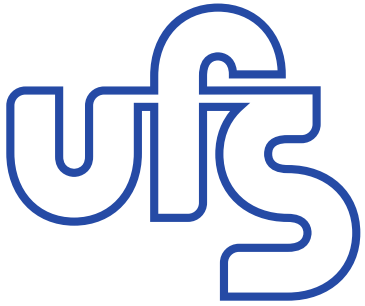
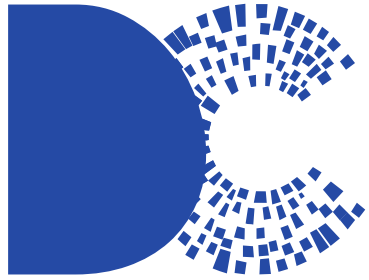


UNIVERSIDADE FEDERAL DE SERGIPE



DEPARTAMENTO DE COMPUTAÇÃO

Superescalar

Arquitetura de Computadores

Bruno Prado

Departamento de Computação / UFS

Introdução

- O que é um *superescalar*?
  - É uma arquitetura com a capacidade de executar instruções concorrentemente e independentemente em diferentes unidades de processamento

Introdução

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

Introdução

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

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

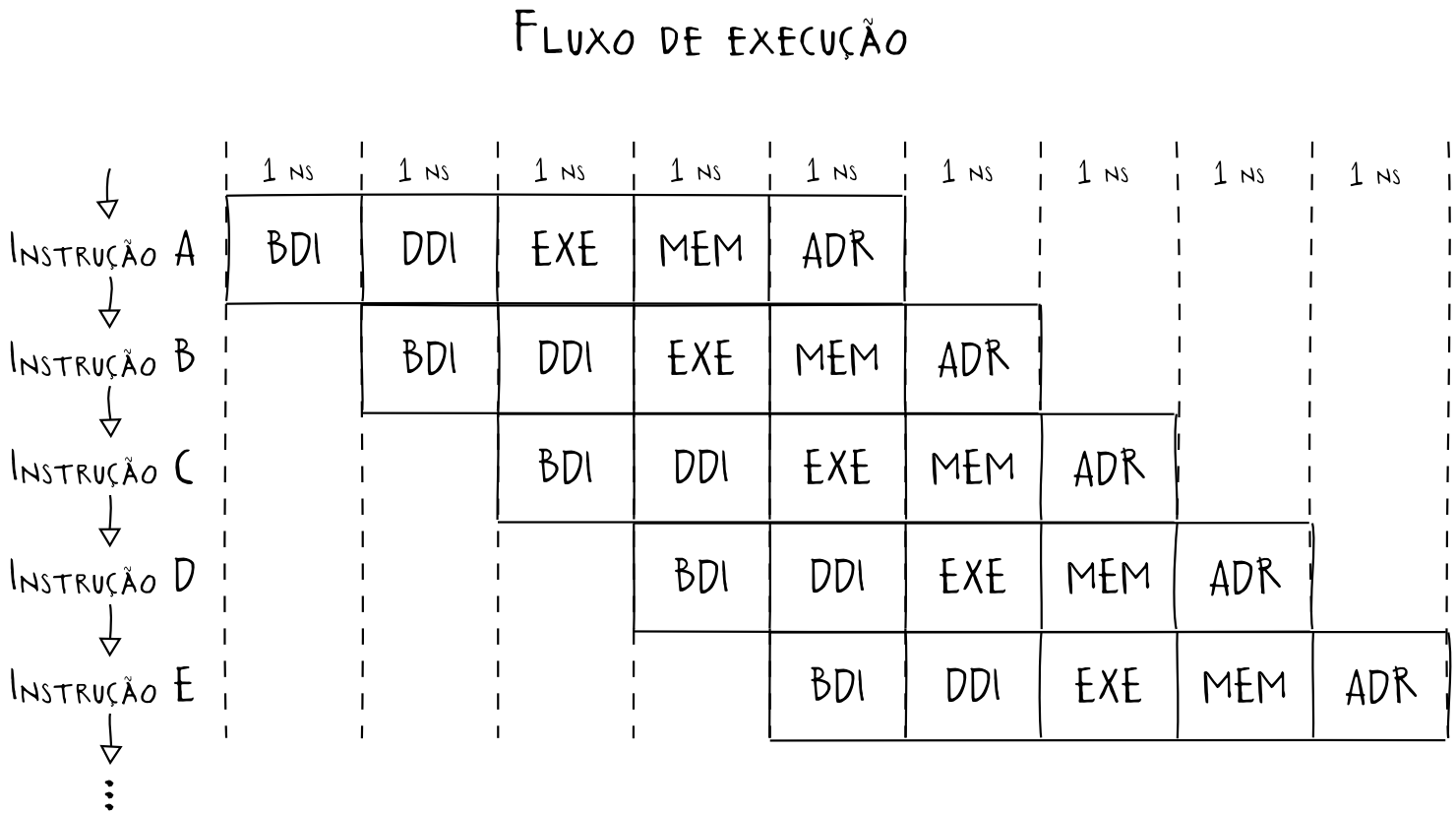
- *Cycles per Instruction* (CPI)
  - É uma métrica para avaliar quantos ciclos de relógio são necessários para executar uma instrução
  - Permite avaliar o desempenho do processador

