Superescalar

Arquitetura de Computadores

Bruno Prado

Departamento de Computação / UFS

Introdução

- O que é um *superescalar*?
  - É um processador com capacidade de executar instruções independentemente e concorrentemente em diferentes estágios do *pipeline*

Introdução

- O que é um *superescalar*?
  - É um processador com capacidade de executar instruções independentemente e concorrentemente em diferentes estágios do *pipeline*
  - Pode permitir que a execução das instruções em uma ordem diferente da sequência do programa

Introdução

- O que é um *superescalar*?
  - É um processador com capacidade de executar instruções independentemente e concorrentemente em diferentes estágios do *pipeline*
  - Pode permitir que a execução das instruções em uma ordem diferente da sequência do programa
  - Este termo surgiu em 1987 para definir esta técnica de projeto que aumenta a escala das operações

Introdução

- *Cycles per Instruction* (CPI)
  - É uma métrica para avaliar quantos ciclos de relógio são necessários para executar uma instrução
  - Permite avaliar o desempenho do processador

$$CPI = \frac{\#Ciclos}{\#Instruções}$$

Introdução

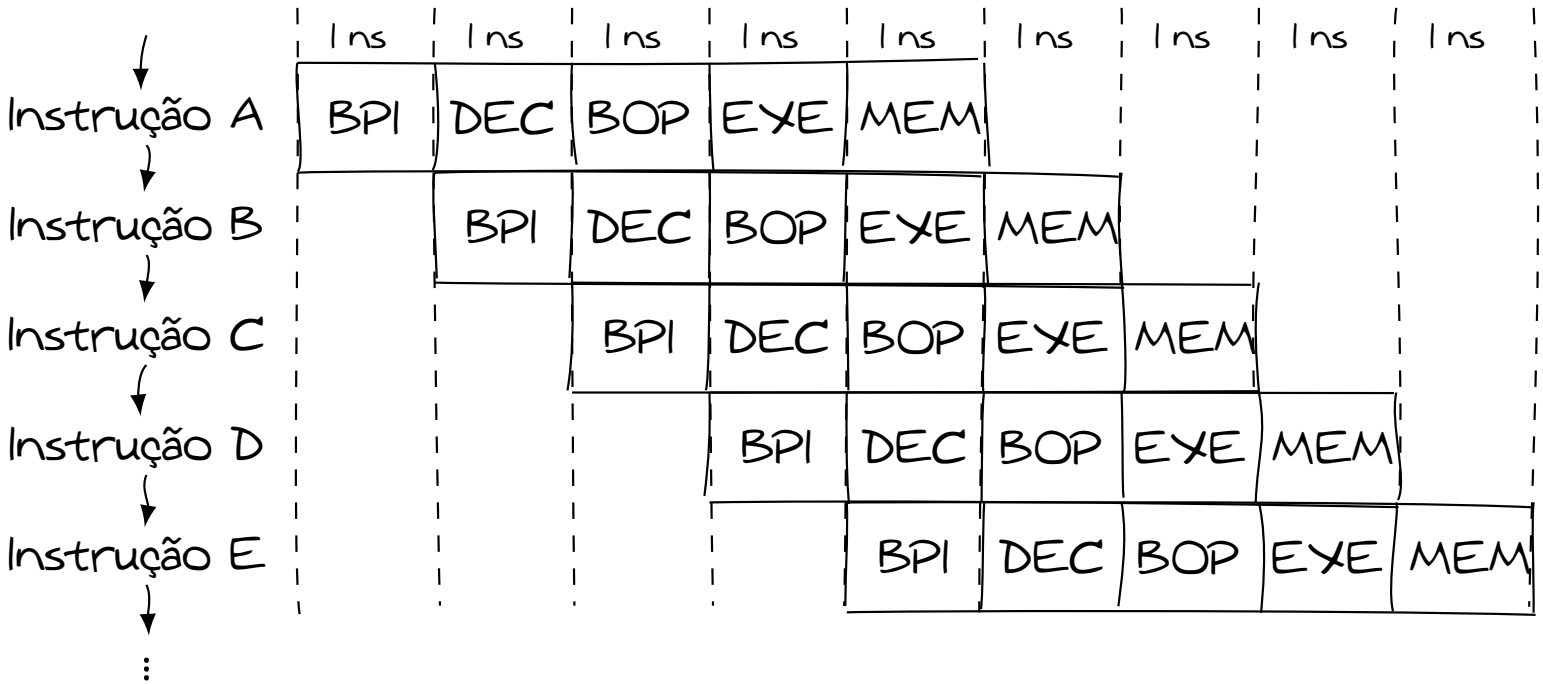
- *Cycles per Instruction* (CPI)
  - É uma métrica para avaliar quantos ciclos de relógio são necessários para executar uma instrução
  - Permite avaliar o desempenho do processador

$$CPI = \frac{\#Ciclos}{\#Instruções}$$

$$\downarrow CPI \quad \longleftrightarrow \quad \uparrow Desempenho$$

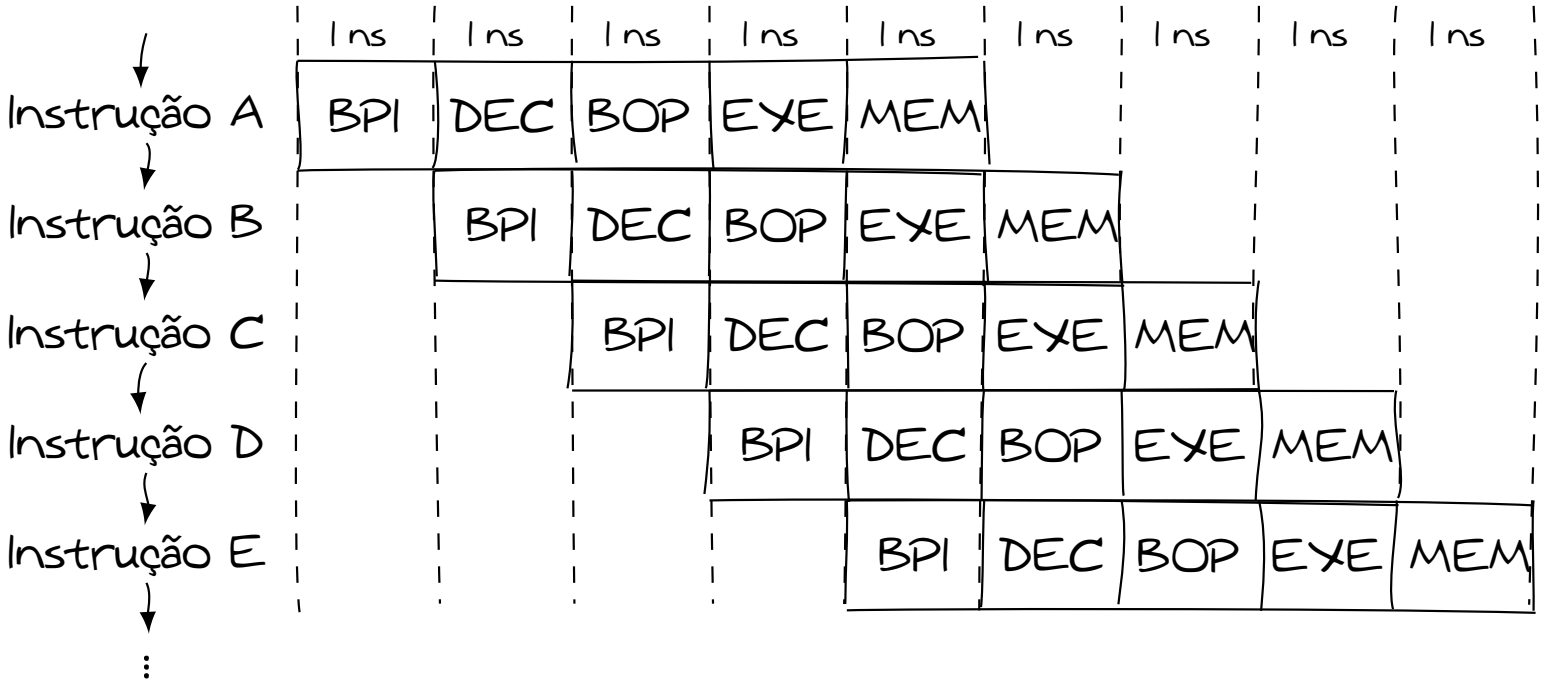
Introdução

- Implementação em *pipeline*
  - Não existe execução concorrente das instruções, mas existe o aumento da taxa de execução



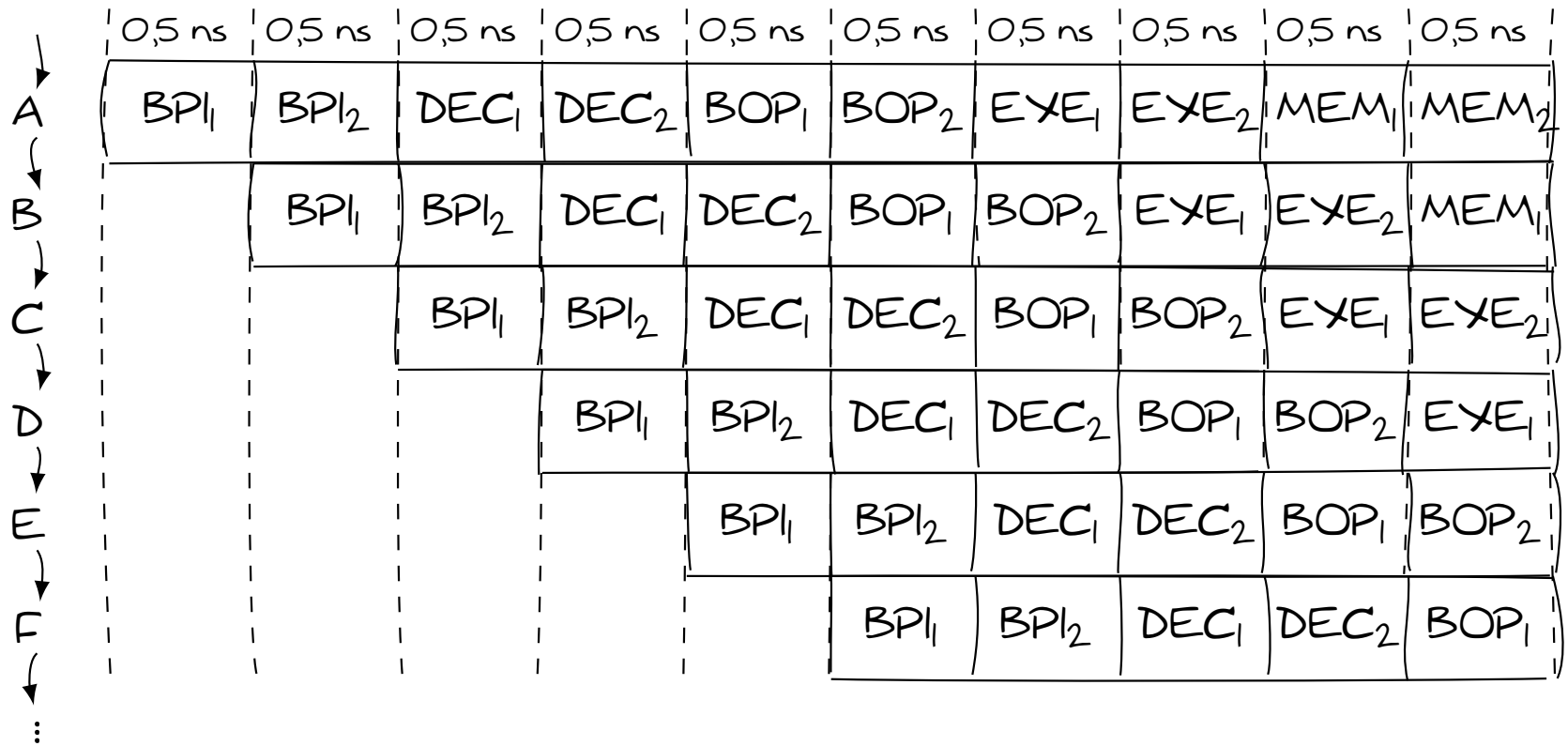
Introdução

- Implementação em *pipeline*
  - Não existe execução concorrente das instruções, mas existe o aumento da taxa de execução



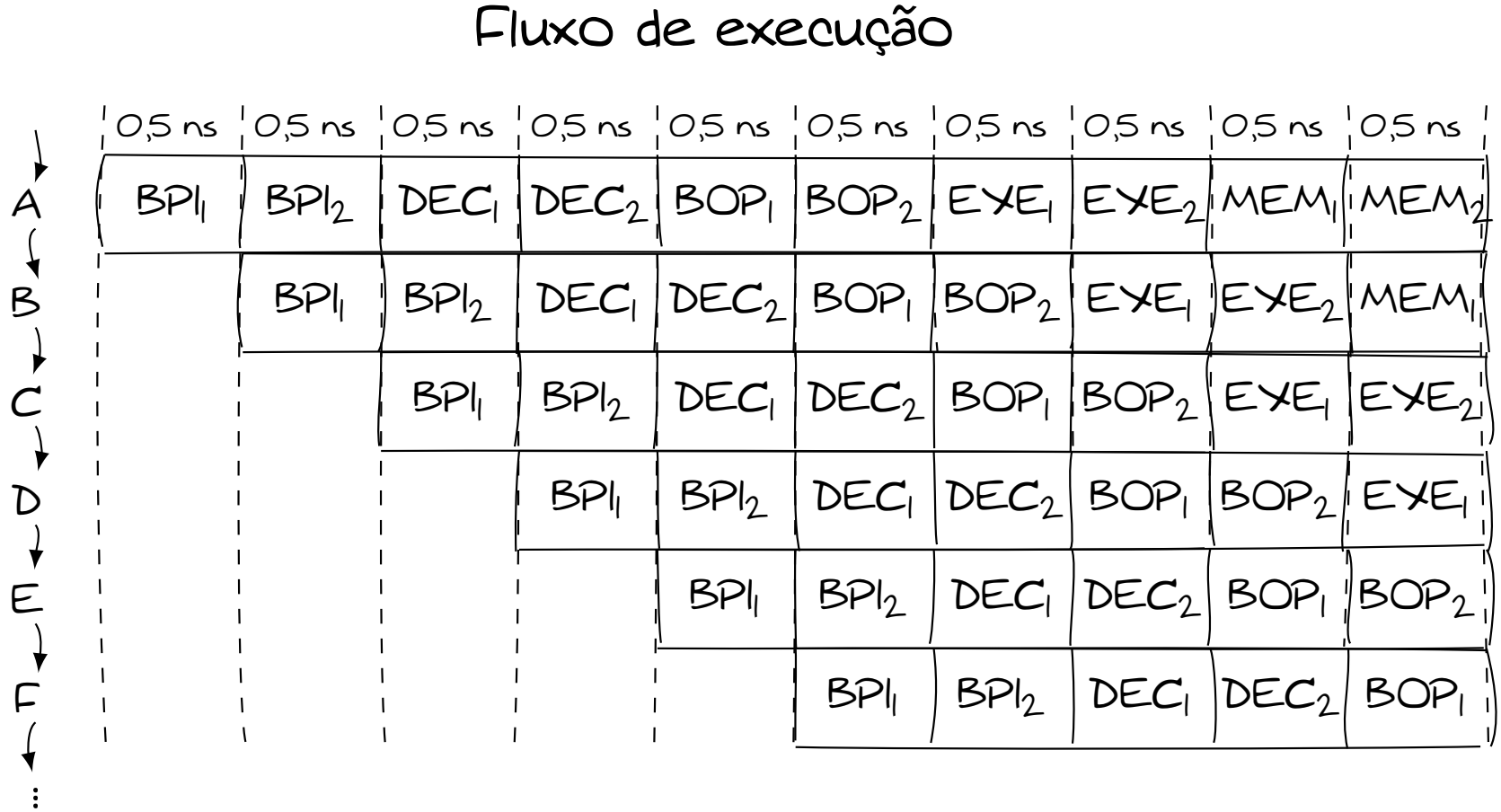
Introdução

- Implementação em *superpipeline*
  - São criados estágios menores com o dobro da frequência para aumentar a taxa de execução



