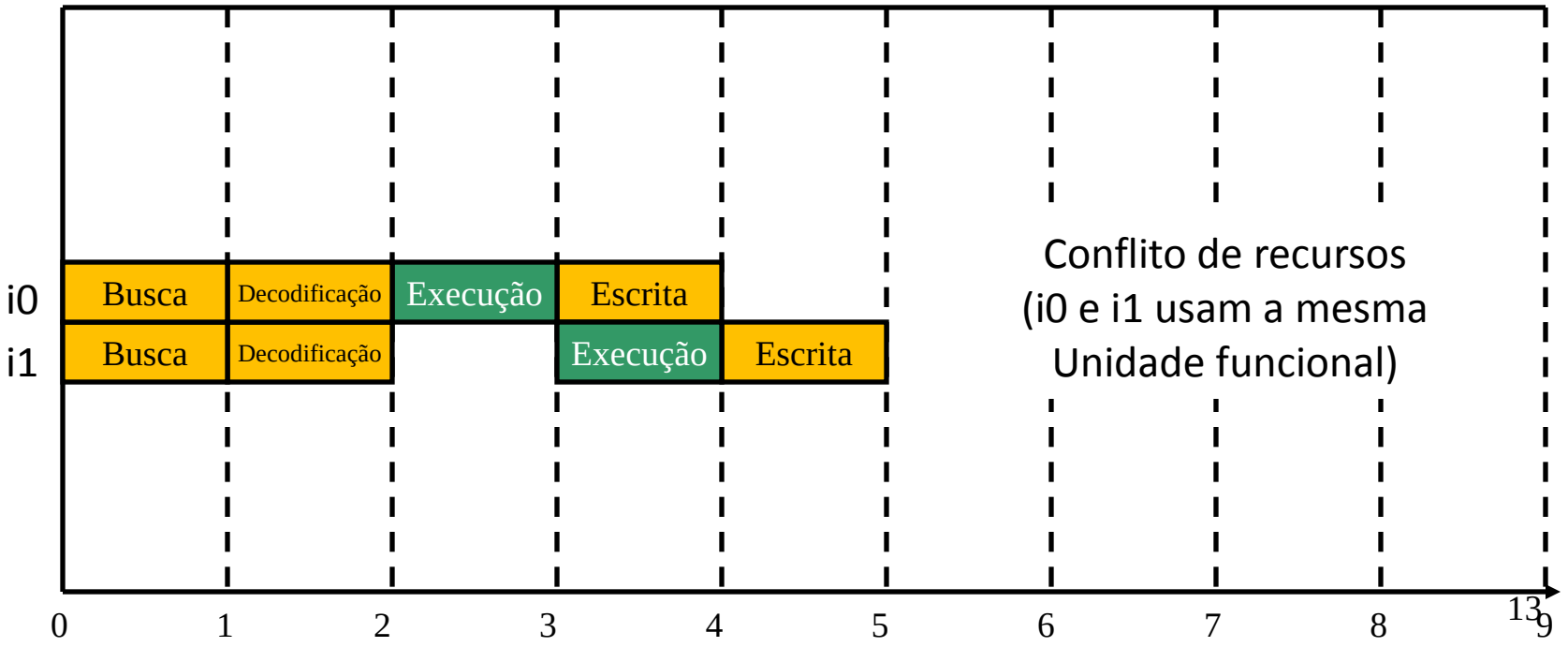


Conflito de Recurso

- Ocorre quando duas ou mais instruções competem, ao mesmo tempo, por um mesmo recurso
- Exemplos de recursos:
  - Caches
  - Barramentos
  - Registradores
  - Unidades funcionais
    - O somador da ULA, por exemplo

Conflito de Recurso

- Semelhante, graficamente, à dependência de dados
- Pode ser eliminada com a duplicação de recursos, entretanto



Paralelismo no Nível de Instruções e Paralelismo de Máquina

- Há paralelismo no nível de instruções quando as instruções de uma seqüência são independentes e podem, portanto, ser executadas em paralelo

```
Load R1 ← R2
Add R3 ← R3, “1”
Add R4 ← R4, R2
```

Instruções independentes
(podem ser executadas em paralelo)

```
Add R3 ← R3, “1”
Add R4 ← R3, R2
Store [R4] ← R0
```

Instruções não são independentes
(não podem ser executadas em paralelo)