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

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

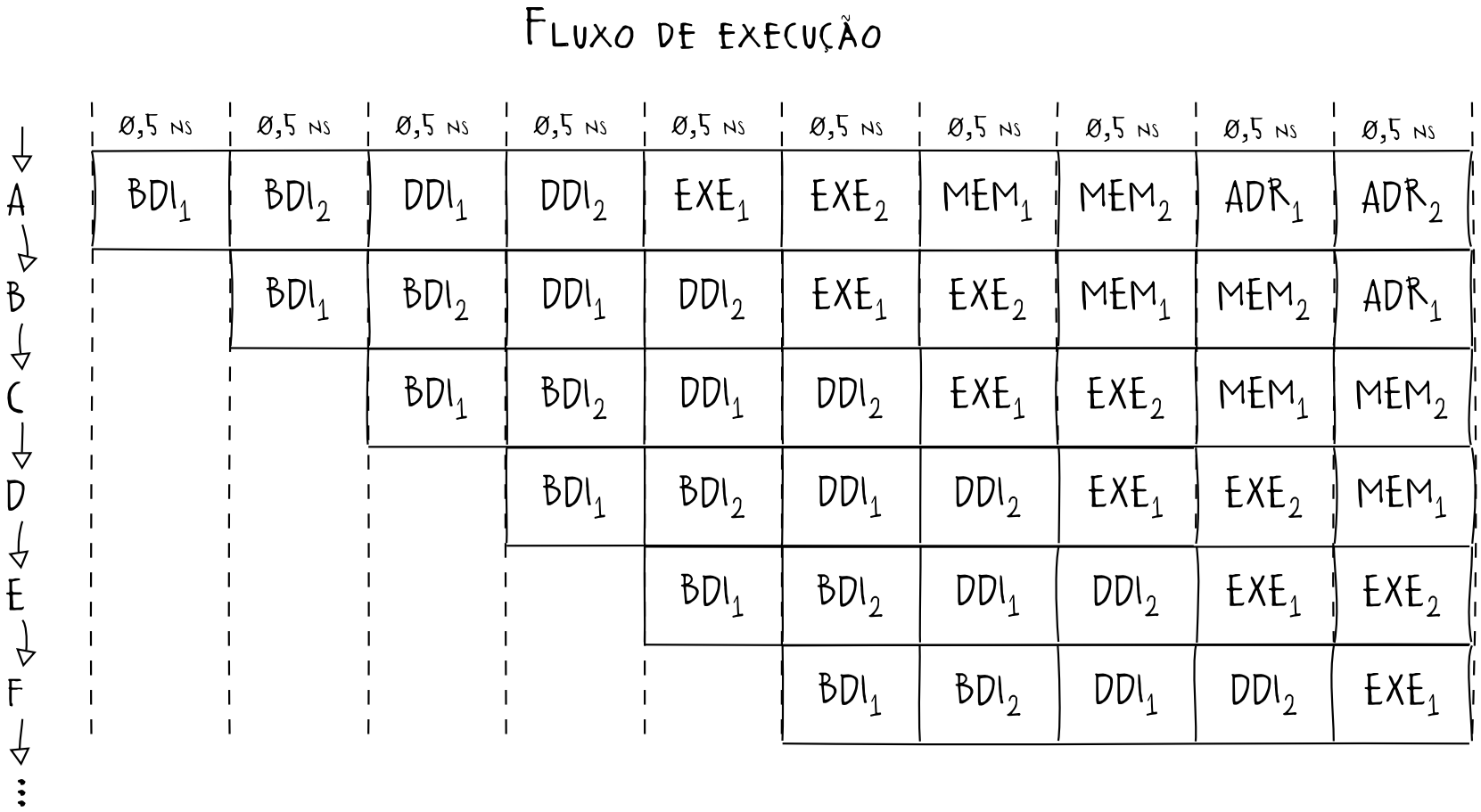
Introdução

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



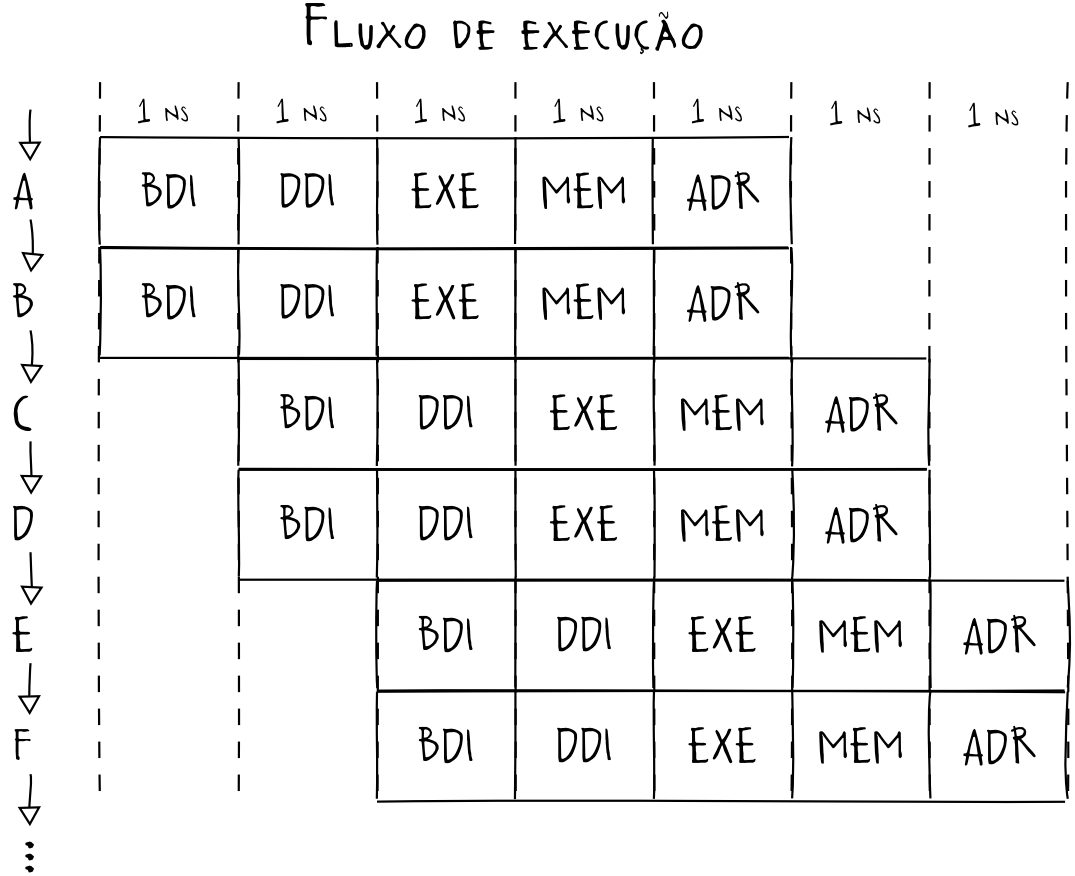
Introdução

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



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 grau de paralelismo é definido pela quantidade média de instruções que podem ser executadas em paralelo nas unidades de processamento (métrica CPI)

Introdução

- *Instruction-Level Parallelism* (ILP)
  - O grau de paralelismo é definido pela quantidade média de instruções que podem ser executadas em paralelo nas unidades de processamento (métrica CPI)
  - Para maximizar o paralelismo são necessárias otimizações combinadas de hardware e de software
    - O hardware de controle procura aproveitar todas as unidades de processamento disponíveis
    - A compilação organiza estaticamente uma sequência de instruções para execução paralela

Introdução

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

```
1  // Multiplicação escalar de vetor
2  void mulsv(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 mulsv(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 das instruções é condicionada pela existência de conflito ou de dependência entre elas

Superescalar

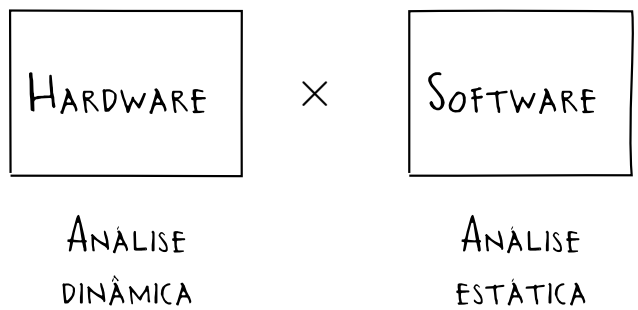
- Conflitos e dependências de dado
  - A execução paralela das instruções é condicionada pela existência de conflito ou de dependência entre elas
  - Duas ou mais instruções são paralelas quando a execução simultânea nos *pipelines* não gera atrasos

Superescalar

- Conflitos e dependências de dado
  - A execução paralela das instruções é condicionada pela existência de conflito ou de dependência entre elas
  - Duas ou mais instruções são paralelas quando a execução simultânea nos *pipelines* 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 das instruções é condicionada pela existência de conflito ou de dependência entre elas
  - Duas ou mais instruções são paralelas quando a execução simultânea nos *pipelines* 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 como operando pela instrução *j*

```
    1  // t1 = 1
i -> 2  addi t1, zero, 1
    3  // t2 = t2 + t1
j -> 4  add t2, t2, t1
```

Superescalar

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

```
    1  // t1 = 1
i ->2  addi t1, zero, 1
    3  // t2 = t2 + t1
j ->4  add t2, t2, t1
```

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

```
    1  // t1 = 1
i ->2  addi t1, zero, 1
    3  // t2 = t2 + t1
j ->4  add t2, t2, t1
    5  // t3 = t1 * t2
k ->6  mul t3, t1, t2
```

Superescalar

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

```
  1  // t1 = 1
i->2 addi t1, zero, 1
  3  // t2 = t2 + t1
j->4 add t2, t2, t1
```

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

```
  1  // t1 = 1
i->2 addi t1, zero, 1
  3  // t2 = t2 + t1
j->4 add t2, t2, t1
  5  // t3 = t1 * t2
k->6 mul t3, t1, t2
```

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
  - Tratamento destes conflitos *Read After Write* (RAW)
    - Manter a dependência e evitar o conflito de dados com adiantamento de dados no *pipeline*
    - Eliminar a dependência pela transformação do código por escalonamento dinâmico ou estático

Superescalar

- Dependência de nome
  - Anti-dependência: a instrução *j* escreve em um registrador ou endereço de memória que também é lido pela instrução *i*, causando o conflito *Write After Read* (WAR)

```
1     // t1 = t2
i->2  add t1, zero, t2
3     // t2 = 1
j->4  addi t2, zero, 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*, causando o conflito *Write After Read* (WAR)

```
   1  // t1 = t2
i->2  add t1, zero, t2
   3  // t2 = 1
j->4  addi t2, zero, 1
```

  - Dependência de saída: duas instruções *i* e *j* escrevem no mesmo registrador ou endereço de memória, gerando um conflito *Write After Write* (WAW)

```
   1  // mem[t3] = t4
i->2  sw t4, 0(t3)
   3  // mem[t3] = t5
j->4  sw t5, 0(t3)
```

Superescalar

- Dependência de nome
  - Anti-dependência: a instrução *j* escreve em um registrador ou endereço de memória que também é lido pela instrução *i*, causando o conflito *Write After Read* (WAR)

```
   1  // t1 = t2
i->2  add t1, zero, t2
   3  // t2 = 1
j->4  addi t2, zero, 1
```

  - Dependência de saída: duas instruções *i* e *j* escrevem no mesmo registrador ou endereço de memória, gerando um conflito *Write After Write* (WAW)

```
   1  // mem[t3] = t4
i->2  sw t4, 0(t3)
   3  // mem[t3] = t5
j->4  sw t5, 0(t3)
```

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

- Dependência de controle

```
1   // Inteiros com tamanho fixo
2   #include <stdint.h>
3   // Biblioteca padrão
4   #include <stdlib.h>
5   // Procedimento DAA
6   void DAA(uint32_t id, uint8_t faltas) {
7       // (faltas > 15 horas) -> reprovar
8       if(faltas > 15) reprovar(id);
9       // (faltas <= 15) -> checar nota
10      else checar_nota(id);
11  }
…   ...
```

Fluxos de desvio da aplicação

Superescalar

- Dependência de controle

```
25  # DAA(a0 = id, a1 = faltas)
26  DAA:
27      # t0 = 15
28      li t0, 15
29      # (faltas > 15) -> reprovar
30      bgt a1, t0, reprovar
31      # (faltas <= 15) -> checar_nota
32      checar_nota:
...         ...
45      reprovar:
...         ...
55      # end
56      ret
```

Superescalar

- Dependência de controle

```
25  # DAA(a0 = id, a1 = faltas)
26  DAA:
27      # t0 = 15
28      li t0, 15
29      # (faltas > 15) -> reprovar
30      bgt a1, t0, reprovar
31      # (faltas <= 15) -> checar_nota
32      checar_nota:
...         ...
45      reprovar:
...         ...
55      # end
56      ret
```