Introdução

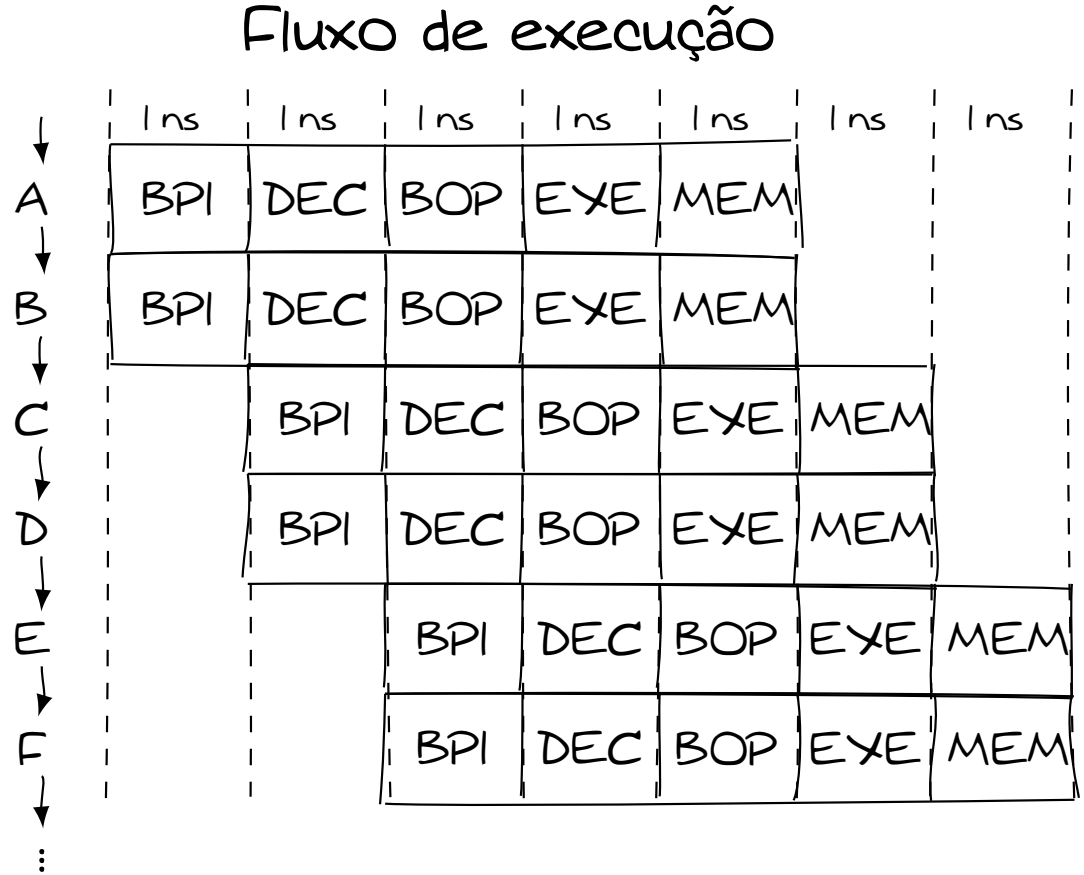
- Implementação em *superpipeline*
  - São criados estágios menores com o dobro da frequência para aumentar a taxa de execução



Taxa de até 2 instruções/ns (2 GHz, CPI = 1)

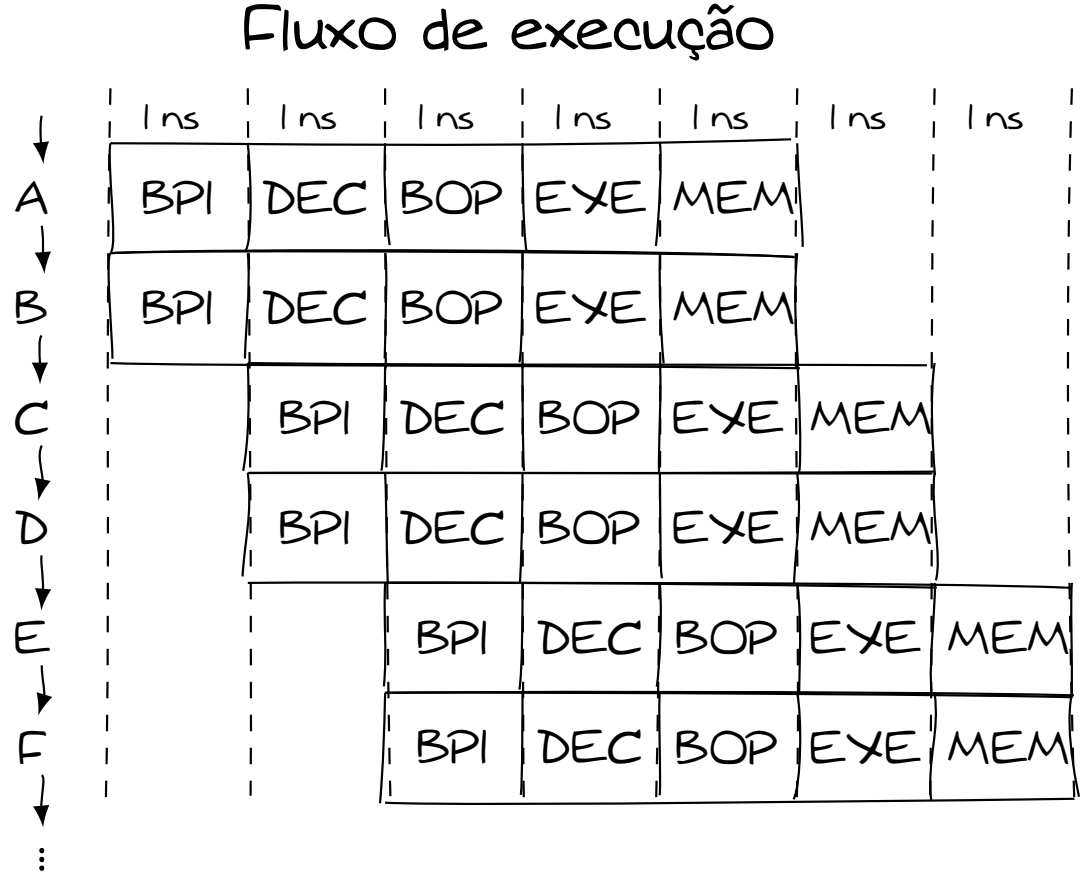
Introdução

- Implementação em *superescalar*
  - Os estágios do *pipeline* são replicados para que as instruções sejam executadas em paralelo



Introdução

- Implementação em *superescalar*
  - Os estágios do *pipeline* são replicados para que as instruções sejam executadas em paralelo



Taxa de até 2 instruções/ns (1 GHz, CPI = 0,5)

Introdução

- *Instruction-Level Parallelism* (ILP)
  - O paradigma *superescalar* está baseado no paralelismo em nível de instrução
    - O grau de paralelismo é definido pela quantidade média de instruções que podem ser executadas em paralelo nas unidades de processamento
    - Na avaliação do grau de paralelismo atingido pelo processador é utilizada a métrica de CPI

Introdução

- *Instruction-Level Parallelism* (ILP)
  - O paradigma *superescalar* está baseado no paralelismo em nível de instrução
    - O grau de paralelismo é definido pela quantidade média de instruções que podem ser executadas em paralelo nas unidades de processamento
    - Na avaliação do grau de paralelismo atingido pelo processador é utilizada a métrica de CPI
  - Para maximizar o paralelismo são necessárias otimizações combinadas de hardware e software
    - O hardware de controle procura aproveitar todas as unidades de processamento disponíveis
    - A compilação organiza estaticamente a sequência de instruções geradas para execução paralela

Introdução

- *Instruction-Level Parallelism* (ILP)
  - Paralelização do software

```
1  // Multiplicação escalar de vetor
2  void mult(int32_t k, int32_t V[], uint32_t n) {
3      // Índices
4      for(uint32_t i = 0; i < n; i++) {
5          // Multiplicação escalar
6          V[i] = k * V[i];
7      }
8  }
```

Introdução

- *Instruction-Level Parallelism* (ILP)
  - Paralelização do software

```
1 // Multiplicação escalar de vetor
2 void mult(int32_t k, int32_t V[], uint32_t n) {
3     // Índices
4     for(uint32_t i = 0; i < n; i++) {
5         // Multiplicação escalar
6         V[i] = k * V[i];
7     }
8 }
```

Todas as operações nos índices do vetor
podem ser realizadas em paralelo

Introdução

- *Instruction-Level Parallelism* (ILP)
  - Conflitos e dependências

```
1  // Sequência de Fibonacci
2  uint32_t fibonacci(uint32_t n) {
3      // Caso base
4      uint32_t r, tn2 = 0, tn1 = 1;
5      if(n <= 1) r = n;
6      // Cálculo sequencial
7      for(uint32_t i = 1; i < n; i++) {
8          r = tn2 + tn1;
9          tn2 = tn1;
10         tn1 = r;
11     }
12     // Retorno de resultado
13     return r;
14 }
```

Introdução

- *Instruction-Level Parallelism* (ILP)
  - Conflitos e dependências

```
1  // Sequência de Fibonacci
2  uint32_t fibonacci(uint32_t n) {
3      // Caso base
4      uint32_t r, tn2 = 0, tn1 = 1;
5      if(n <= 1) r = n;
6      // Cálculo sequencial
7      for(uint32_t i = 1; i < n; i++) {
8          r = tn2 + tn1;
9          tn2 = tn1;
10         tn1 = r;
11     }
12     // Retorno de resultado
13     return r;
14 }
```

A definição desta função de Fibonacci possui um comportamento inerentemente sequencial

Superescalar

- Conflitos e dependências de dado
  - A execução paralela de uma instrução é determinada pela análise de existência de conflito ou de dependência com outras instruções

Superescalar

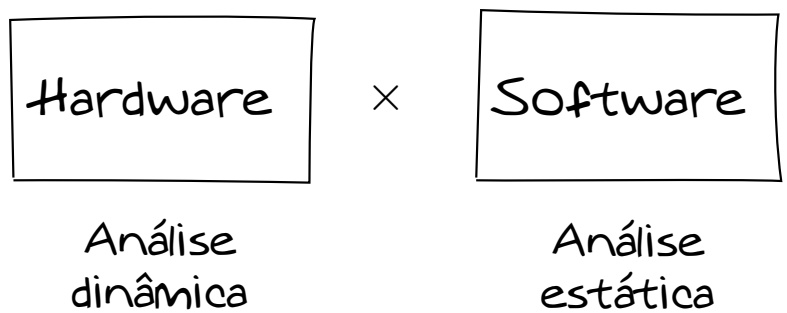
- Conflitos e dependências de dado
  - A execução paralela de uma instrução é determinada pela análise de existência de conflito ou de dependência com outras instruções
  - Duas ou mais instruções são paralelas quando a execução simultânea no *pipeline* não gera atrasos

Superescalar

- Conflitos e dependências de dado
  - A execução paralela de uma instrução é determinada pela análise de existência de conflito ou de dependência com outras instruções
  - Duas ou mais instruções são paralelas quando a execução simultânea no *pipeline* não gera atrasos
  - Quando existe um conflito ou dependência entre as instruções, sua execução deve ser sequencial

Superescalar

- Conflitos e dependências de dado
  - A execução paralela de uma instrução é determinada pela análise de existência de conflito ou de dependência com outras instruções
  - Duas ou mais instruções são paralelas quando a execução simultânea no *pipeline* não gera atrasos
  - Quando existe um conflito ou dependência entre as instruções, sua execução deve ser sequencial



Superescalar

- Dependência de dado
  - Direta: a instrução *i* produz um resultado que será utilizado pela instrução *j*

```
    1 | // R1 = 1
i -> 2 | addi r1, r0, 1
    3 | // R2 = R2 + R1
j -> 4 | add r2, r2, r1
```

Superescalar

- Dependência de dado
  - Direta: a instrução *i* produz um resultado que será utilizado pela instrução *j*

```
    1  // R1 = 1
i -> 2  addi r1, r0, 1
    3  // R2 = R2 + R1
j -> 4  add r2, r2, r1
```

  - Indireta: a instrução *k* depende do resultado gerado pela instrução *j* que também depende da instrução *i*

```
    1  // R1 = 1
i -> 2  addi r1, r0, 1
    3  // R2 = R2 + R1
j -> 4  add r2, r2, r1
    5  // R3 = R1 * R2
k -> 6  mul r3, r1, r2
```

Superescalar

- Dependência de dado
  - Direta: a instrução *i* produz um resultado que será utilizado pela instrução *j*

```
    1  // R1 = 1
i -> 2  addi r1, r0, 1
    3  // R2 = R2 + R1
j -> 4  add r2, r2, r1
```

  - Indireta: a instrução *k* depende do resultado gerado pela instrução *j* que também depende da instrução *i*

```
    1  // R1 = 1
i -> 2  addi r1, r0, 1
    3  // R2 = R2 + R1
j -> 4  add r2, r2, r1
    5  // R3 = R1 * R2
k -> 6  mul r3, r1, r2
```

A execução deve ser feita em ordem
e com sobreposição parcial no *pipeline*

Superescalar

- Dependência de dado
  - São inerentes aos programas, limitando a quantidade de operações que podem ser feitas em paralelo

Superescalar

- Dependência de dado
  - São inerentes aos programas, limitando a quantidade de operações que podem ser feitas em paralelo
  - A análise de conflitos e dependências permite explorar o potencial de paralelismo das instruções

Superescalar

- Dependência de dado
  - São inerentes aos programas, limitando a quantidade de operações que podem ser feitas em paralelo
  - A análise de conflitos e dependências permite explorar o potencial de paralelismo das instruções
  - Técnicas para tratar estas dependências
    - Manter a dependência e evitar o conflito de dados com adiantamento de dados no *pipeline*
    - Eliminar a dependência pela transformação do código por escalonamento estático ou dinâmico

Superescalar

- Dependência de nome
  - Anti-dependência: a instrução *j* escreve em um registrador ou endereço de memória que também é lido pela instrução *i*

```
    1   // R1 = R2
i -> 2  add r1, r0, r2
    3   // R2 = 1
j -> 4  addi r2, r0, 1
```

Superescalar

- Dependência de nome
  - Anti-dependência: a instrução *j* escreve em um registrador ou endereço de memória que também é lido pela instrução *i*

```
   1  // R1 = R2
i -> 2  add r1, r0, r2
   3  // R2 = 1
j -> 4  addi r2, r0, 1
```

  - Dependência de saída: duas instruções *i* e *j* escrevem no mesmo registrador ou endereço de memória

```
   1  // MEM[0x404] = R5
i -> 2  s32 [0x101], r5
   3  // MEM[0x404] = R7
j -> 4  s32 [0x101], r7
```

Superescalar

- Dependência de nome
  - Anti-dependência: a instrução *j* escreve em um registrador ou endereço de memória que também é lido pela instrução *i*

```
    1  // R1 = R2
i -> 2  add r1, r0, r2
    3  // R2 = 1
j -> 4  addi r2, r0, 1
```

  - Dependência de saída: duas instruções *i* e *j* escrevem no mesmo registrador ou endereço de memória

```
    1  // MEM[0x404] = R5
i -> 2  s32 [0x101], r5
    3  // MEM[0x404] = R7
j -> 4  s32 [0x101], r7
```

Não existe um fluxo de dados entre as instruções, mas compartilham a mesma entrada ou saída (nome)

Superescalar

- Dependência de nome
  - Como não existe a dependência de dados, as instruções podem ser executadas em paralelo

Superescalar

- Dependência de nome
  - Como não existe a dependência de dados, as instruções podem ser executadas em paralelo
  - Pode haver mudança de ordem nas instruções desde que os registradores sejam renomeados
    - Estática: realizada pelas etapas de otimização do compilador para geração de código
    - Dinâmica: aplicada durante a execução das instruções pelo hardware do processador

Superescalar

- Conflito de dados
  - São causados pela execução paralela de instruções com dependências de dado ou de nome

Superescalar

- Conflito de dados
  - São causados pela execução paralela de instruções com dependências de dado ou de nome
  - A ordem de execução do programa deve ser preservada na exploração do paralelismo

Superescalar

- Conflito de dados
  - São causados pela execução paralela de instruções com dependências de dado ou de nome
  - A ordem de execução do programa deve ser preservada na exploração do paralelismo
  - Tipos de conflitos de dados
    - *Read After Write* (RAW)
    - *Write After Read* (WAR)
    - *Write After Write* (WAW)

Superescalar

- Conflito de dados
  - *Read After Write* (RAW)
    - A instrução *j* realiza a leitura de um dado antes que que a instrução *i* tenha finalizado o processo de escrita, resultando na execução da instrução *j* com um valor anterior que já estava armazenado

```
    1 | // MEM[0x404] = R1
i -> 2 | s32 [0x101], r1
    3 | // R2 = MEM[0x404]
j -> 4 | l32 r2, [0x101]
```

Superescalar

- Conflito de dados
  - *Read After Write* (RAW)
    - A instrução *j* realiza a leitura de um dado antes que que a instrução *i* tenha finalizado o processo de escrita, resultando na execução da instrução *j* com um valor anterior que já estava armazenado

```
    1 | // MEM[0x404] = R1
i -> 2 | s32 [0x101], r1
    3 | // R2 = MEM[0x404]
j -> 4 | l32 r2, [0x101]
```

Este é o tipo de conflito mais comum (dependência verdadeira de dados)

Superescalar

- Conflito de dados
  - *Write After Read* (WAR)
    - A instrução *j* escreve em um operando da instrução *i* antes que ele seja carregado, causando a leitura incorreta do valor pela instrução *i*

```
    1 | // R1 = R1 * 2
i -> 2 | muli r1, r1, 2
    3 | // R1 = R2 / 3
j -> 4 | divi r1, r2, 3
```