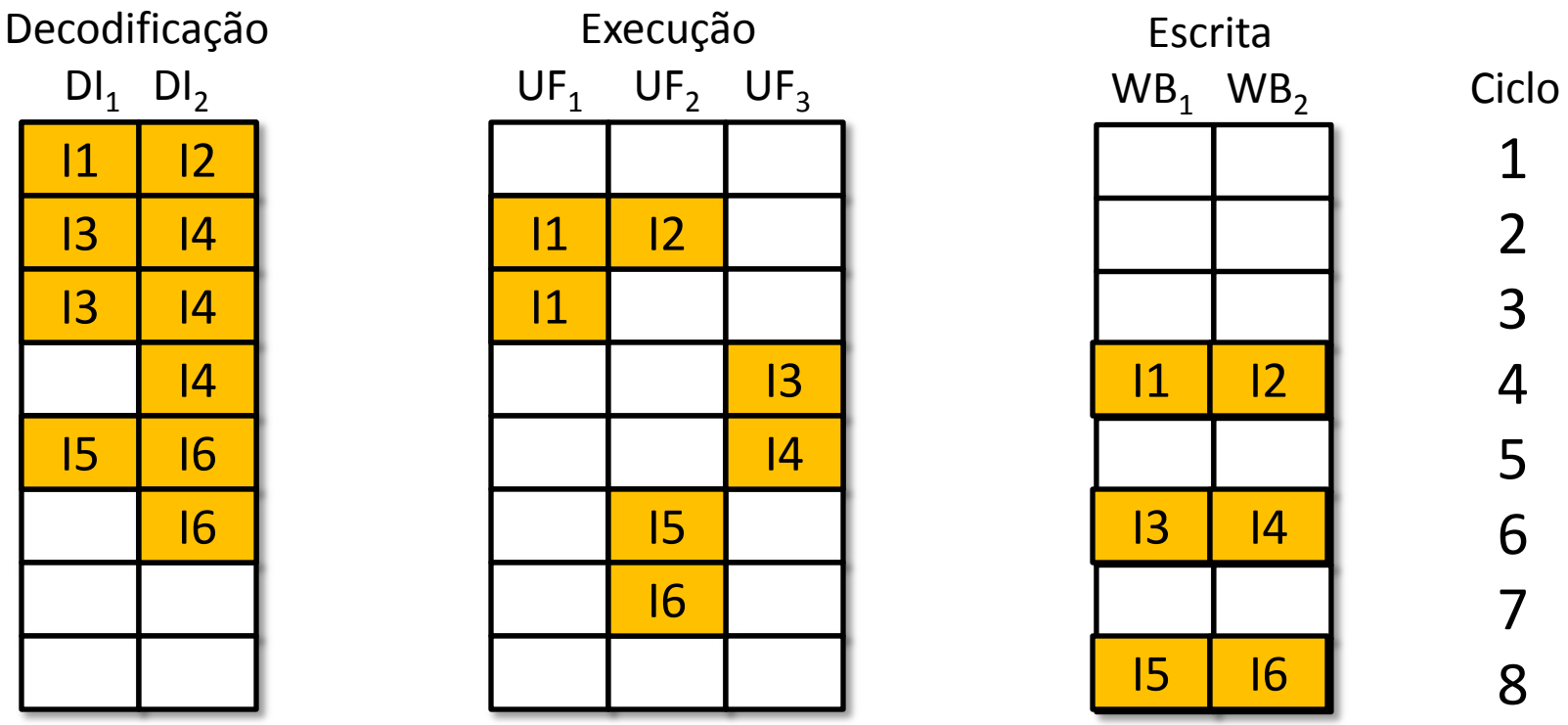
Paralelismo no Nível de Instruções e Paralelismo de Máquina

- O paralelismo de máquina é determinado pelo número de instruções que podem ser buscadas e executadas ao mesmo tempo (número de pipelines paralelos) e pela eficácia dos mecanismos usados para identificar instruções independentes
- É uma medida da capacidade do µP em aproveitar o paralelismo no nível de instruções