O erro na predição do desvio
pode executar instruções incorretas

Superescalar

- Comportamento de exceção

```
63  # Função principal
64  main:
…       ...
75      # Instrução inválida
76      .word 0xf0f0f0f0
77      # t1 = t2
78      add t1, zero, t2
79      # mem[t3] = t1
80      sw t1, 0(t3)
81      # t2 = mem[t4]
82      lw t2, 0(t4)
…       ...
```

Superescalar

- Comportamento de exceção

```
63  # Função principal
64  main:
…       ...
75      # Instrução inválida
76      .word 0xf0f0f0f0
77      # t1 = t2
78      add t1, zero, t2
79      # mem[t3] = t1
80      sw t1, 0(t3)
81      # t2 = mem[t4]
82      lw t2, 0(t4)
…       ...
```

É feito o cancelamento das instruções 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

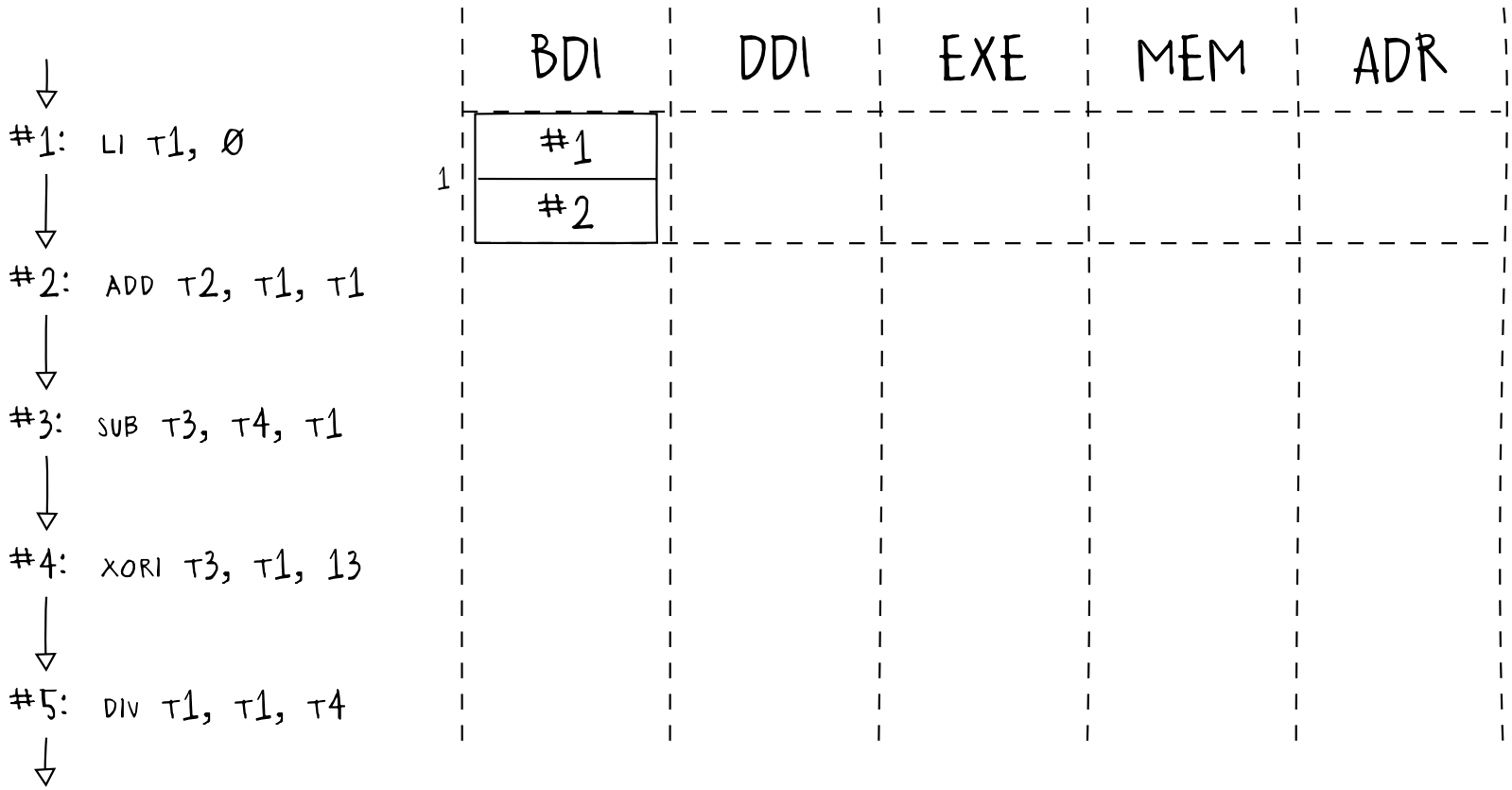
Superescalar

- Políticas de emissão e finalização de instruções
  - Sequência de instruções com conflitos para execução paralela em um superescalar com dois *pipelines*

```
1 li t1, 0
2 add t2, t1, t1
3 sub t3, t4, t1
4 xori t3, t1, 13
5 div t1, t1, t4
```