Superescalar

- Conflito de dados
  - *Write After Read* (WAR)
    - A instrução *j* escreve em um operando da instrução *i* antes que ele seja carregado, causando a leitura incorreta do valor pela instrução *i*

```
    1 | // R1 = R1 * 2
i -> 2 | muli r1, r1, 2
    3 | // R1 = R2 / 3
j -> 4 | divi r1, r2, 3
```

Este conflito é causado pela anti-dependência e pela execução fora de ordem no *pipeline*

Superescalar

- Conflito de dados
  - *Write After Write* (WAW)
    - A instrução *j* possui um um operando de saída em comum com a instrução *i*, deixando o valor calculado pela instrução *i*, ao invés do resultado da instrução *j*

```
    1 // R1 = R2 * 5
i -> 2 muli r1, r2, 5
    3 // R1 = R3 / 8
j -> 4 divi r1, r3, 8
```

Superescalar

- Conflito de dados
  - *Write After Write* (WAW)
    - A instrução *j* possui um um operando de saída em comum com a instrução *i*, deixando o valor calculado pela instrução *i*, ao invés do resultado da instrução *j*

```
    1 | // R1 = R2 * 5
i -> 2 | muli r1, r2, 5
    3 | // R1 = R3 / 8
j -> 4 | divi r1, r3, 8
```

Este conflito é decorrente da dependência de saída e só ocorre quando o *pipeline* permite a escrita em mais de um estágio ou a execução fora de ordem das instruções

Superescalar

- Dependência de controle
  - É o resultado da execução de instruções em desvios

```
1  // Função principal
2  int main() {
3      // ID do aluno
4      uint32_t id = rand() * rand();
5      // Quantidade de faltas acumuladas
6      uint8_t faltas = rand() % 61;
7      // Se faltas > 18 horas, então reprovar
8      if(faltas > 18) reprovar(id);
9      // Se não, checar nota
10     else checar_nota(id);
11     // Retorno sem erros
12     return 0;
13 }
```

Superescalar

- Dependência de controle
  - É o resultado da execução de instruções em desvios

```
1  // Função principal
2  int main() {
3      // ID do aluno
4      uint32_t id = rand() * rand();
5      // Quantidade de faltas acumuladas
6      uint8_t faltas = rand() % 61;
7      // Se faltas > 18 horas, então reprovar
8      if(faltas > 18) reprovar(id);
9      // Se não, checar nota
10     else checar_nota(id);
11     // Retorno sem erros
12     return 0;
13 }
```

Superescalar

- Dependência de controle
  - É o resultado da execução de instruções em desvios

```
1  // Função principal
2  main:
3      // r1 = id, r2 = faltas
4      l32 r1, [0x40]
5      l8 r2, [0x107]
6      // faltas ? 18
7      cmpi r2, 18
8      bgt V
9          // faltas <= 18
10     F:  bun checar_nota
11         bun 1
12         // faltas > 18
13     V:  bun reprovar
14     // Fim
15     int 0
```

Superescalar

- Dependência de controle
  - É o resultado da execução de instruções em desvios

```
1  // Função principal
2  main:
3      // r1 = id, r2 = faltas
4      l32 r1, [0x40]
5      l8 r2, [0x107]
6      // faltas ? 18
7      cmpi r2, 18
8      bgt V
9          // faltas <= 18
10     F:  bun checar_nota
11         bun 1
12         // faltas > 18
13     V:  bun reprovar
14     // Fim
15     int 0
```

Superescalar

- Dependência de controle
  - É o resultado da execução de instruções em desvios

```
1  // Função principal
2  main:
3      // r1 = id, r2 = faltas
4      l32 r1, [0x40]
5      l8 r2, [0x107]
6      // faltas ? 18
7      cmpi r2, 18
8      bgt V
9          // faltas <= 18
10     F:  bun checar_nota
11         bun 1
12         // faltas > 18
13     V:  bun reprovar
14     // Fim
15     int 0
```

Superescalar

- Dependência de controle
  - É o resultado da execução de instruções em desvios

```
1  // Função principal
2  main:
3      // r1 = id, r2 = faltas
4      l32 r1, [0x40]
5      l8 r2, [0x107]
6      // faltas ? 18
7      cmpi r2, 18
8      bgt V
9          // faltas <= 18
10     F:  bun checar_nota
11         bun 1
12         // faltas > 18
13     V:  bun reprovar
14     // Fim
15     int 0
```

Execução incorreta por erro na predição do desvio

Superescalar

- Dependência de controle
  - É o resultado da execução de instruções em desvios

```
1  // Função principal
2  main:
3      // r1 = id, r2 = faltas
4      l32 r1, [0x40]
5      l8 r2, [0x107]
6      // faltas ? 18
7      cmpi r2, 18
8      bgt V
9          // faltas <= 18
10     F:  bun checar_nota
11         bun 1
12         // faltas > 18
13     V:  bun reprovar
14     // Fim
15     int 0
```

Execução incorreta por erro na predição do desvio

Superescalar

- Comportamento de exceção
  - As mudanças na ordem de execução das instruções não devem alterar o comportamento das exceções

```
1  // Função principal
2  main:
3      mov r1, 0
4      mov r2, 1
5      cmpi r3, 1
6      beq 3
7      divi r4, r4, 0
8      addi r4, r4, 1
9      subi r5, r4, 1
10     int 0
```

Superescalar

- Comportamento de exceção
  - As mudanças na ordem de execução das instruções não devem alterar o comportamento das exceções

```
1  // Função principal
2  main:
3      mov r1, 0
4      mov r2, 1
5      cmpi r3, 1
6      beq 3
7      divi r4, r4, 0
8      addi r4, r4, 1
9      subi r5, r4, 1
10     int 0
```

Superescalar

- Comportamento de exceção
  - As mudanças na ordem de execução das instruções não devem alterar o comportamento das exceções

```
1  // Função principal
2  main:
3      mov r1, 0
4      mov r2, 1
5      cmpi r3, 1
6      beq 3
7      divi r4, r4, 0
8      addi r4, r4, 1
9      subi r5, r4, 1
10     int 0
```

Superescalar

- Comportamento de exceção
  - As mudanças na ordem de execução das instruções não devem alterar o comportamento das exceções

```
1  // Função principal
2  main:
3      mov r1, 0
4      mov r2, 1
5      cmpi r3, 1
6      beq 3
7      divi r4, r4, 0
8      addi r4, r4, 1
9      subi r5, r4, 1
10     int 0
```

Superescalar

- Comportamento de exceção
  - As mudanças na ordem de execução das instruções não devem alterar o comportamento das exceções

```
1  // Função principal
2  main:
3       mov r1, 0
4       mov r2, 1
5       cmpi r3, 1
6       beq 3
7       divi r4, r4, 0
8       addi r4, r4, 1
9       subi r5, r4, 1
10      int 0
```

É feito o cancelamento da instrução após a exceção

Superescalar

- Políticas de emissão e finalização de instruções
  - Emissão de instruções (*issue*)
    - É o processo de busca e decodificação de instruções para serem executadas pelo processador
    - Pode ser realizada em ordem, seguindo a sequência das instruções, ou fora de ordem, armazenando as instruções e as executando fora de sequência

Superescalar

- Políticas de emissão e finalização de instruções
  - Emissão de instruções (*issue*)
    - É o processo de busca e decodificação de instruções para serem executadas pelo processador
    - Pode ser realizada em ordem, seguindo a sequência das instruções, ou fora de ordem, armazenando as instruções e as executando fora de sequência
  - Finalização de operações (*commit*)
    - Ocorre quando uma instrução gera um resultado, modificando os dados de registradores ou da memória
    - Para maximizar o desempenho e tratar conflitos, os dados podem ser armazenados fora de ordem

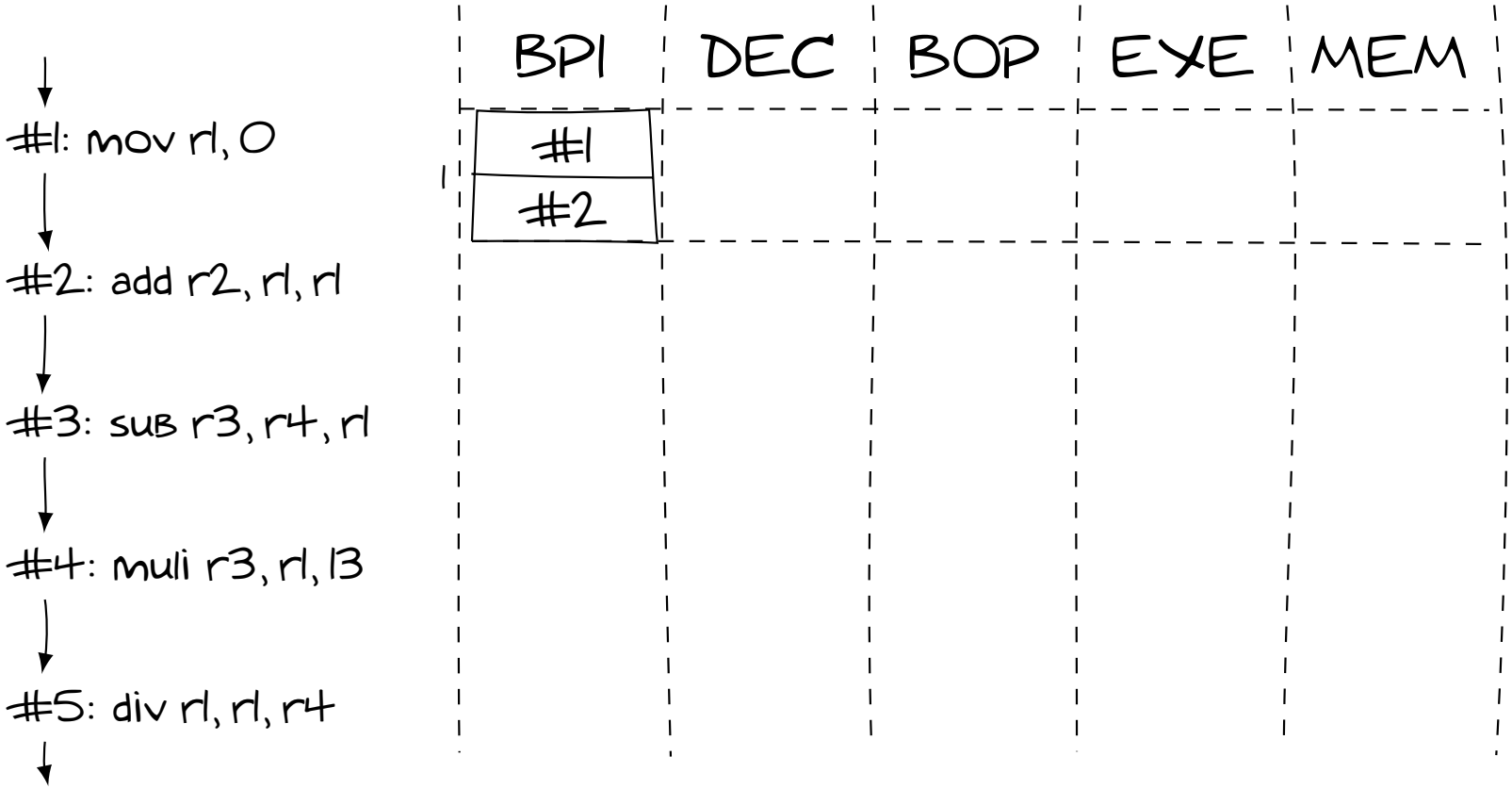
Superescalar

- Políticas de emissão e finalização de instruções
  - Sequência de instruções para execução paralela em um processador com com estágios replicados

```
1 mov r1, 0
2 add r2, r1, r1
3 sub r3, r4, r1
4 muli r3, r1, 13
5 div r1, r1, r4
```

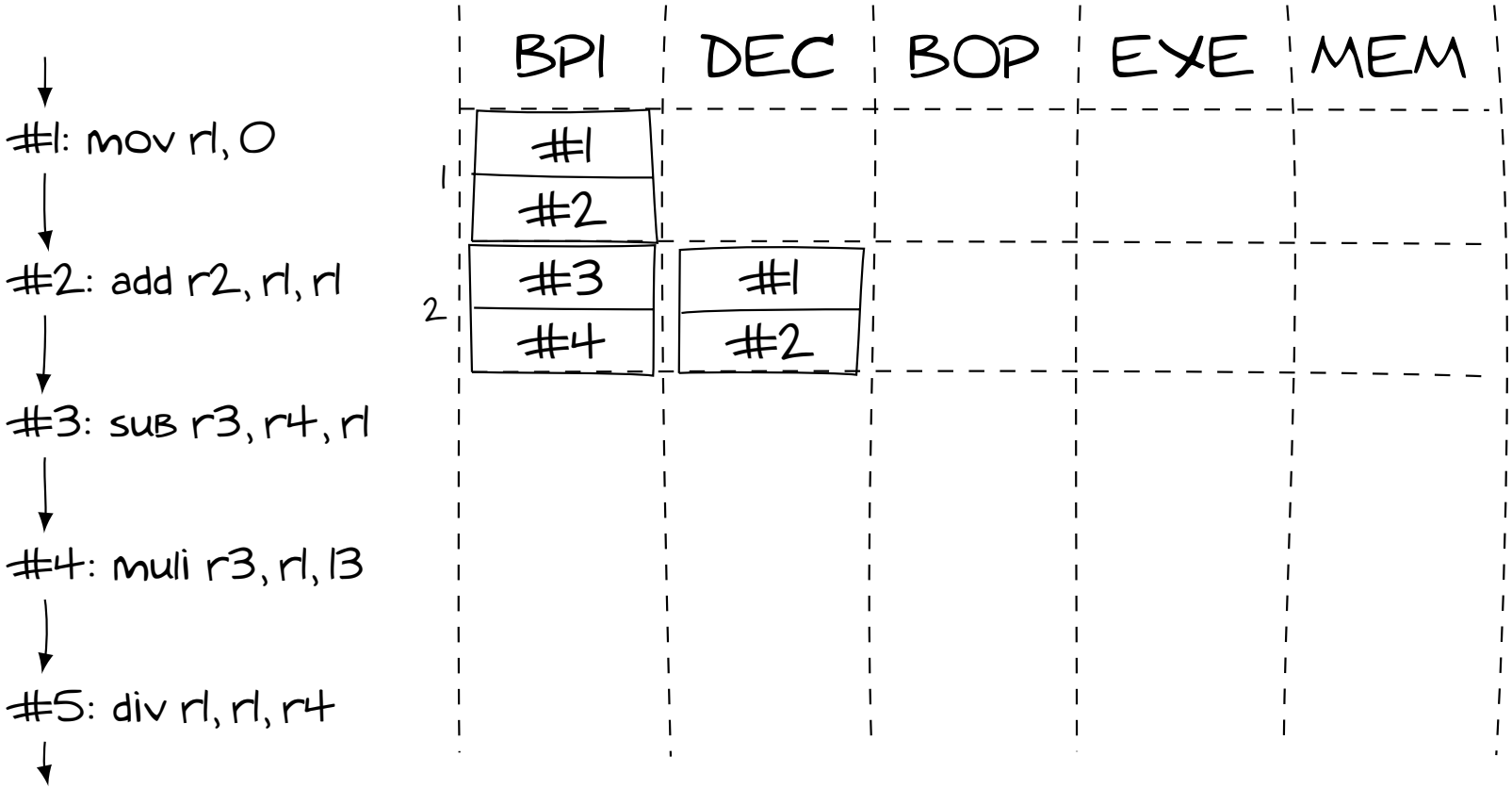
Superescalar

- Emissão e finalização em ordem
  - As instruções são buscadas em ordem, mantendo o comportamento sequencial do software