Políticas de Iniciação de Instruções

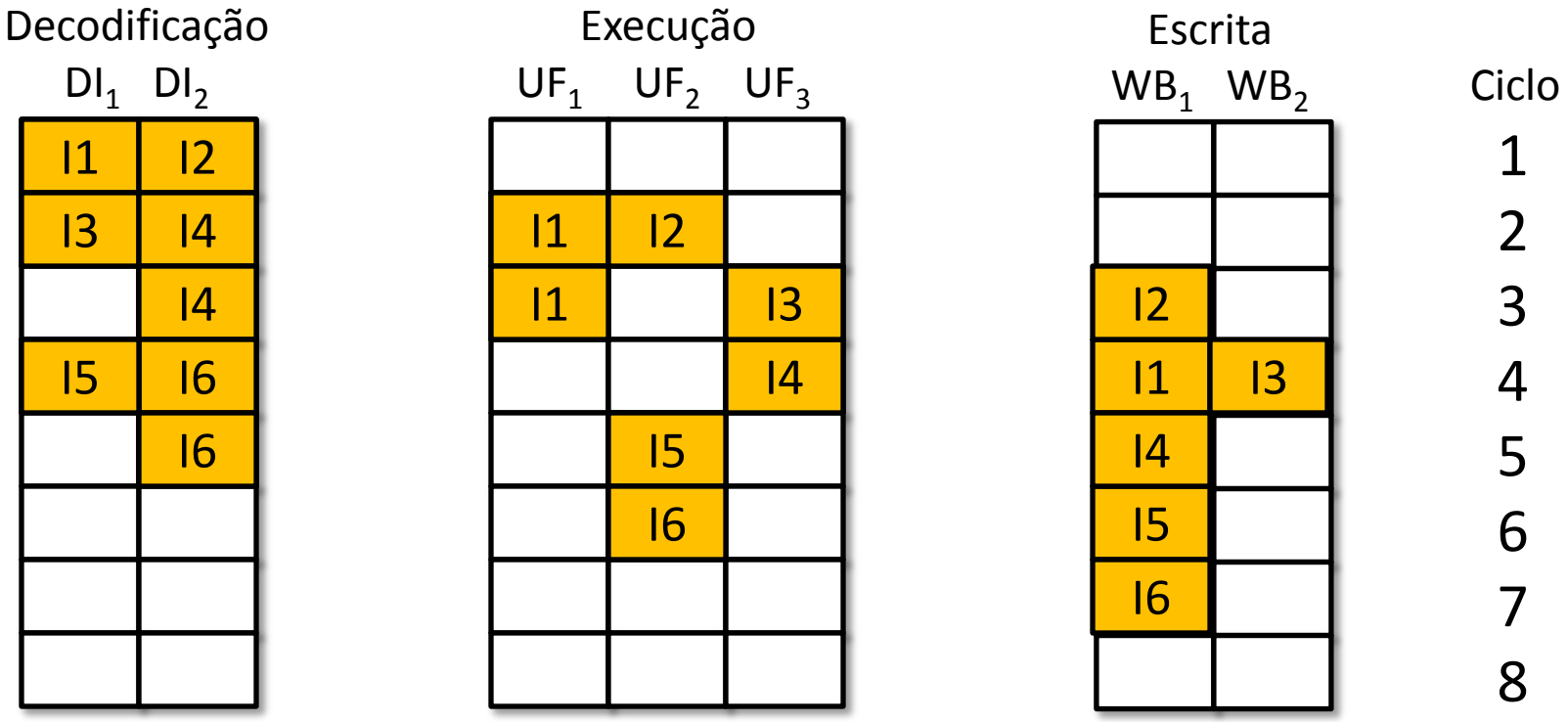
- Para otimizar o uso dos recursos do pipeline, o µP deve alterar a ordem de execução das instruções
- Tipos de políticas usadas
  - Iniciação em ordem com terminação em ordem
  - Iniciação em ordem com terminação fora de ordem
  - Iniciação fora de ordem com terminação fora de ordem
- A única restrição é a de que o resultado da execução deve ser igual ao resultado que seria obtido com a execução seqüencial

Iniciação em Ordem com Terminação em Ordem



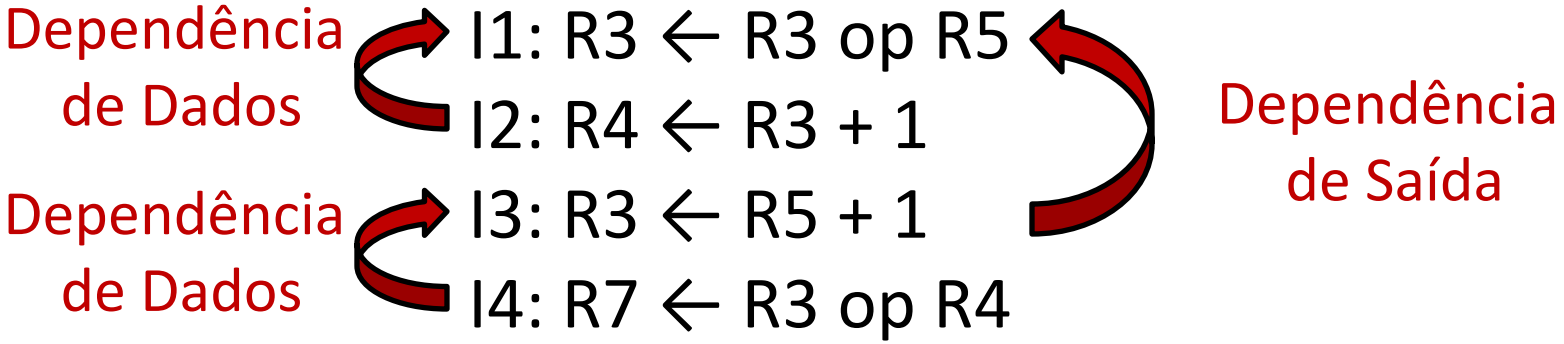
- I1 requer dois ciclos para ser executada
- I3 e I4 competem pela unidade funcional UF_3
- I5 depende do valor produzido por I4
- I5 e I6 competem pela unidade funcional UF_2