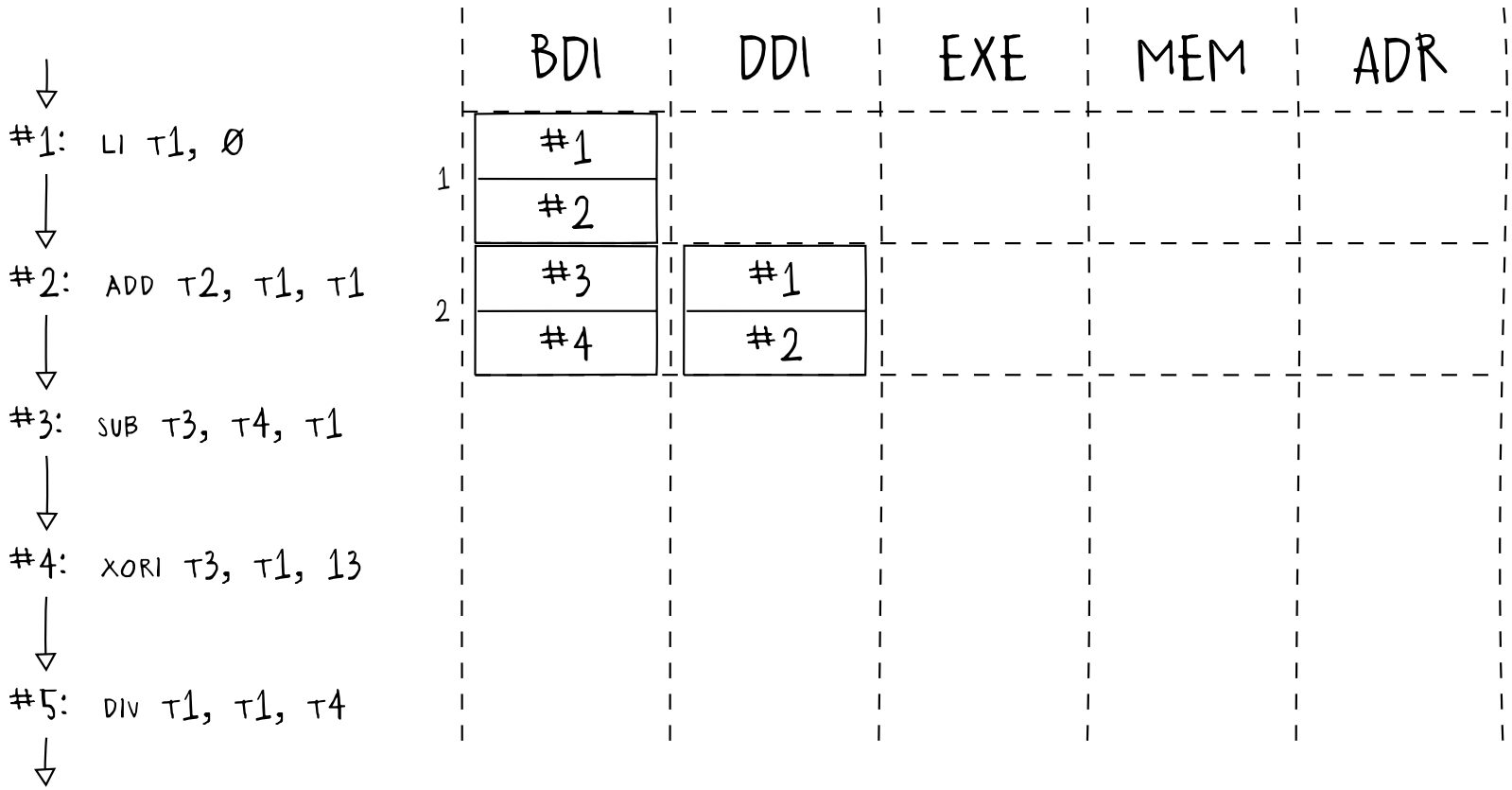
Superescalar

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



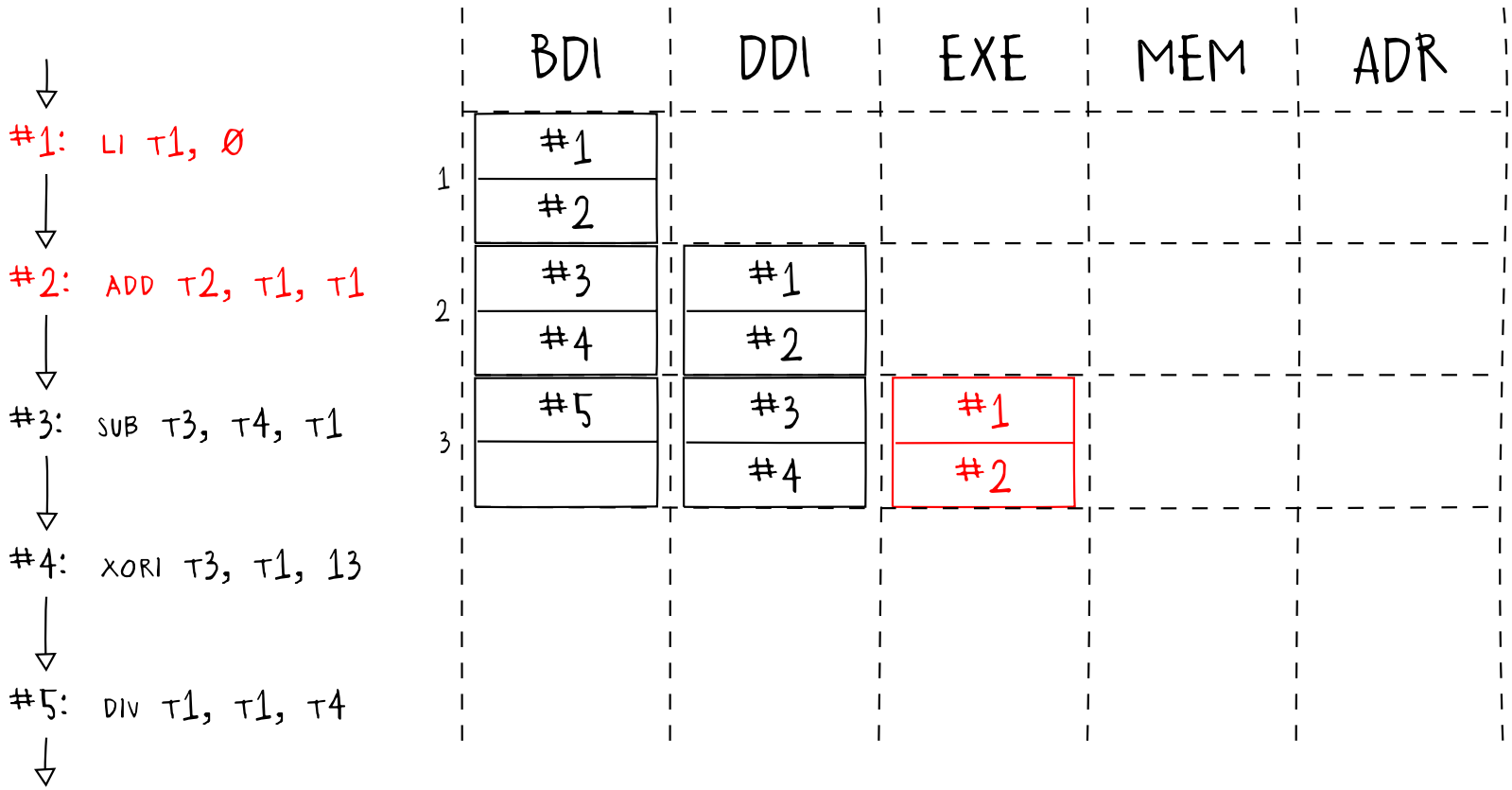
Superescalar

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



Superescalar

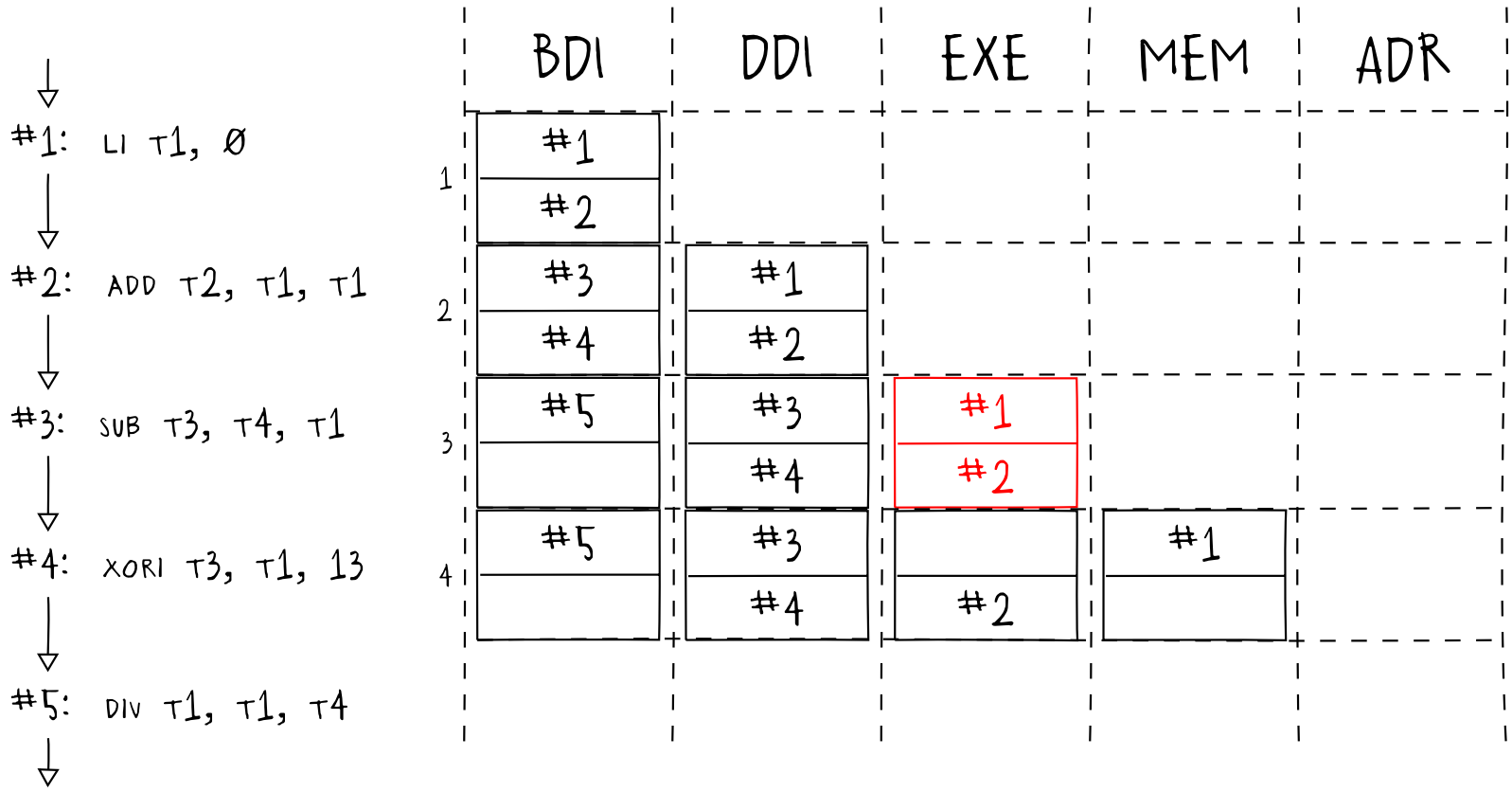
- Emissão e finalização em ordem
  - As instruções são buscadas em ordem, mantendo o comportamento sequencial do software



A INSTRUÇÃO #1 TEM COMO SAÍDA T1 QUE É ENTRADA PARA INSTRUÇÃO #2 (CONFLITO RAW)