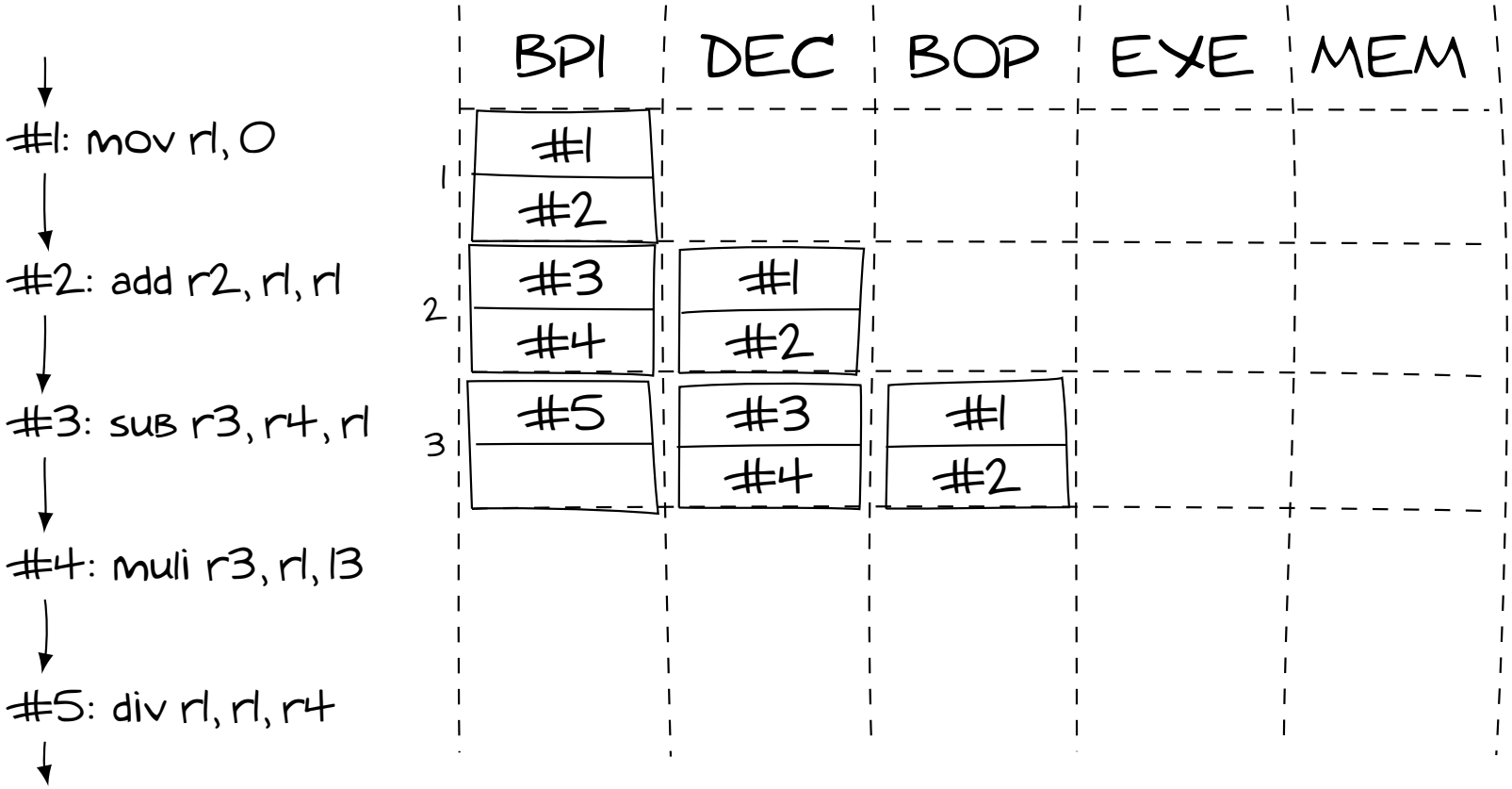
Superescalar

- Emissão e finalização em ordem
  - As instruções são buscadas em ordem, mantendo o comportamento sequencial do software



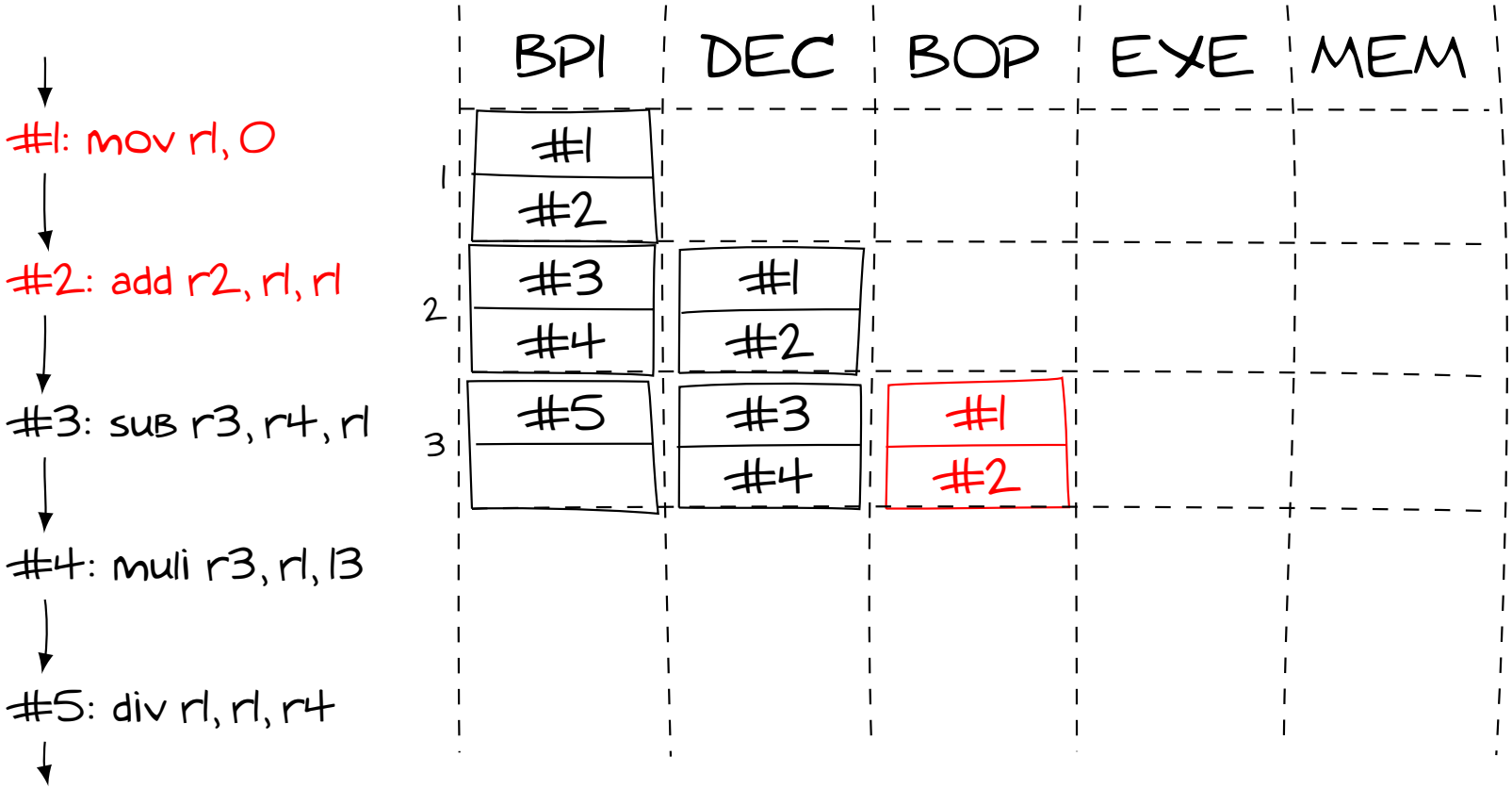
Superescalar

- Emissão e finalização em ordem
  - As instruções são buscadas em ordem, mantendo o comportamento sequencial do software



Superescalar

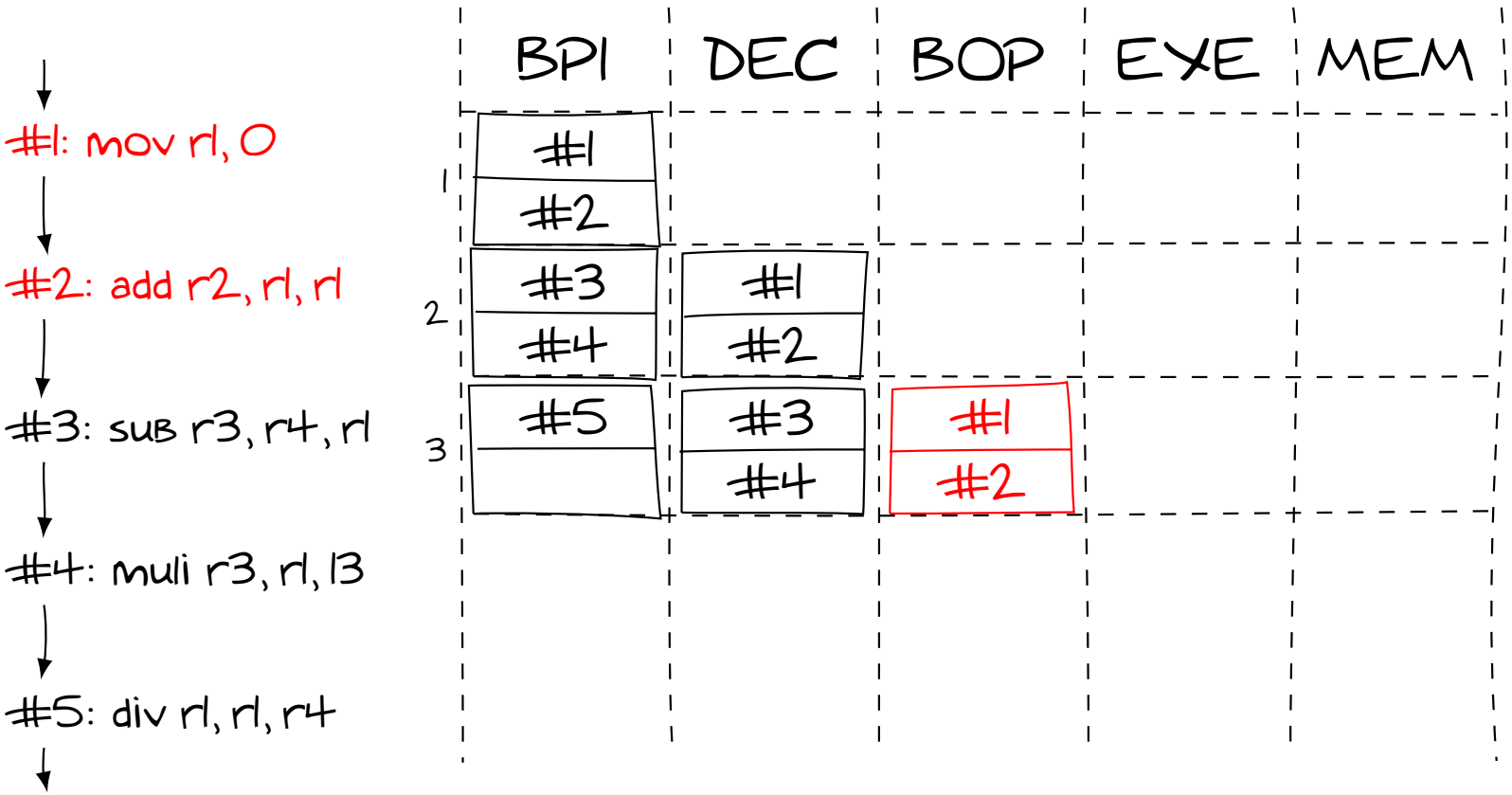
- Emissão e finalização em ordem
  - As instruções são buscadas em ordem, mantendo o comportamento sequencial do software



A instrução #1 tem como saída R1 que é entrada para instrução #2 (conflito RAW)

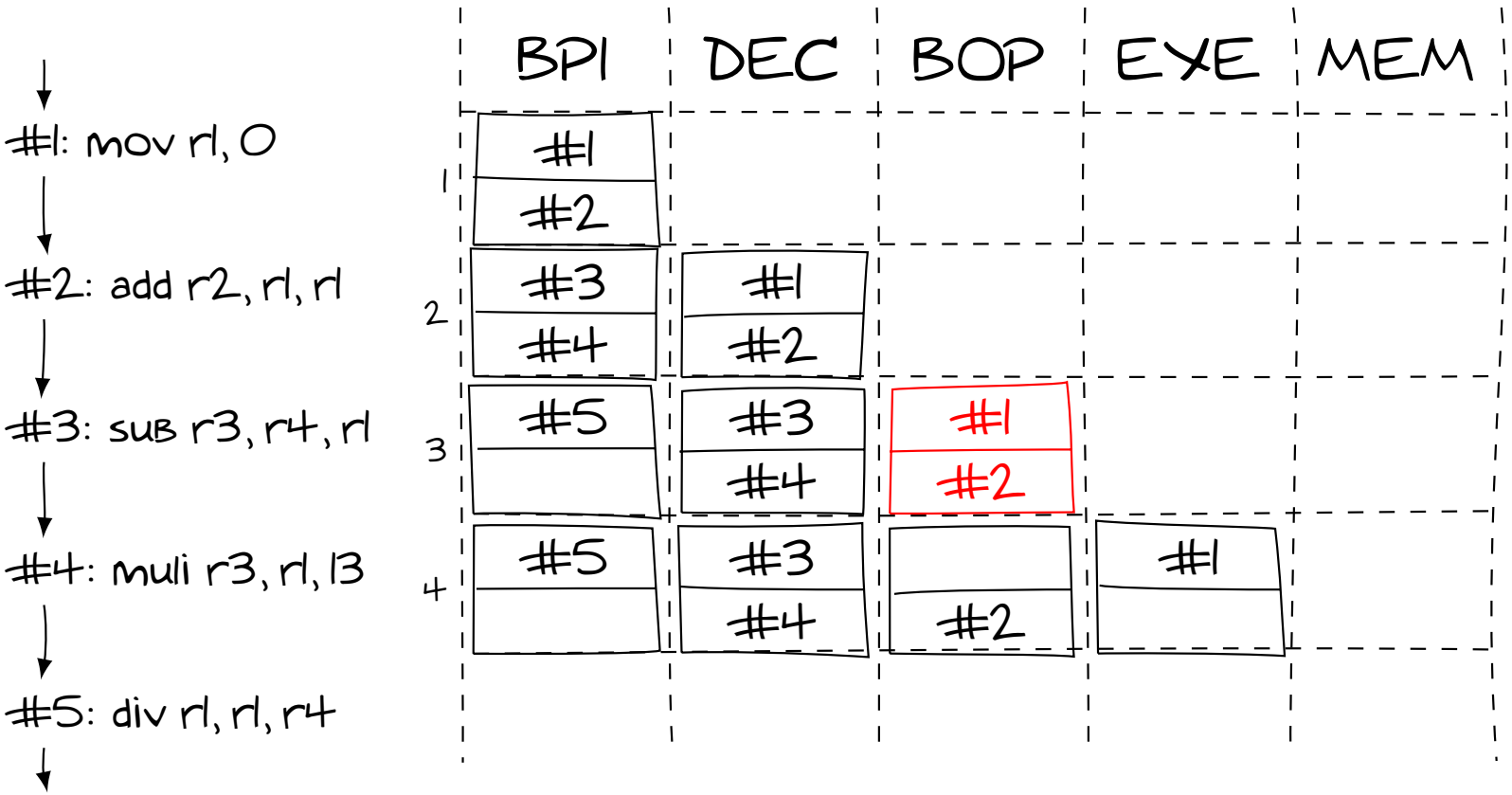
Superescalar

- Emissão e finalização em ordem
  - As instruções são buscadas em ordem, mantendo o comportamento sequencial do software



Superescalar

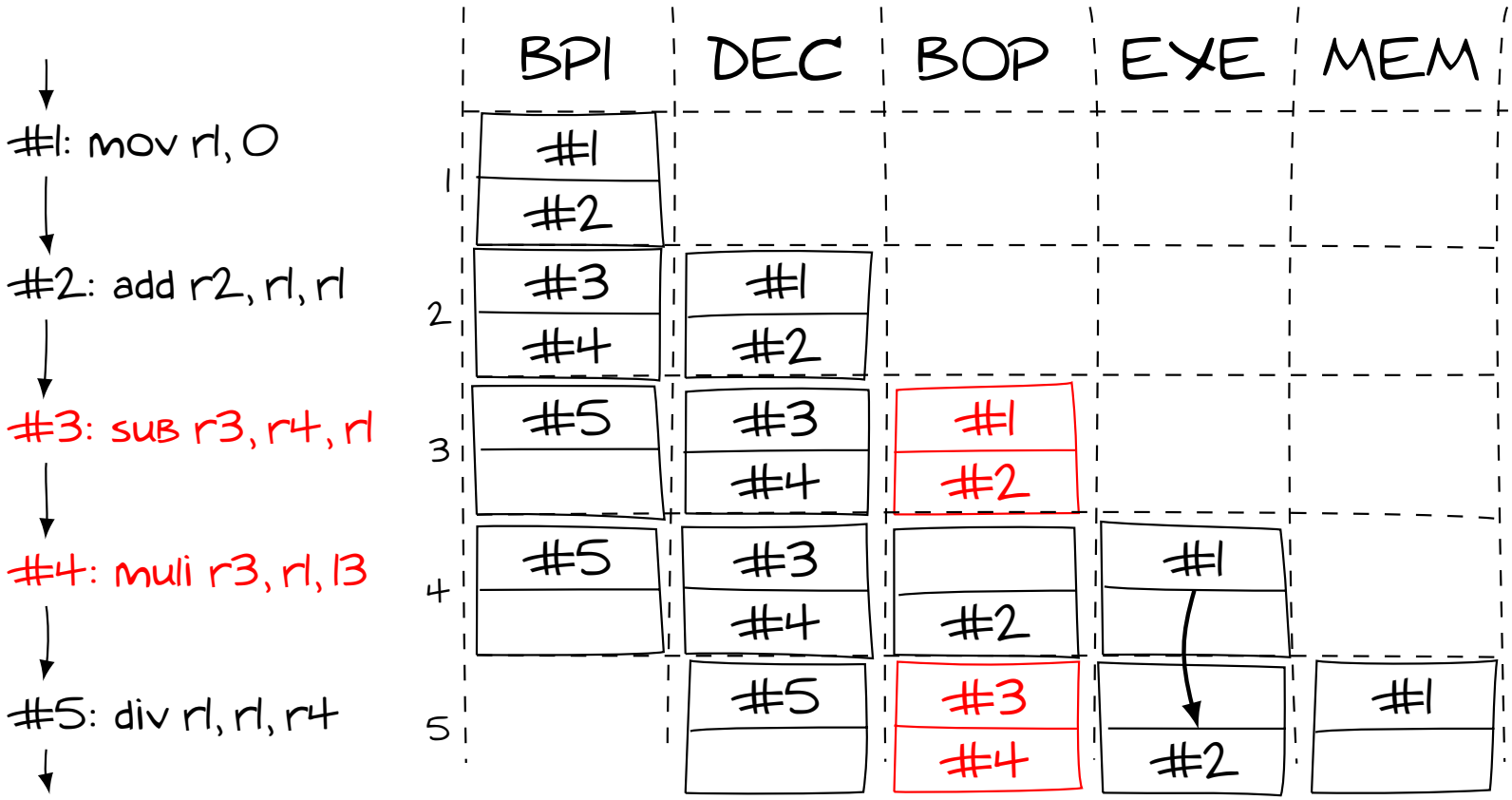
- Emissão e finalização em ordem
  - As instruções são buscadas em ordem, mantendo o comportamento sequencial do software