Iniciação em Ordem com Terminação Fora de Ordem



- I1 requer dois ciclos para ser executada
- I3 e I4 competem pela unidade funcional UF_3
- I5 depende do valor produzido por I4
- I5 e I6 competem pela unidade funcional UF_2

Dependência de Saída



- Dependência de Dados Verdadeira entre I1 e I2 é óbvia
- O mesmo se aplica a I3 e I4
- E qual é a relação entre I1 e I3?
  - Se I3 for completada antes de I1, o valor de R3 usado na execução de I4 estará errado
  - Portanto, I3 deve ser completada depois de I1 para que os valores de saída produzidos estejam corretos

Iniciação Fora de Ordem com Terminação Fora de Ordem

- Com a iniciação em ordem, o µP decodifica instruções apenas até o ponto em que ocorre uma dependência ou conflito
- Nenhuma instrução adicional é decodificada até que o conflito seja resolvido
- Assim, o µP não pode examinar instruções adiante do ponto de conflito, para tentar encontrar instruções independentes que possam ser introduzidas no pipeline

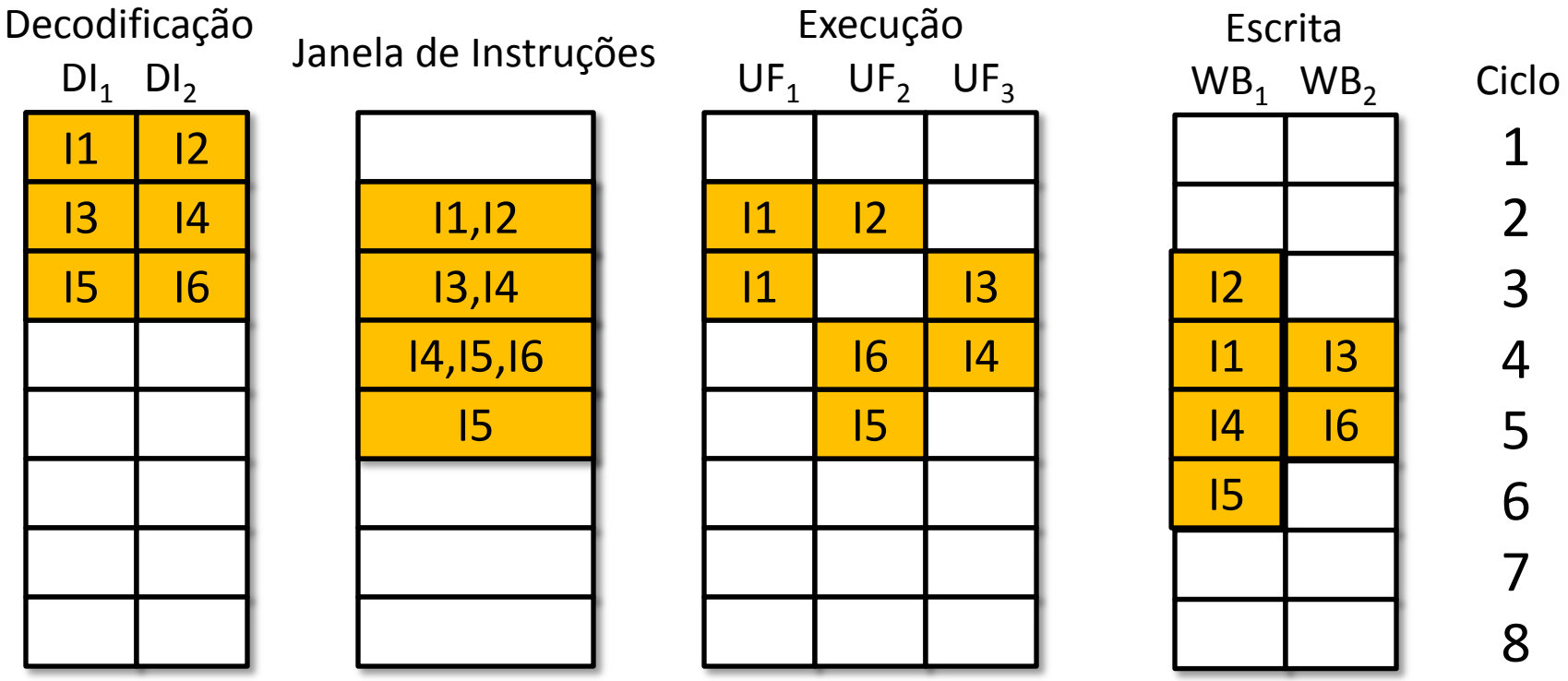
Iniciação Fora de Ordem com Terminação Fora de Ordem