Superescalar

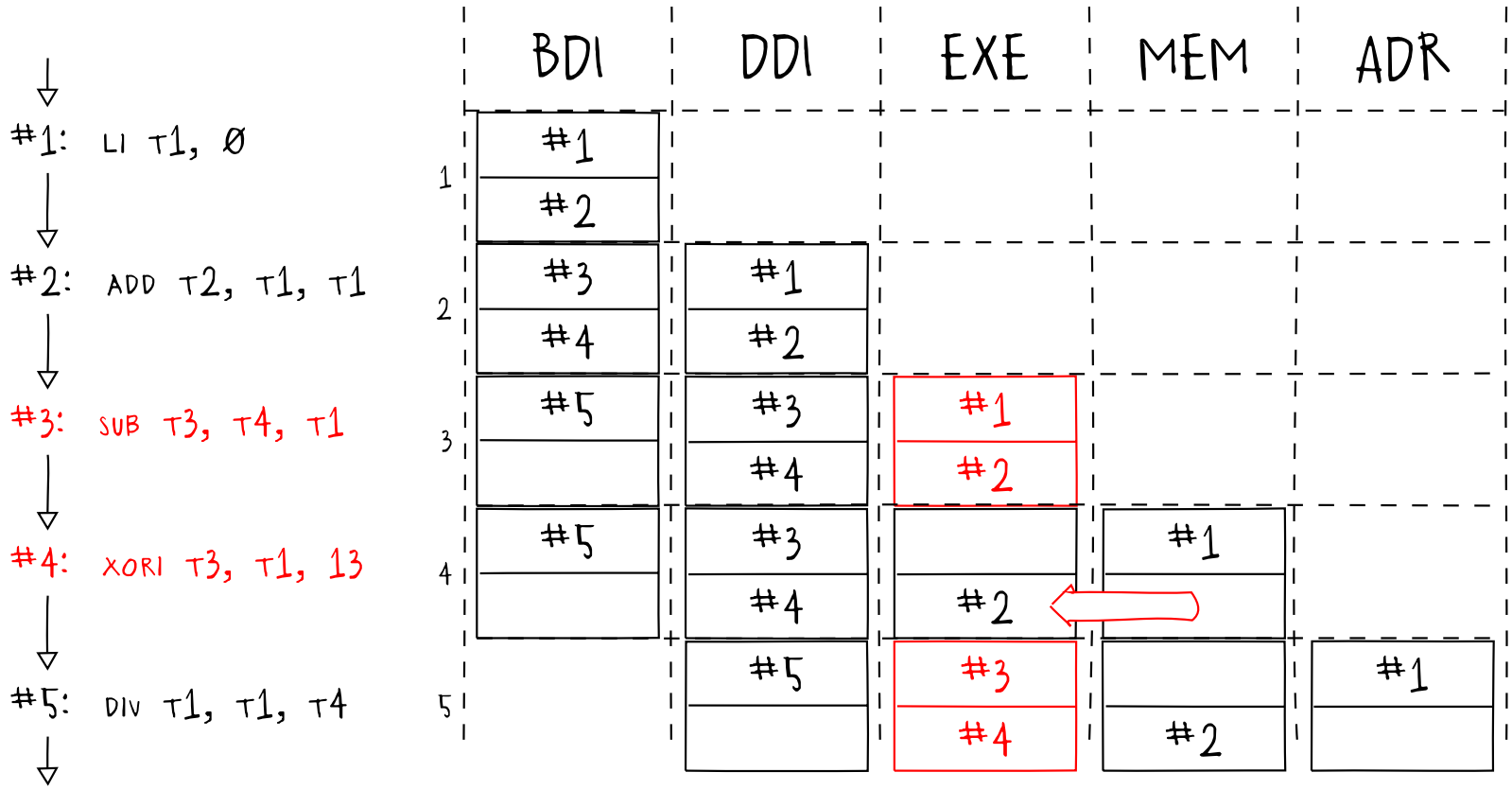
- Emissão e finalização em ordem
  - As instruções são buscadas em ordem, mantendo o comportamento sequencial do software



A INSTRUÇÃO #1 SEGUE PARA O PRÓXIMO ESTÁGIO, ENQUANTO QUE INSTRUÇÃO #2 FICA EM ESPERA (STALLED)

Superescalar

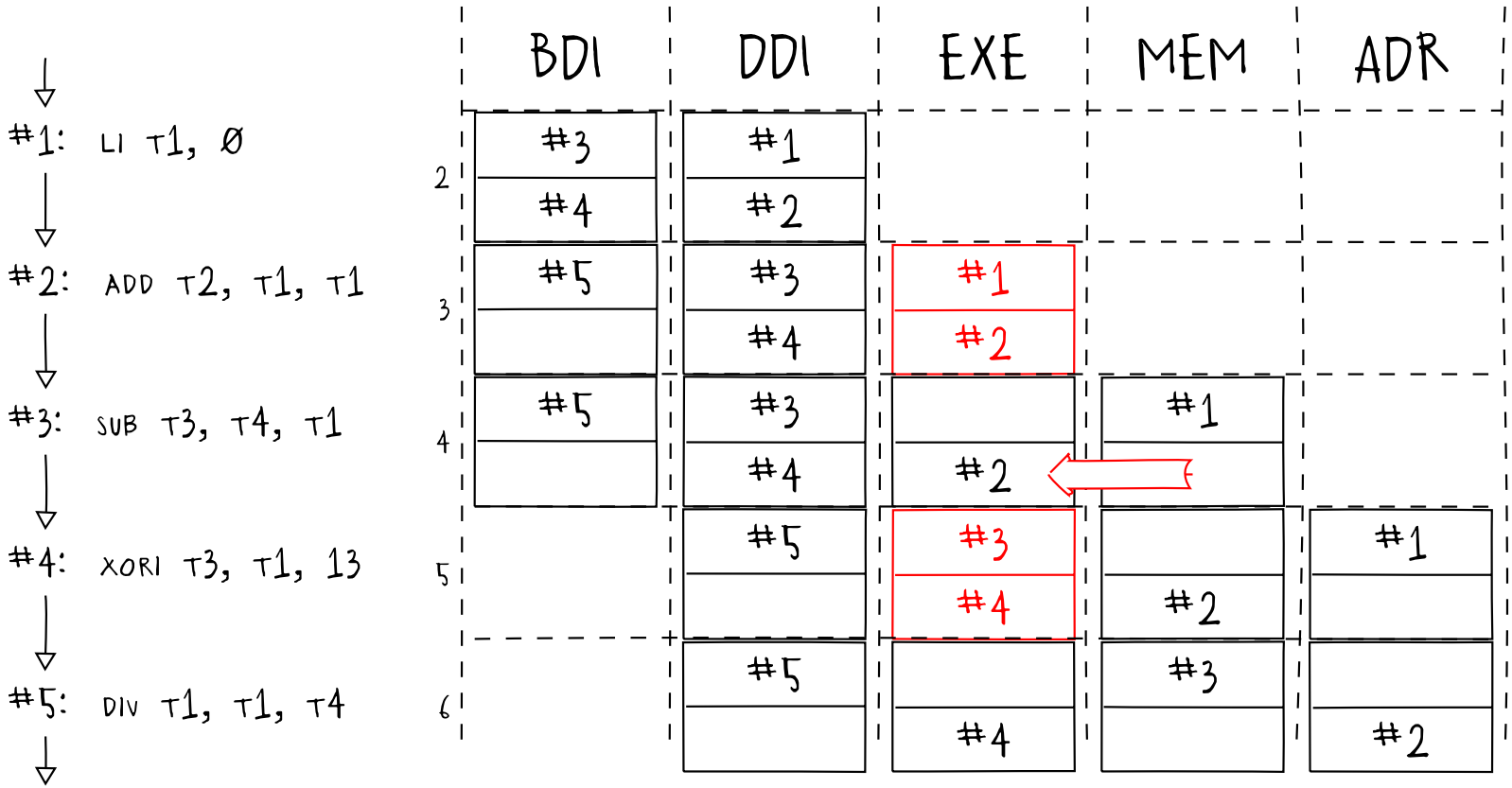
- Emissão e finalização em ordem
  - As instruções são buscadas em ordem, mantendo o comportamento sequencial do software