A instrução #1 segue para o próximo estágio, enquanto que instrução #2 fica em espera (stalled)

Superescalar

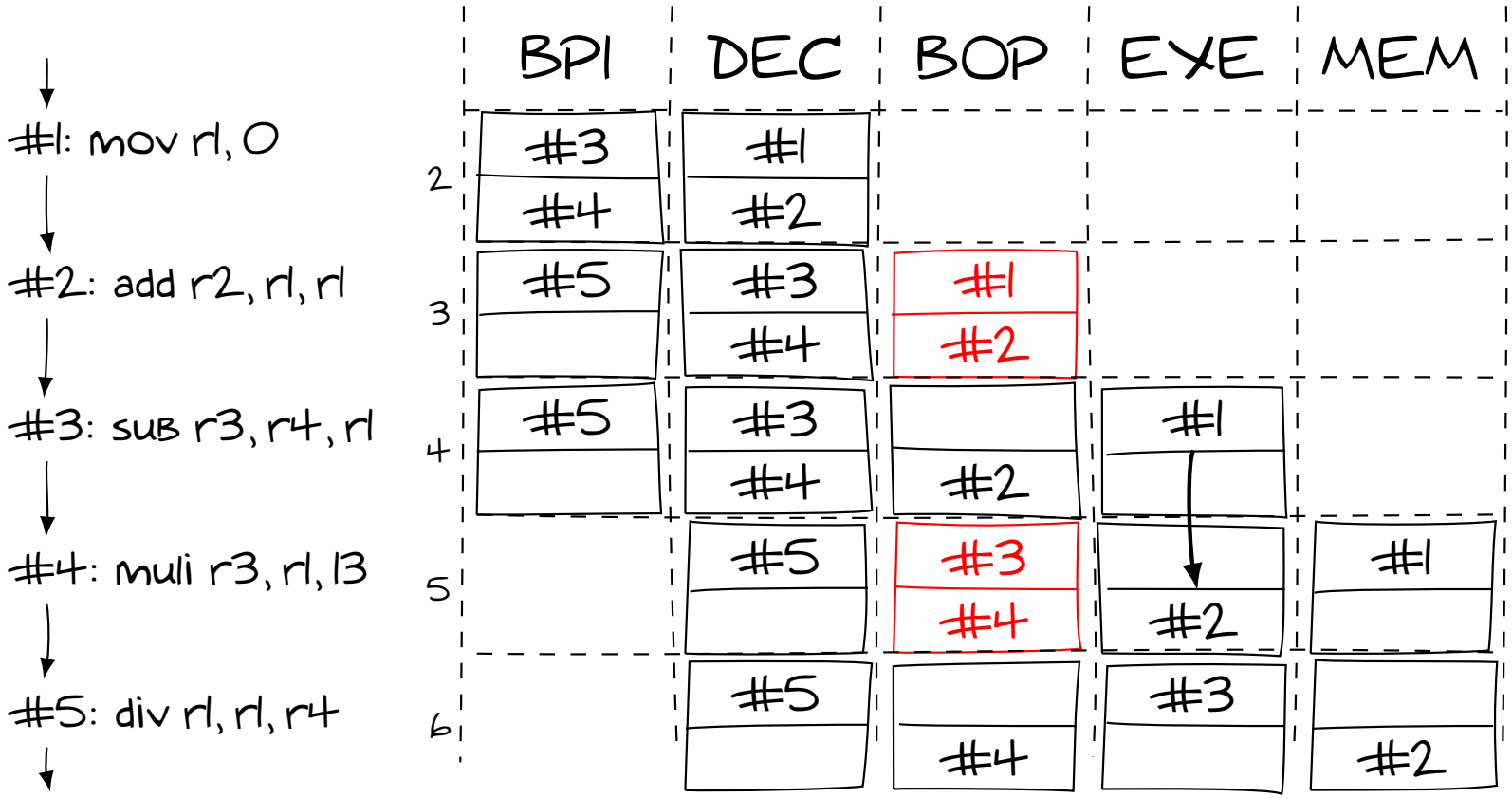
- Emissão e finalização em ordem
  - As instruções são buscadas em ordem, mantendo o comportamento sequencial do software



É feito o adiantamento para a instrução #2 e é detectado um conflito WAW entre #3 e #4

Superescalar

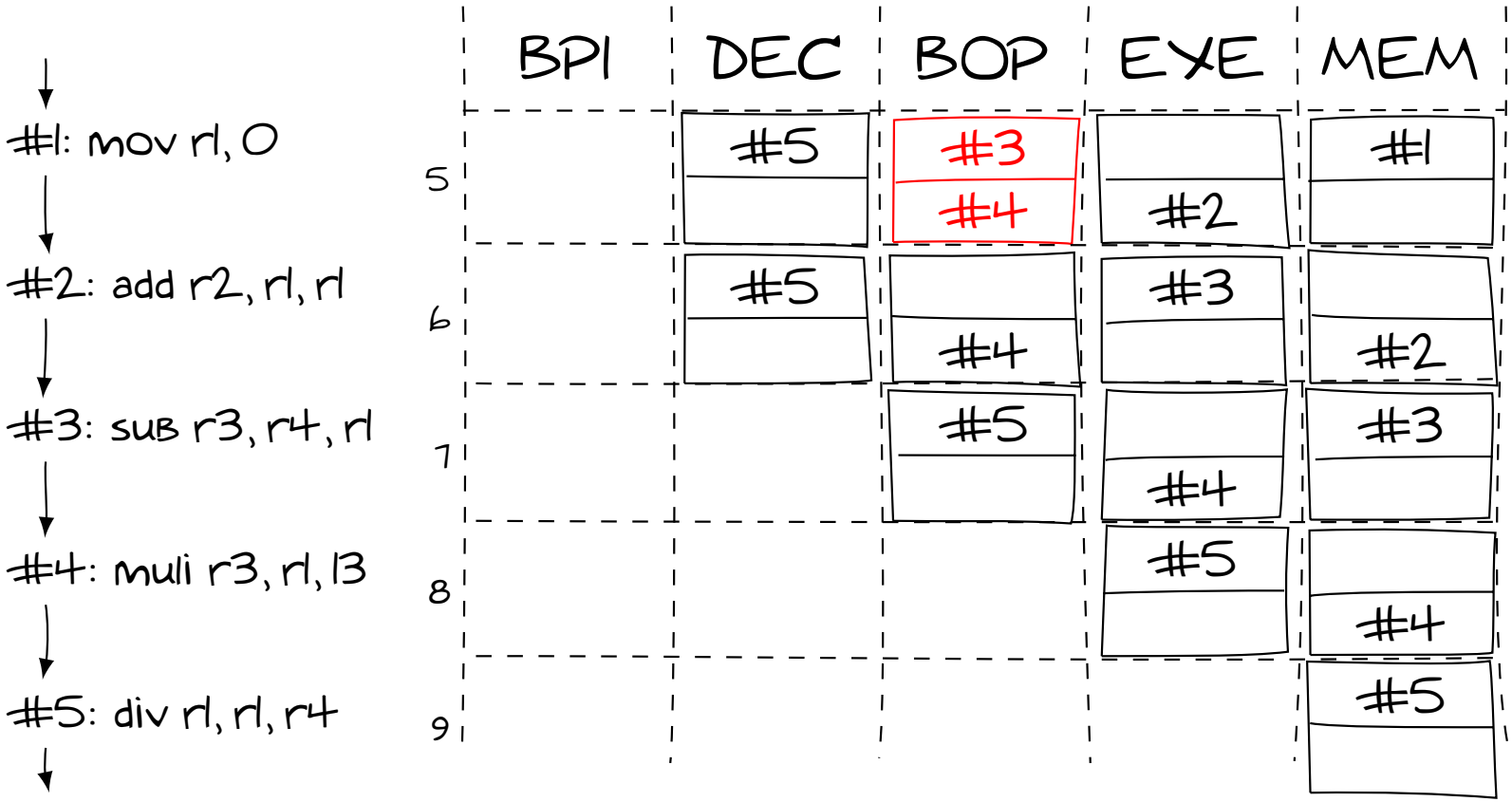
- Emissão e finalização em ordem
  - As instruções são buscadas em ordem, mantendo o comportamento sequencial do software



Para garantir a ordem de execução, instrução #4 fica em espera (stalled)

Superescalar

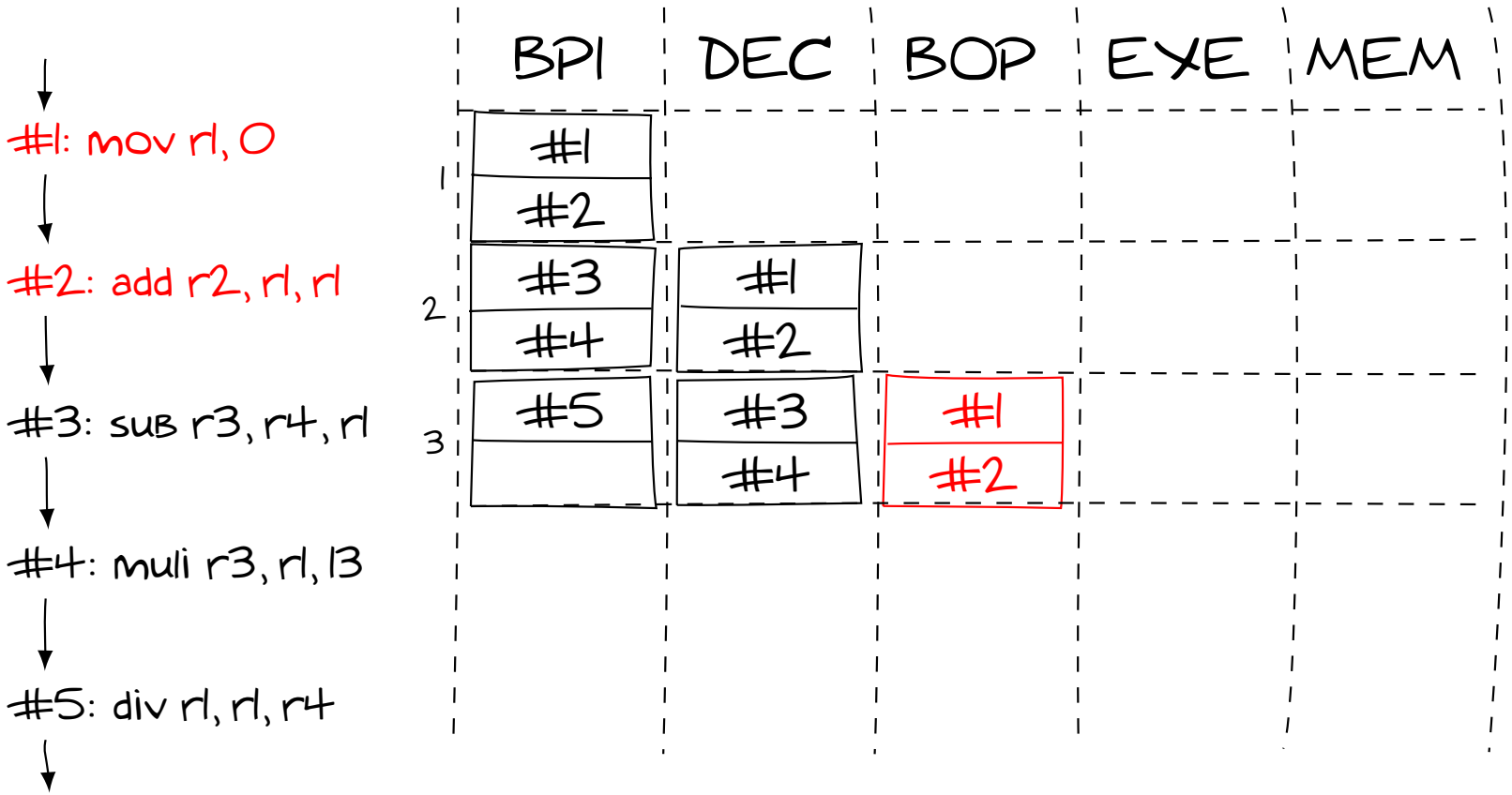
- Emissão e finalização em ordem
  - As instruções são buscadas em ordem, mantendo o comportamento sequencial do software



A execução é finalizada em 9 ciclos

Superescalar

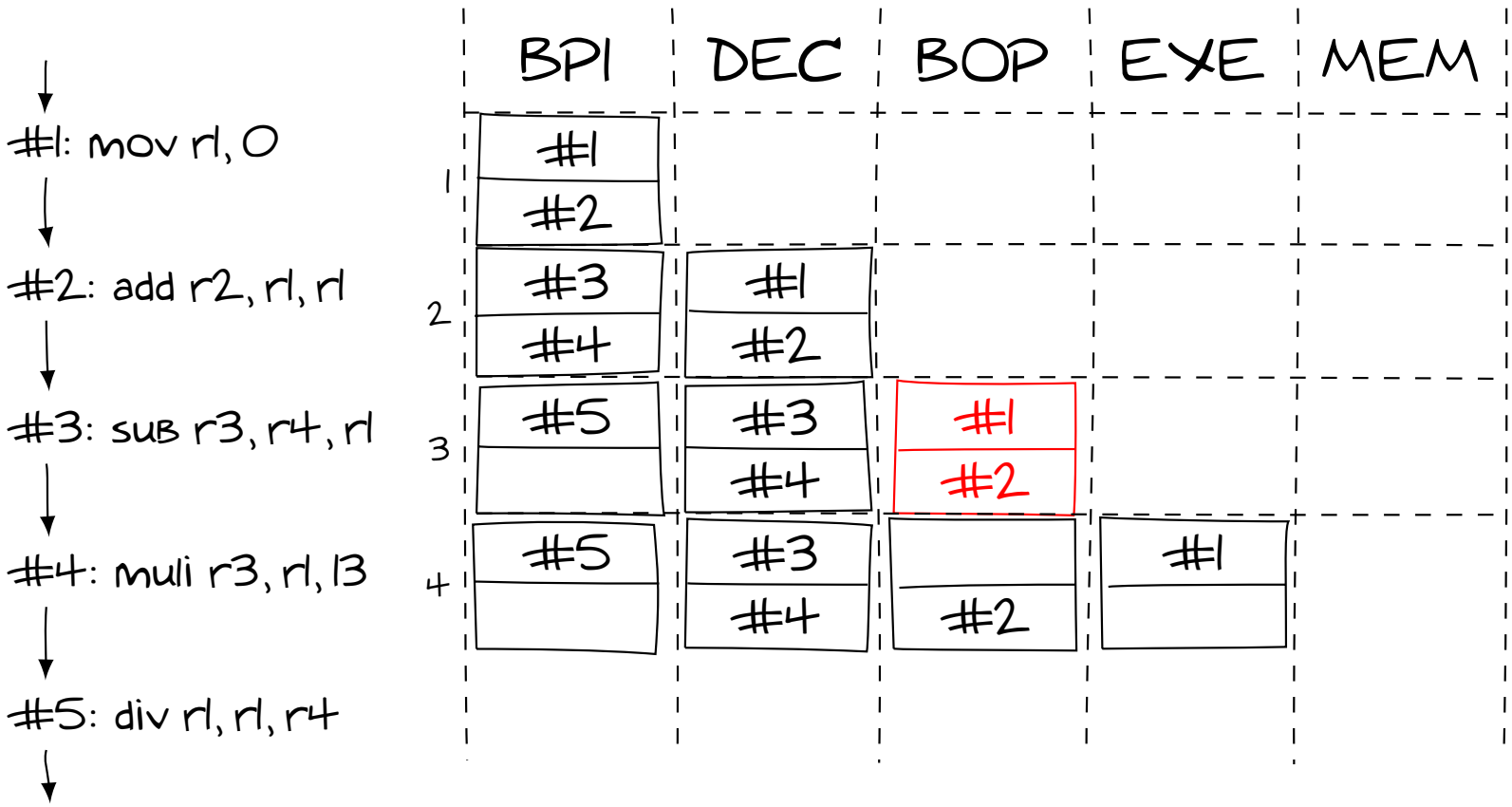
- Emissão em ordem com finalização fora de ordem
  - As instruções são executadas em ordem e o armazenamento dos resultados é feito fora de ordem