- Para a execução de instruções fora de ordem, é necessário desvincular os estágios de decodificação e execução do pipeline
- Isso é feito usando uma área de armazenamento temporário, chamada **janela de instruções**
- Depois que o µP termina a decodificação de uma instrução, ela é colocada na janela de instruções
- O µP pode continuar a busca e decodificação de novas instruções, desde que a janela de instruções não esteja cheia

Iniciação Fora de Ordem com Terminação Fora de Ordem

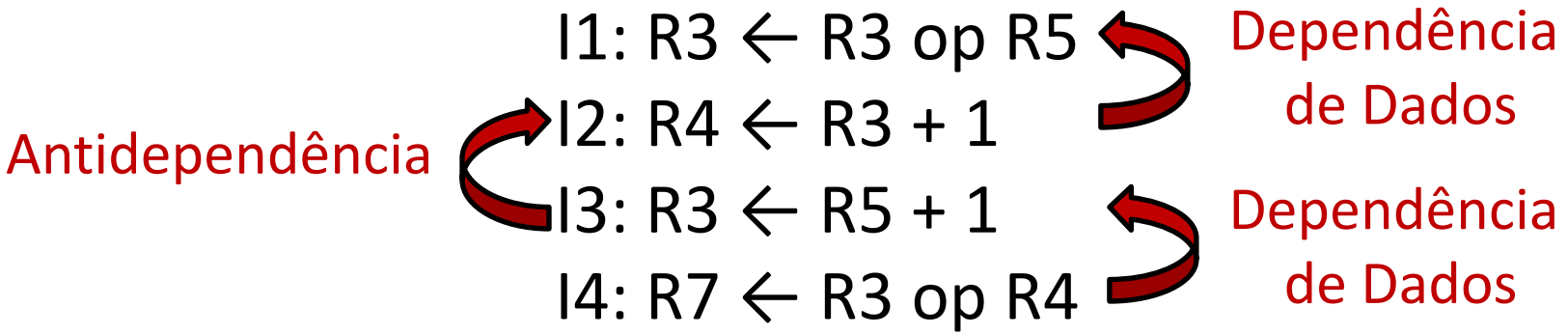
- Quando uma unidade funcional de execução se torna disponível, uma instrução presente na janela pode ser executada
- Qualquer instrução pode ser executada, desde que:
  - Necessite da unidade que está disponível
  - Nenhum conflito ou dependência a bloqueie
- O resultado é que o µP pode examinar instruções à frente, identificando instruções independentes que podem ser trazidas para o estágio de execução

Iniciação Fora de Ordem com Terminação Fora de Ordem



- I1 requer dois ciclos para ser executada
- I3 e I4 competem pela unidade funcional UF_3
- I5 depende do valor produzido por I4
- I5 e I6 competem pela unidade funcional UF_2

Antidependência



- A execução de I3 não pode ser completada antes da execução de I2
- Isso acontece porque I3 atualiza o registrador R3, que constitui um operando para a instrução I2
- A **antidependência** é semelhante a uma dependência de dados verdadeira, porém invertida
  - Ao invés de a primeira instrução produzir um valor usado pela segunda instrução, a segunda destrói um valor usado pela primeira

Renomeação de Registradores

- Objetiva eliminar/diminuir a ocorrência de dependências de saída e antidependências

$$I1: R3_b \leftarrow R3_a \text{ op } R5_a$$
$$I2: R4_b \leftarrow R3_b + 1$$
$$I3: R3_c \leftarrow R5_a + 1$$
$$I4: R7_b \leftarrow R3_c \text{ op } R4_b$$

- Objetiva eliminar/diminuir a ocorrência de dependências de saída e antidependências
  - Criação de $R3_c$ em I3 evita a ocorrência de antidependência com I2 e dependência de saída com I1