É FEITO O ADIANTAMENTO PARA A INSTRUÇÃO #2 E
É DETECTADO UM CONFLITO WAW ENTRE #3 E #4

Superescalar

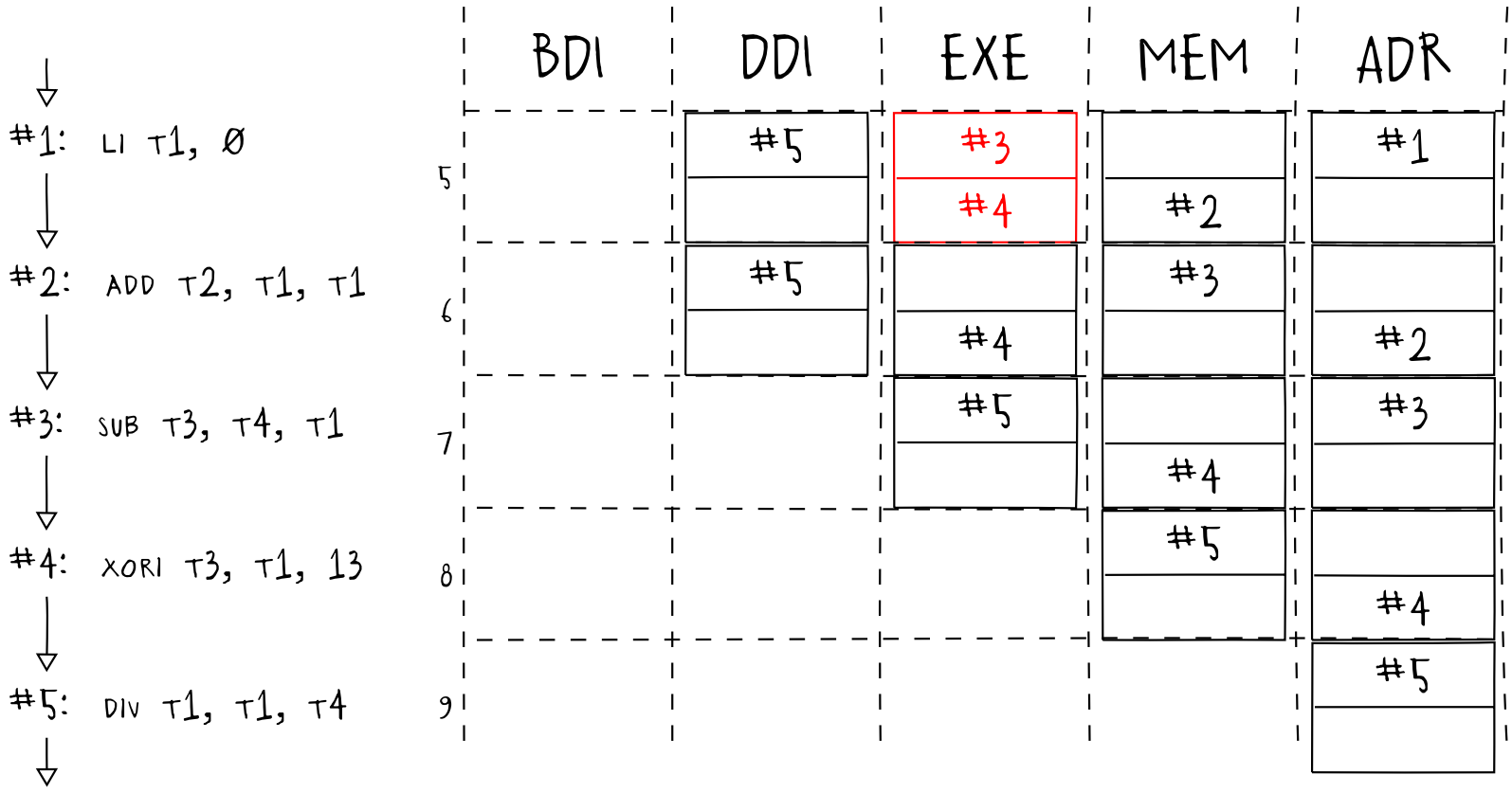
- Emissão e finalização em ordem
  - As instruções são buscadas em ordem, mantendo o comportamento sequencial do software



PARA GARANTIR A ORDEM DE EXECUÇÃO,
INSTRUÇÃO #4 FICA EM ESPERA (STALLED)

Superescalar

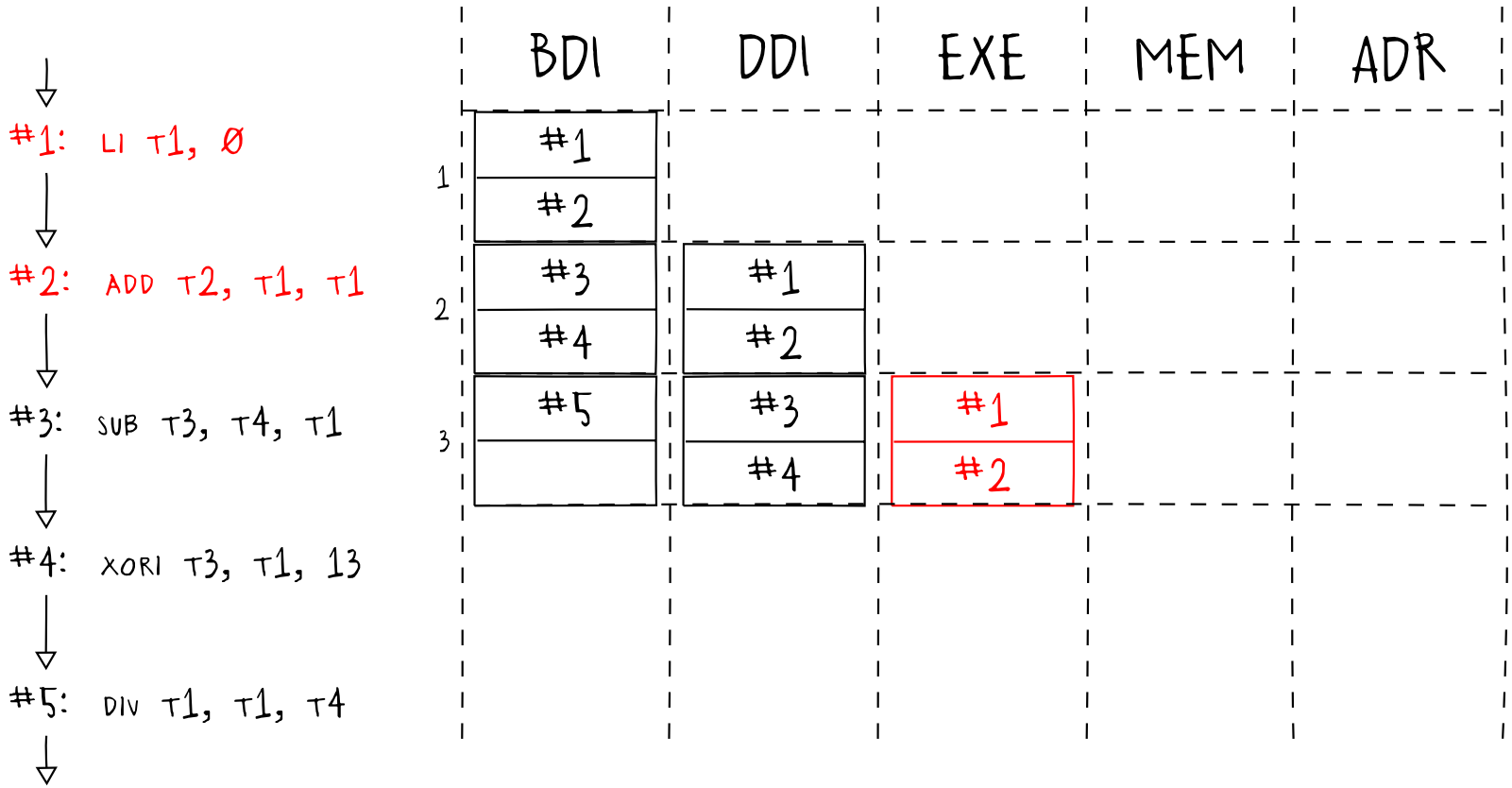
- Emissão e finalização em ordem
  - As instruções são buscadas em ordem, mantendo o comportamento sequencial do software



A EXECUÇÃO É FINALIZADA EM 9 CICLOS

Superescalar

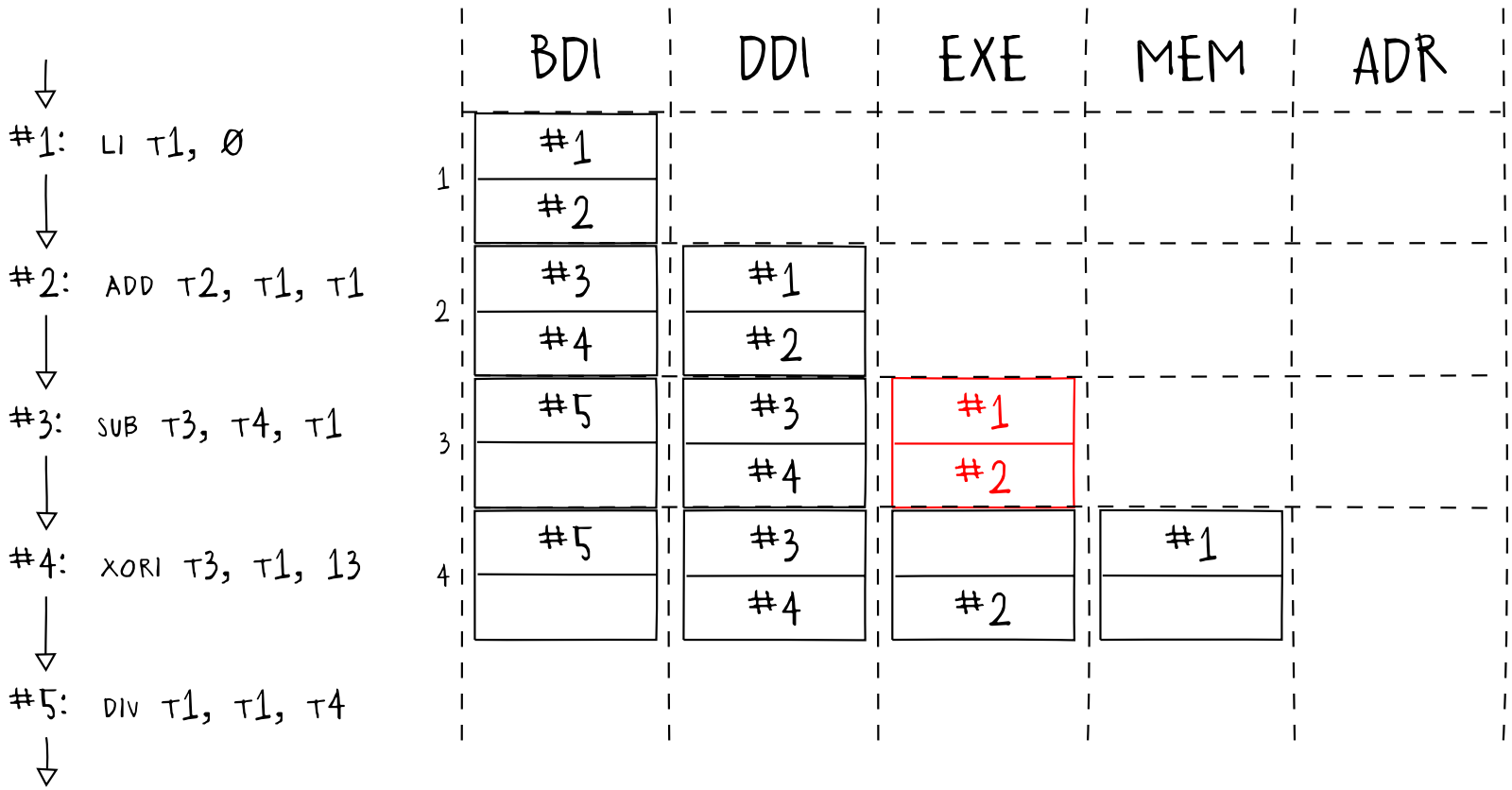
- Emissão em ordem com finalização fora de ordem
  - As instruções são executadas em ordem e o armazenamento dos resultados é feito fora de ordem