A instrução #1 tem como saída R1 que é entrada para instrução #2 (conflito RAW)

Superescalar

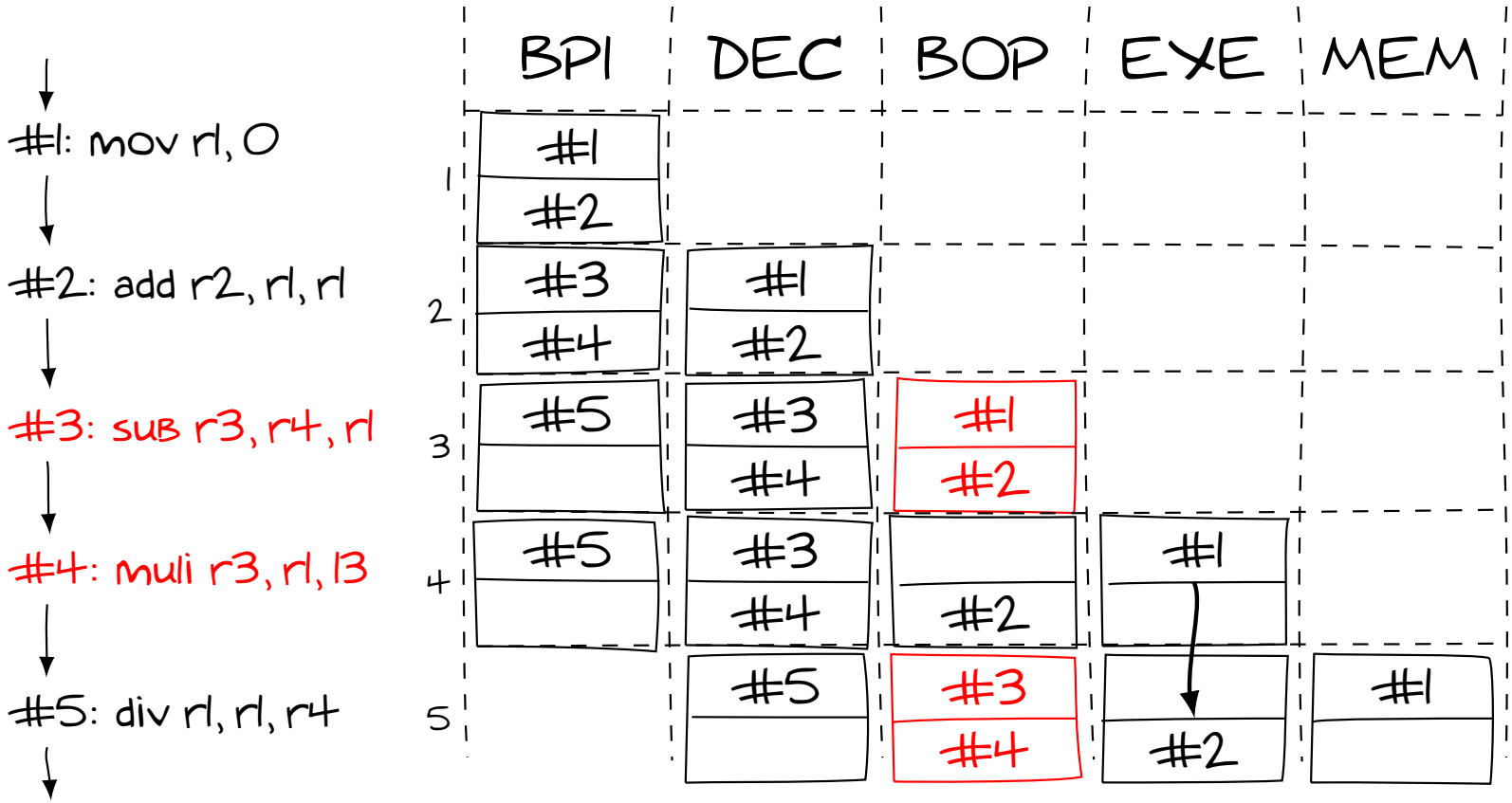
- Emissão em ordem com finalização fora de ordem
  - As instruções são executadas em ordem e o armazenamento dos resultados é feito fora de ordem



A instrução #1 segue para o próximo estágio, enquanto que instrução #2 fica em espera (stalled)

Superescalar

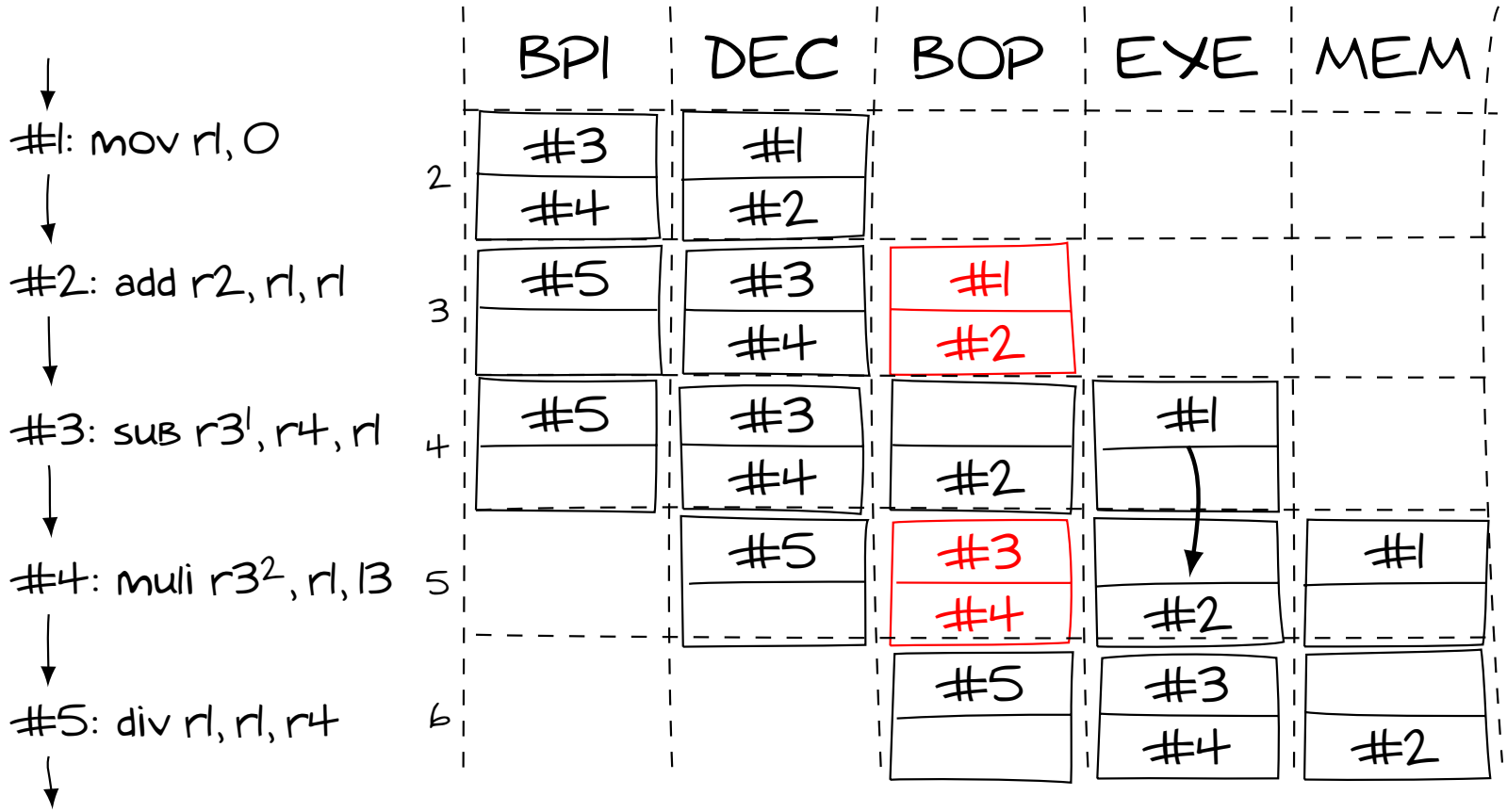
- Emissão em ordem com finalização fora de ordem
  - As instruções são executadas em ordem e o armazenamento dos resultados é feito fora de ordem



É feito o adiantamento para a instrução #2 e é detectado um conflito WAW entre #3 e #4

Superescalar

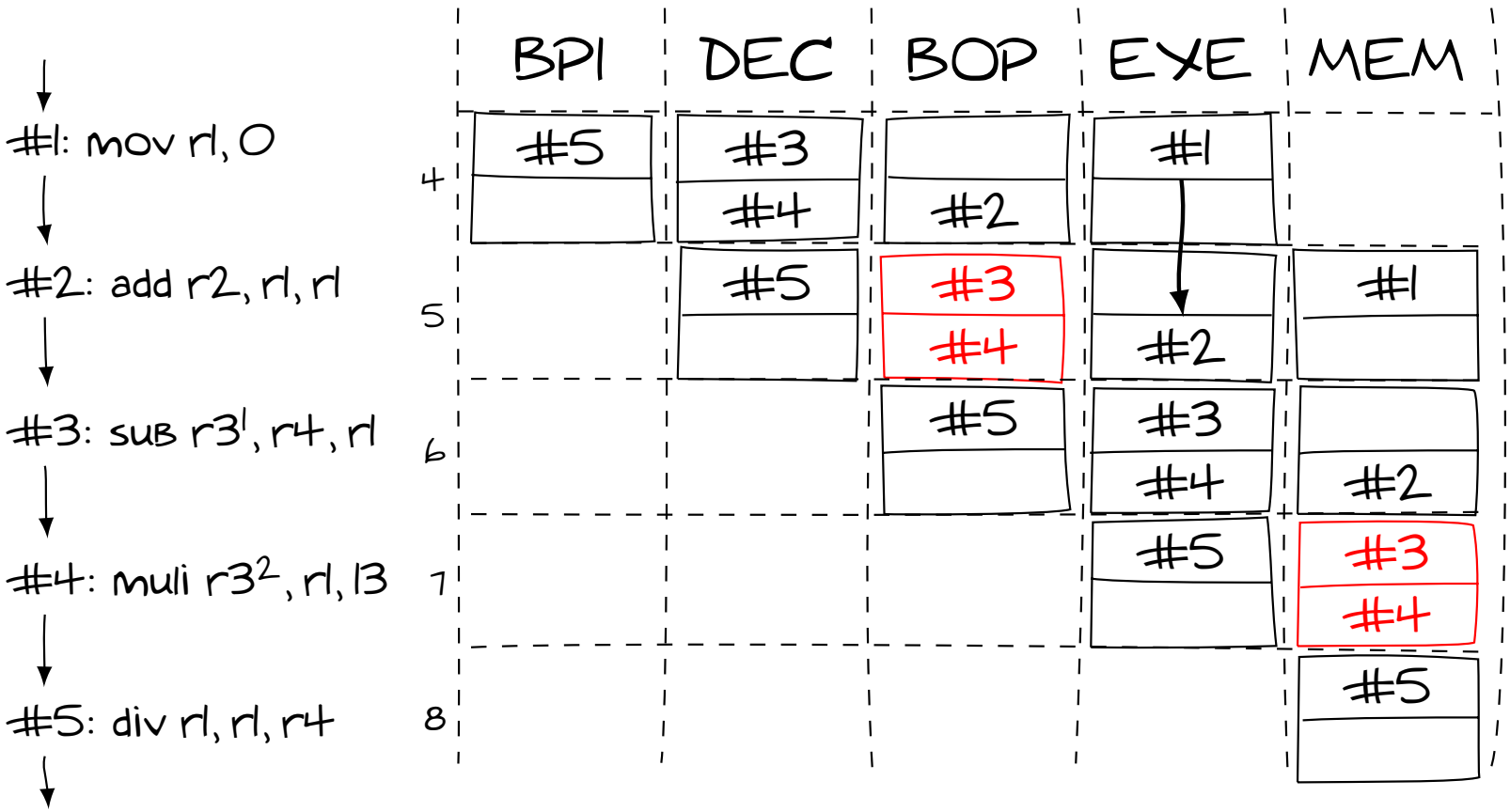
- Emissão em ordem com finalização fora de ordem
  - As instruções são executadas em ordem e o armazenamento dos resultados é feito fora de ordem



Ao invés de sequencializar #3 e #4,
é feito o renomeamento de R3 (WAW)

Superescalar

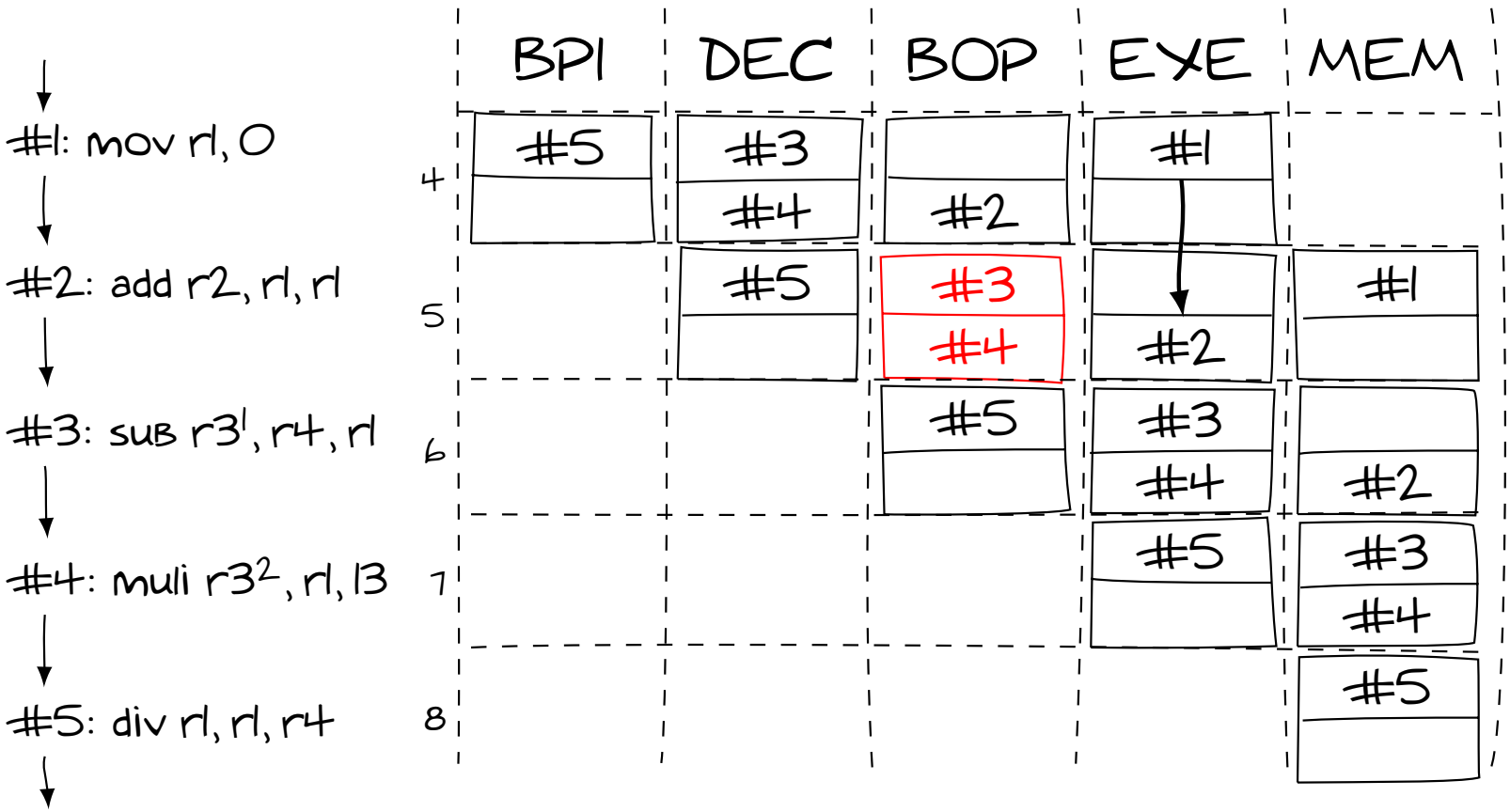
- Emissão em ordem com finalização fora de ordem
  - As instruções são executadas em ordem e o armazenamento dos resultados é feito fora de ordem



R3 recebe o resultado de $R3^2$

Superescalar

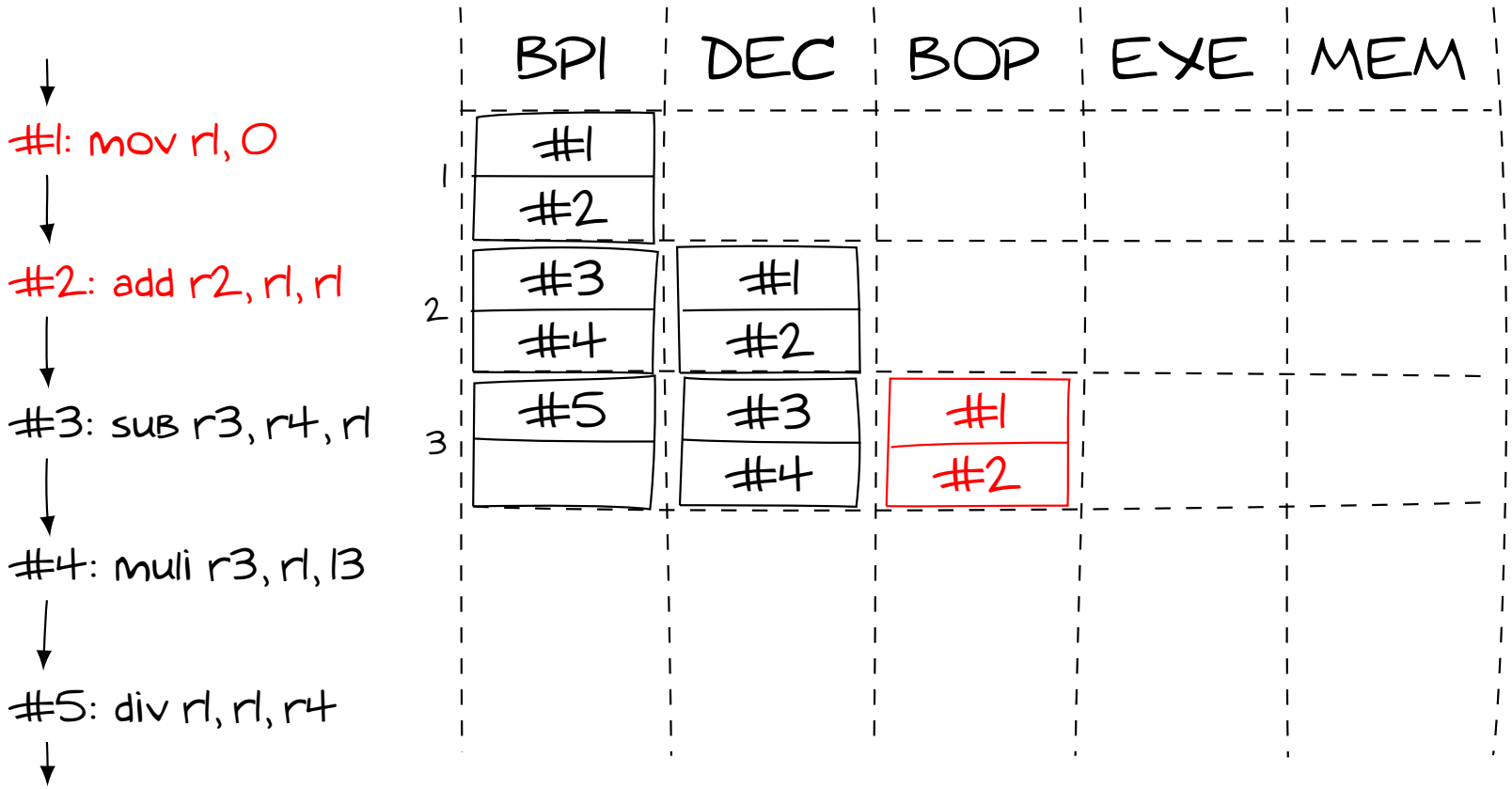
- Emissão em ordem com finalização fora de ordem
  - As instruções são executadas em ordem e o armazenamento dos resultados é feito fora de ordem



A execução é finalizada em 8 ciclos

Superescalar

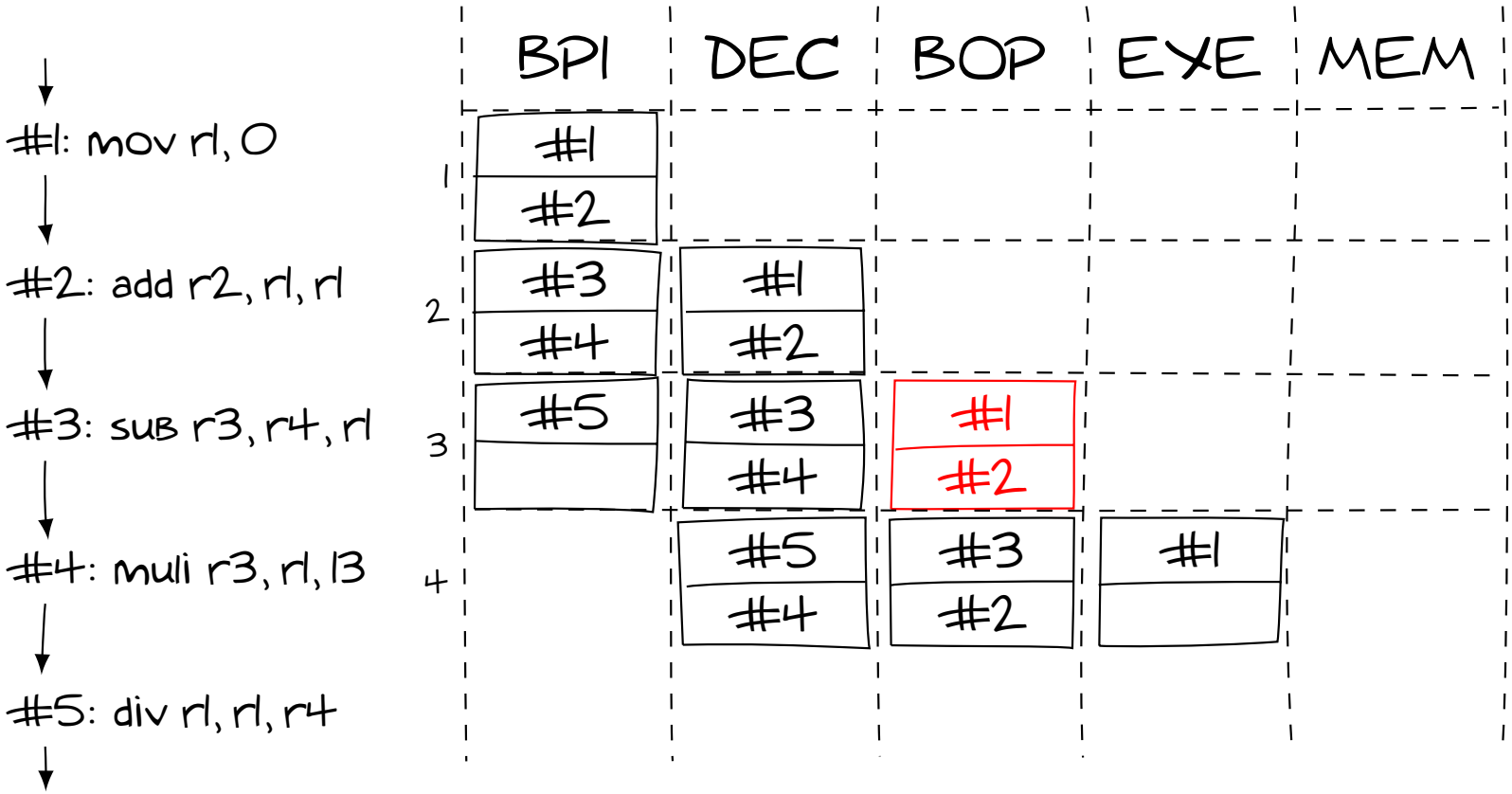
- Emissão e finalização fora de ordem
  - São utilizadas unidades de reserva para suportar a execução fora de ordem das instruções



A instrução #1 tem como saída R1 que é entrada para instrução #2 (conflito RAW)

Superescalar

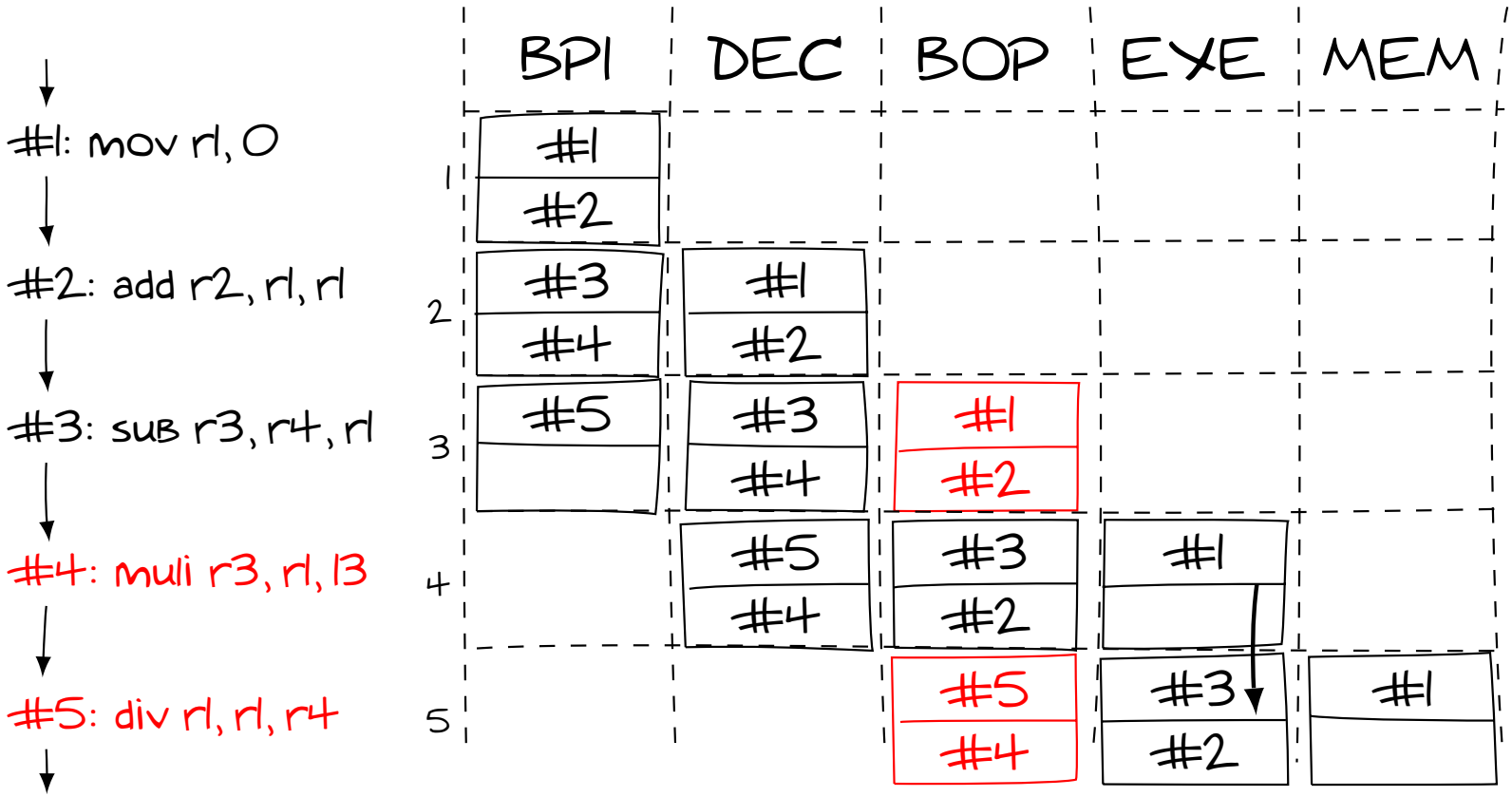
- Emissão e finalização fora de ordem
  - São utilizadas unidades de reserva para suportar a execução fora de ordem das instruções



A instrução #2 fica em espera (stalled)
e as instruções #3 e #5 são emitidas fora de ordem

Superescalar

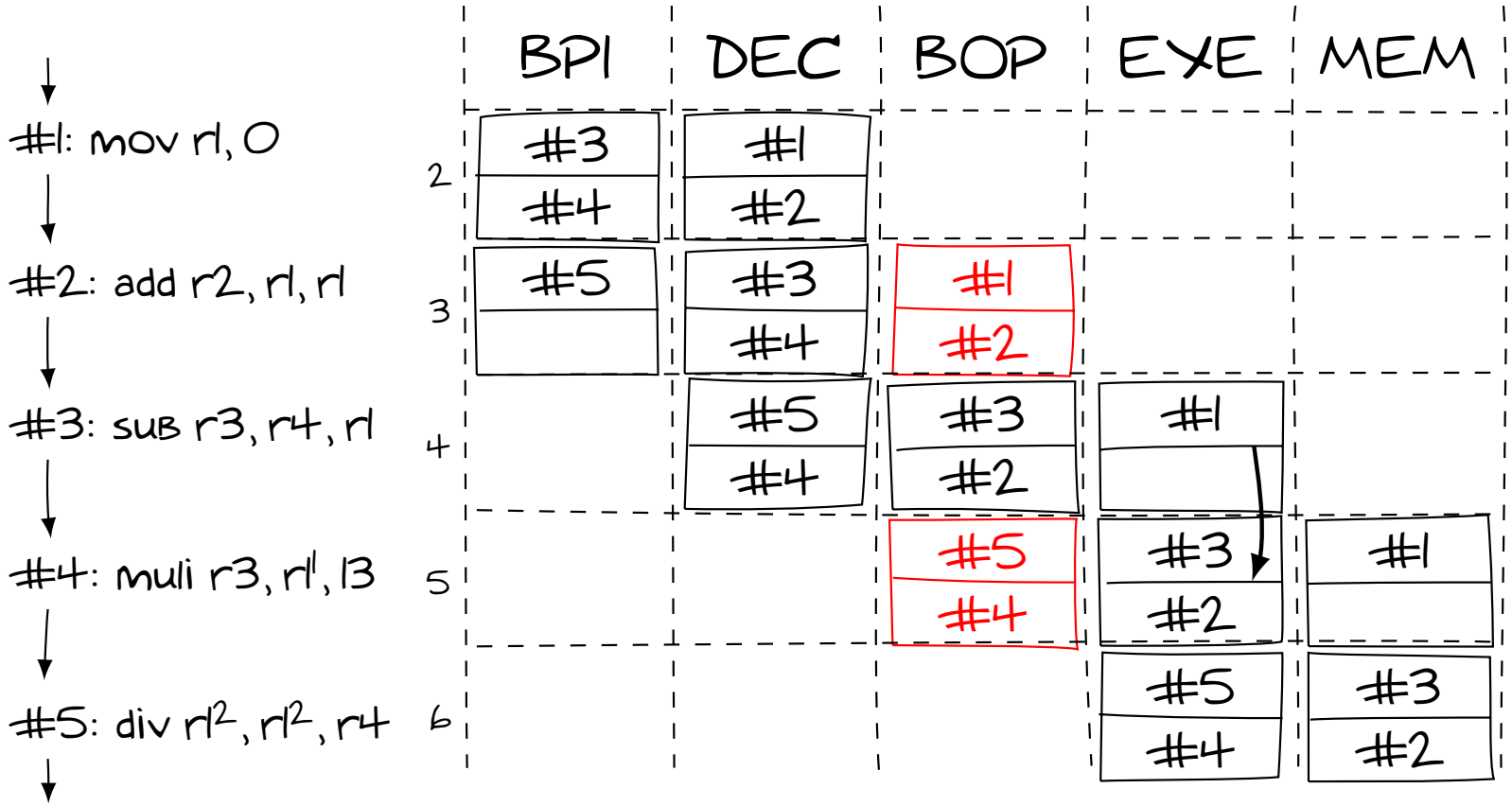
- Emissão e finalização fora de ordem
  - São utilizadas unidades de reserva para suportar a execução fora de ordem das instruções



É feito o adiantamento para a instrução #2 e é detectado um conflito WAR entre #5 e #4