A INSTRUÇÃO #1 TEM COMO SAÍDA T1 QUE É ENTRADA PARA INSTRUÇÃO #2 (CONFLITO RAW)

Superescalar

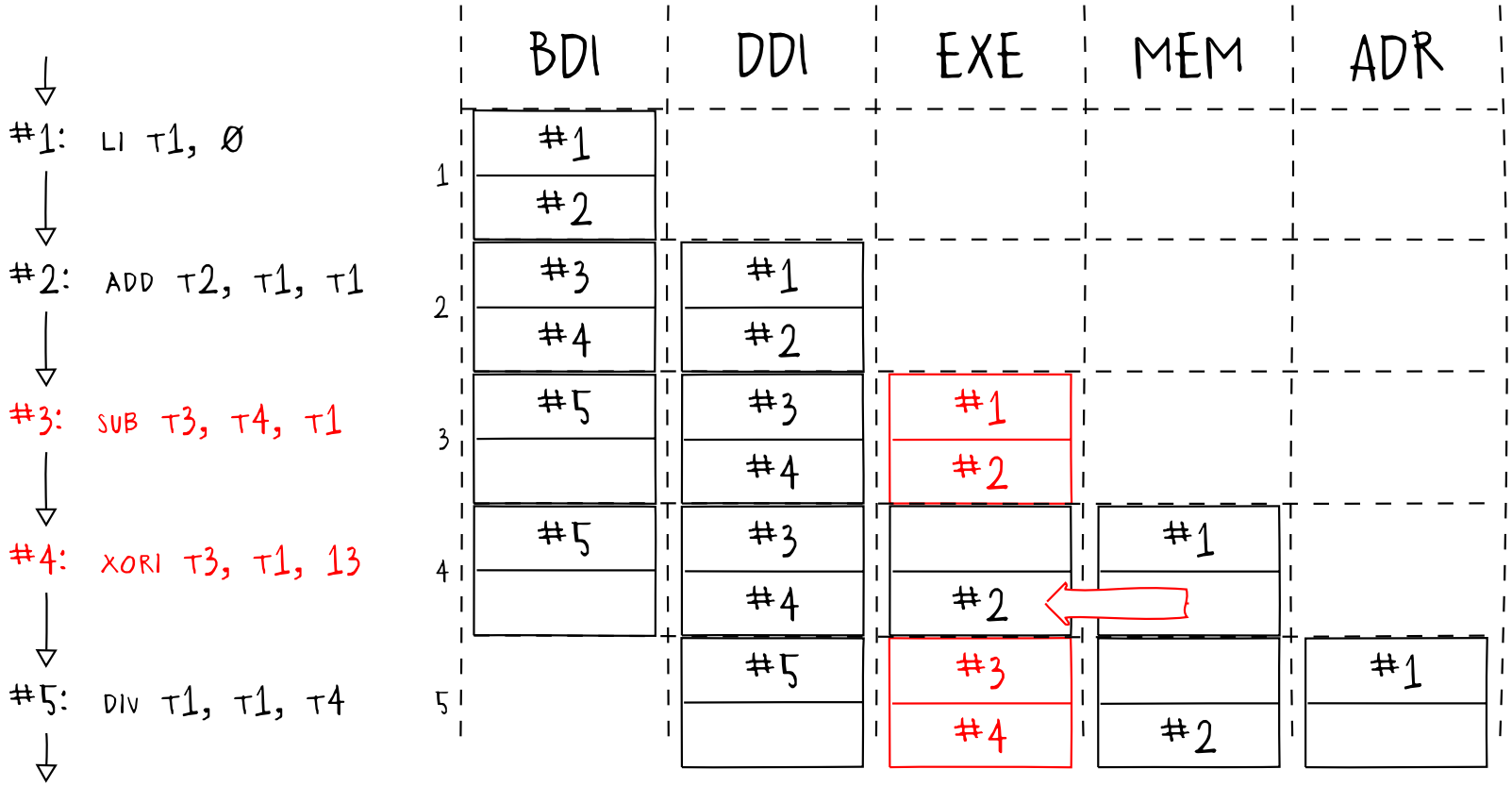
- Emissão em ordem com finalização fora de ordem
  - As instruções são executadas em ordem e o armazenamento dos resultados é feito fora de ordem



A INSTRUÇÃO #1 SEGUE PARA O PRÓXIMO ESTÁGIO, ENQUANTO QUE INSTRUÇÃO #2 FICA EM ESPERA (STALLED)

Superescalar

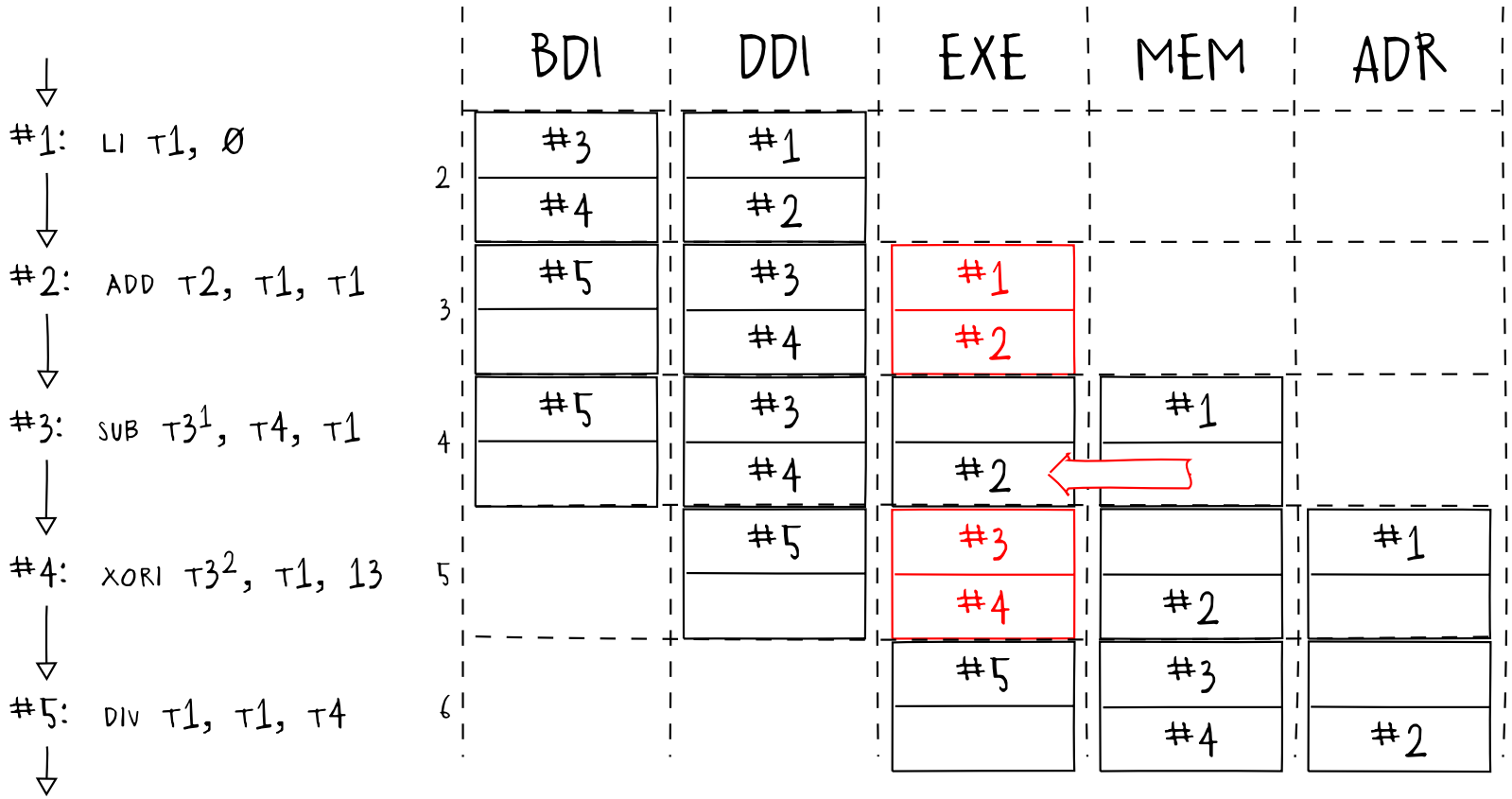
- Emissão em ordem com finalização fora de ordem
  - As instruções são executadas em ordem e o armazenamento dos resultados é feito fora de ordem