Superescalar

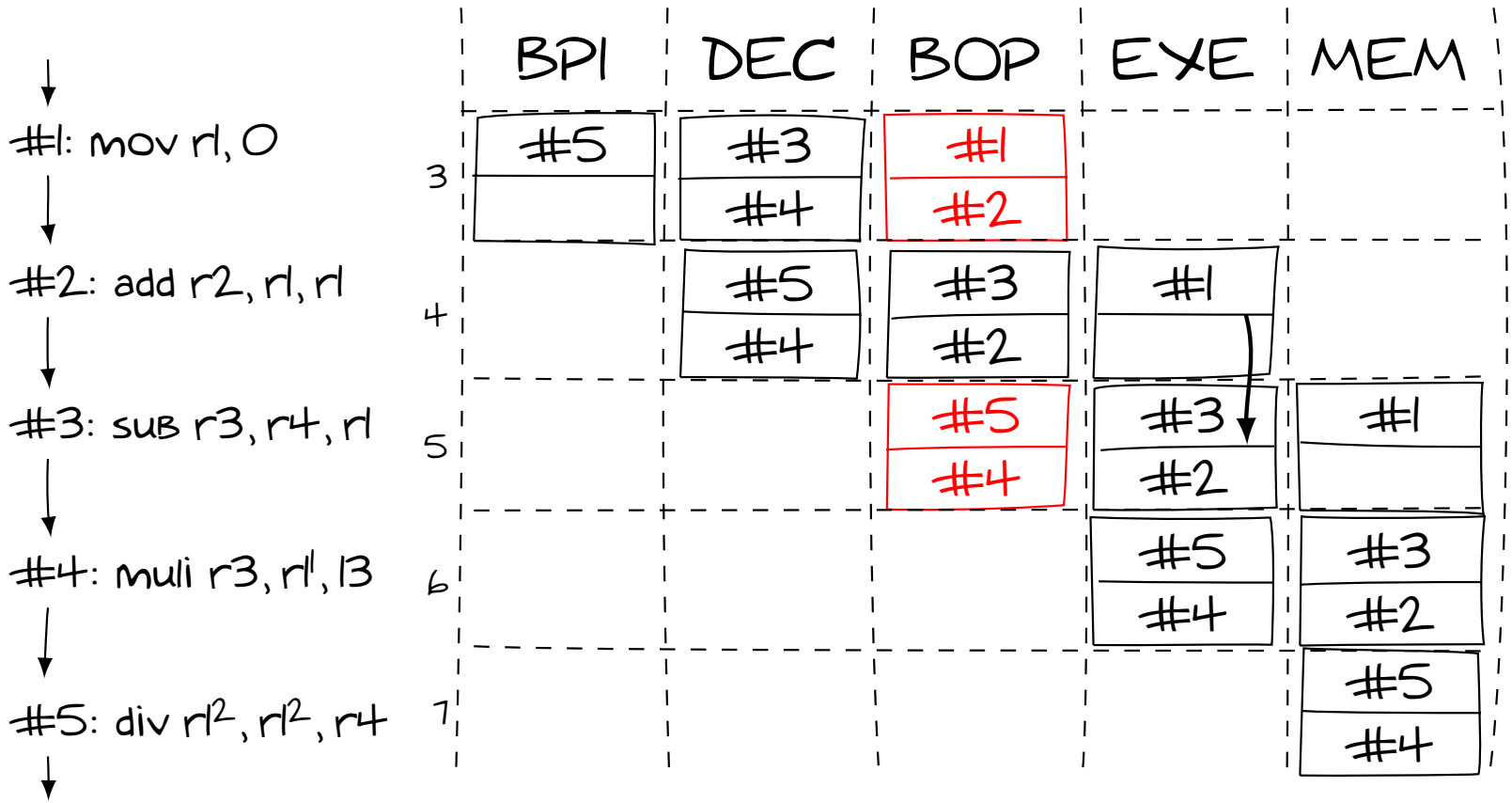
- Emissão e finalização fora de ordem
  - São utilizadas unidades de reserva para suportar a execução fora de ordem das instruções



Ao invés de sequencializar #5 e #4,
é feito o renomeamento de R1 (WAR)

Superescalar

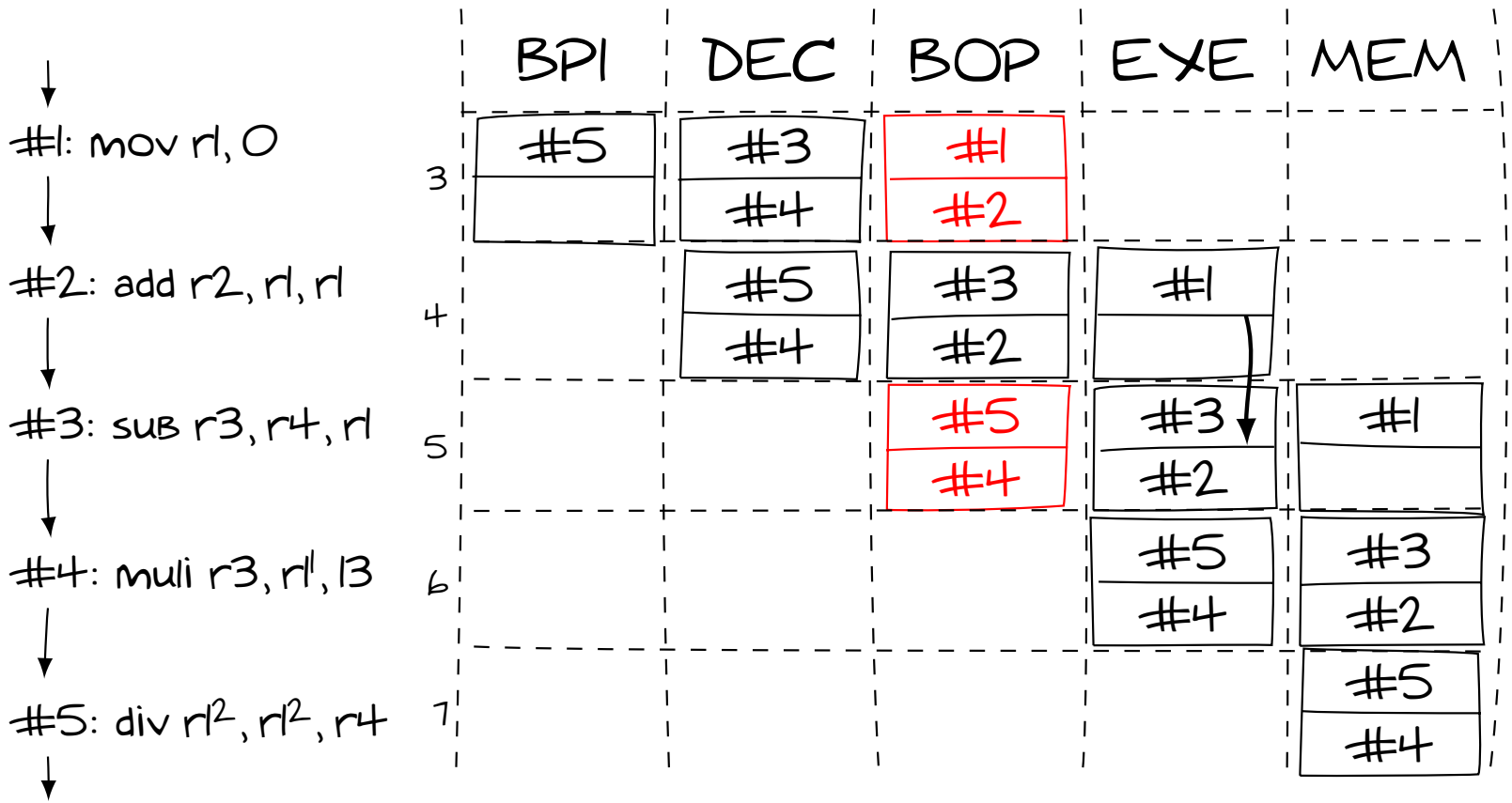
- Emissão e finalização fora de ordem
  - São utilizadas unidades de reserva para suportar a execução fora de ordem das instruções



R1' é lido independentemente da escrita de R1²

Superescalar

- Emissão e finalização fora de ordem
  - São utilizadas unidades de reserva para suportar a execução fora de ordem das instruções



A execução é finalizada em 7 ciclos

Superescalar

- Técnicas estáticas de compilação do software para explorar o paralelismo de instruções
  - Desenrolamento de laço (*loop unrolling*)
    - É possível ser aplicado em casos que as operações realizadas nos laços são independentes e que o número de iterações é conhecido
    - Replica as operações do laço, removendo as instruções de desvio e utilizando diferentes registradores para armazenar os dados

Superescalar

- Técnicas estáticas de compilação do software para explorar o paralelismo de instruções
  - Desenrolamento de laço (*loop unrolling*)
    - É possível ser aplicado em casos que as operações realizadas nos laços são independentes e que o número de iterações é conhecido
    - Replica as operações do laço, removendo as instruções de desvio e utilizando diferentes registradores para armazenar os dados
  - *Very Long Instruction Word* (VLIW)
    - O compilador escalona estaticamente as instruções do software, gerando um pacote de instruções explicitamente paralelas e sem dependências
    - Este pacote de instruções possui um tamanho fixo e aumenta a quantidade de instruções em execução

Superescalar

- Desenrolamento de laços (*loop unrolling*)
  - Código fonte em C ($k = 3, V = 0x100, n = 4$)

```
1 // Multiplicação escalar de vetor
2 void mult(int32_t k, int32_t V[], uint32_t n) {
3     // Índices
4     for(uint32_t i = 0; i < n; i++) {
5         // Multiplicação escalar
6         V[i] = k * V[i];
7     }
8 }
```

Superescalar

- Desenrolamento de laços (*loop unrolling*)
  - Código de montagem não otimizado

```
1  // R1 = k, R2 = V, R3 = n, R4 = i
2  mov r1, 3
3  mov r2, 0x40
4  mov r3, 4
5  mov r4, 0
6  // i ? n
7  cmpi r4, r3
8  bae 6
9  // V[i] = k * V[i]
10 l32 r5, [r2]
11 mul r5, r1, r5
12 s32 [r2], r5
13 // i++
14 addi r2, r2, 1
15 addi r4, r4, 1
16 bun -8
17 // Fim
18 int 0
```

Superescalar

- Desenrolamento de laços (*loop unrolling*)
  - Código de montagem otimizado

```
1   // R1 = k, R2 = V
...
4   l32 r3, [r2]
5   mul r3, r1, r3
6   s32 [r2], r3
7   addi r, r2, 1
...
16  l32 r3, [r2]
17  mul r3, r1, r3
18  s32 [r2], r3
19  addi r2, r2, 1
20  // Fim
21  int 0
```

Os desvios do laço são eliminados, entretanto, mais memória é utilizada para armazenar as instruções

Superescalar

- *Very Long Instruction Word* (VLIW)
  - Múltiplas instruções são organizadas em pacotes, considerando a estrutura do interna do processador

```
1  l32 r3, [r2 + 0]
2  l32 r4, [r2 + 1]
3  l32 r5, [r2 + 2]
4  l32 r6, [r2 + 3]
5  muli r3, r3, 3
6  muli r4, r4, 3
7  muli r5, r5, 3
8  muli r6, r6, 3
9  s32 [r2 + 0], r3
10 s32 [r2 + 1], r4
11 s32 [r2 + 2], r5
12 s32 [r2 + 3], r6
```

$\Longrightarrow$

Instrução VLIW

#01	#02	#03	#04
#05	#06	#07	#08
#09	#10	#11	#12

Superescalar

- *Very Long Instruction Word* (VLIW)
  - Aumento do tamanho do código gerado
    - Otimizações de compilação
    - Exploração do paralelismo

Superescalar

- *Very Long Instruction Word* (VLIW)
  - Aumento do tamanho do código gerado
    - Otimizações de compilação
    - Exploração do paralelismo
  - Preenchimento parcial dos pacotes de instruções
    - O escalonamento estático não preenche o pacote
    - Os espaços vazios são completados com *nop*

Superescalar

- *Very Long Instruction Word* (VLIW)
  - Aumento do tamanho do código gerado
    - Otimizações de compilação
    - Exploração do paralelismo
  - Preenchimento parcial dos pacotes de instruções
    - O escalonamento estático não preenche o pacote
    - Os espaços vazios são completados com *nop*
  - Problemas de incompatibilidade binária
    - O código gerado reflete estrutura do processador
    - Diferentes processadores possuem diferentes organizações e quantidades de unidades funcionais

Superescalar

- Limitações do paralelismo em nível de instrução
  - A exploração do paradigma ILP teve início nos anos de 1960 e atingiu os maiores níveis de melhoria de desempenho nos anos de 1980 e 1990

Superescalar

- Limitações do paralelismo em nível de instrução
  - A exploração do paradigma ILP teve início nos anos de 1960 e atingiu os maiores níveis de melhoria de desempenho nos anos de 1980 e 1990
  - Alguns estudos foram conduzidos para se descobrir o que seria necessário para aumentar ainda mais o desempenho, tanto na perspectiva do projeto de hardware como na construção de compiladores

Superescalar

- Limitações do paralelismo em nível de instrução
  - A exploração do paradigma ILP teve início nos anos de 1960 e atingiu os maiores níveis de melhoria de desempenho nos anos de 1980 e 1990
  - Alguns estudos foram conduzidos para se descobrir o que seria necessário para aumentar ainda mais o desempenho, tanto na perspectiva do projeto de hardware como na construção de compiladores
    - Quantidade infinita de registradores

Superescalar

- Limitações do paralelismo em nível de instrução
  - A exploração do paradigma ILP teve início nos anos de 1960 e atingiu os maiores níveis de melhoria de desempenho nos anos de 1980 e 1990
  - Alguns estudos foram conduzidos para se descobrir o que seria necessário para aumentar ainda mais o desempenho, tanto na perspectiva do projeto de hardware como na construção de compiladores
    - Quantidade infinita de registradores
    - Predição perfeita de desvios

Superescalar

- Limitações do paralelismo em nível de instrução
  - A exploração do paradigma ILP teve início nos anos de 1960 e atingiu os maiores níveis de melhoria de desempenho nos anos de 1980 e 1990
  - Alguns estudos foram conduzidos para se descobrir o que seria necessário para aumentar ainda mais o desempenho, tanto na perspectiva do projeto de hardware como na construção de compiladores
    - Quantidade infinita de registradores
    - Predição perfeita de desvios
    - Caches sem faltas de dados

Superescalar

- Limitações do paralelismo em nível de instrução
  - Os resultados mostraram barreiras formidáveis para aumentar o desempenho no paradigma ILP, sendo observado que a área de silício utilizada e o consumo de potência são excessivamente altos

Superescalar

- Limitações do paralelismo em nível de instrução
  - Os resultados mostraram barreiras formidáveis para aumentar o desempenho no paradigma ILP, sendo observado que a área de silício utilizada e o consumo de potência são excessivamente altos
  - O aumento de complexidade, a redução da frequência de operação e o aumento de potência não são compensados pelos pequenos ganhos

Superescalar

- Limitações do paralelismo em nível de instrução
  - Os resultados mostraram barreiras formidáveis para aumentar o desempenho no paradigma ILP, sendo observado que a área de silício utilizada e o consumo de potência são excessivamente altos
  - O aumento de complexidade, a redução da frequência de operação e o aumento de potência não são compensados pelos pequenos ganhos
    - O paradigma de multiprocessamento emergiu como alternativa para manter a taxa de crescimento de capacidade dos processadores

Superescalar

- Limitações do paralelismo em nível de instrução
  - Os resultados mostraram barreiras formidáveis para aumentar o desempenho no paradigma ILP, sendo observado que a área de silício utilizada e o consumo de potência são excessivamente altos
  - O aumento de complexidade, a redução da frequência de operação e o aumento de potência não são compensados pelos pequenos ganhos
    - O paradigma de multiprocessamento emergiu como alternativa para manter a taxa de crescimento de capacidade dos processadores
    - Com núcleos de processamento menores e mais eficientes, a organização multiprocessada permite que o sistema seja escalável e redundante

Superescalar

- Limitações do paralelismo em nível de instrução
  - Os resultados mostraram barreiras formidáveis para aumentar o desempenho no paradigma ILP, sendo observado que a área de silício utilizada e o consumo de potência são excessivamente altos
  - O aumento de complexidade, a redução da frequência de operação e o aumento de potência não são compensados pelos pequenos ganhos
    - O paradigma de multiprocessamento emergiu como alternativa para manter a taxa de crescimento de capacidade dos processadores
    - Com núcleos de processamento menores e mais eficientes, a organização multiprocessada permite que o sistema seja escalável e redundante
    - Em oposição à exploração do paralelismo implícito entre as instruções, o multiprocessamento depende que a programação seja feita de forma explícita para aproveitar os processadores disponíveis

Exercício

- Implemente um *pipeline* de 5 estágios com caches de dados e de instrução e tratamento de conflitos
  - Adiantamento de dados
    - Memória: sobreposição parcial ou total de endereços da memória nas instruções de escrita ou leitura
    - Registrador: operandos de saída são utilizados como entrada pela instrução consecutiva, com exceção dos registradores especiais R0, CR, IPC, IR, PC, SP e SR
  - Execução especulativa
    - Instrução de desvio ou interrupção de software: a predição é dinâmica (máquina de estados) com endereços de desvio calculados a partir dos operandos da instrução ou utilizando o último endereço de retorno armazenado (ret e reti)
    - Interrupção de hardware: os eventos de interrupção assíncronos não podem ser previstos, logo, seu impacto não é contabilizado na predição de desvios, sendo feito apenas a invalidação das instruções

Exercício

- Os eventos de adiantamento de dados em memória e registradores e de execução especulativa devem ser exibidos durante a execução de cada uma das instruções, com as taxas de acerto de predição e aumento de desempenho no final da execução

```
[START OF SIMULATION]
.
.
.
[BRANCH MISPREDICTION @ 0x????????]
[DATA FORWARD @ Rx]
[DATA FORWARD @ 0x????????]
[INSTRUCTION FLUSHING]
.
.
.
[PIPELINE]
branch_prediction_accuracy: ??.??%
performance_speed_up: ?.??x
[END OF SIMULATION]
```