É FEITO O ADIANTAMENTO PARA A INSTRUÇÃO #2 E
É DETECTADO UM CONFLITO WAW ENTRE #3 E #4

Superescalar

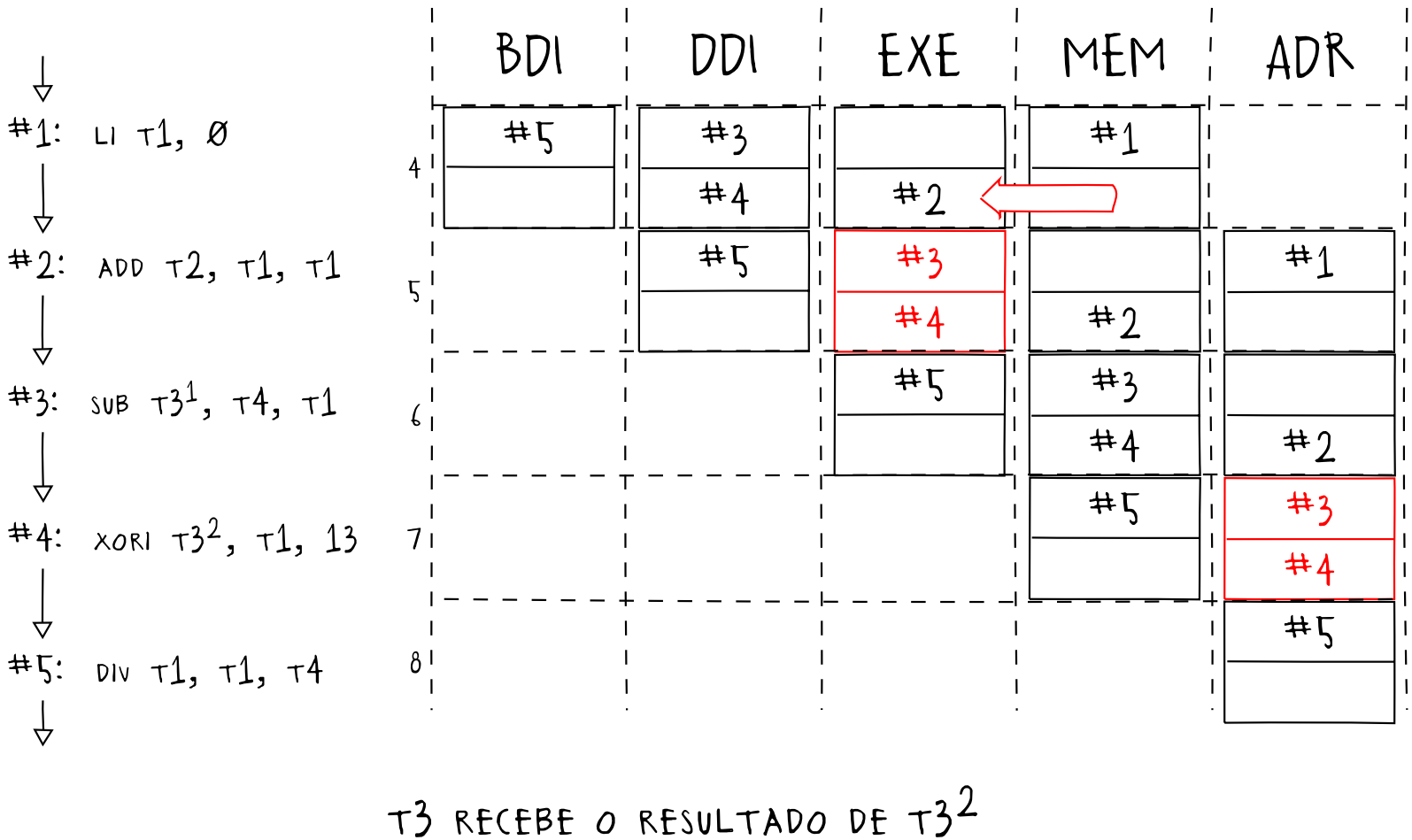
- Emissão em ordem com finalização fora de ordem
  - As instruções são executadas em ordem e o armazenamento dos resultados é feito fora de ordem



AO INVÉS DE SEQUENCIALIZAR #3 E #4,
É FEITO O RENOMEAMENTO DE T3 (WAW)

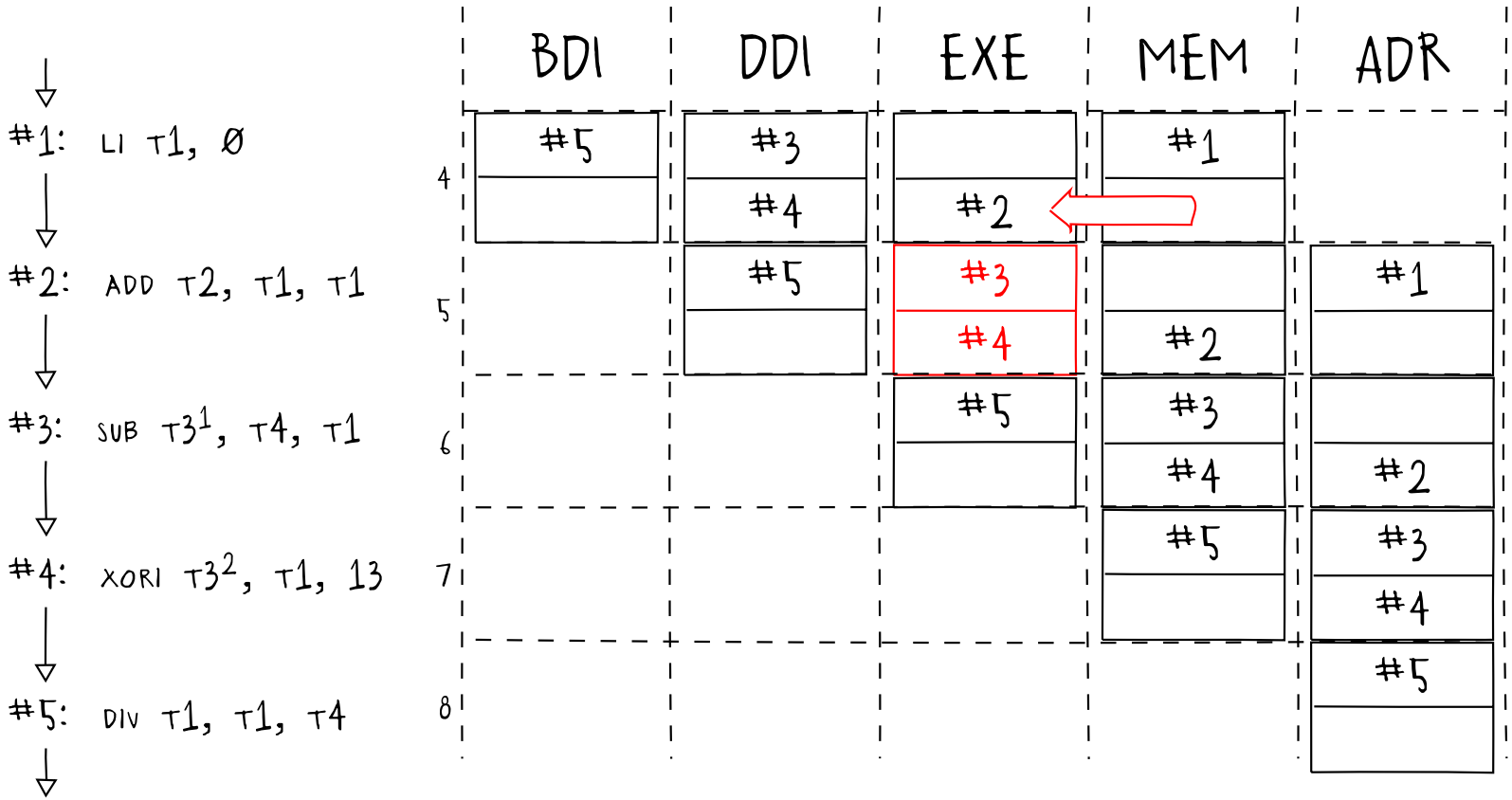
Superescalar

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



Superescalar

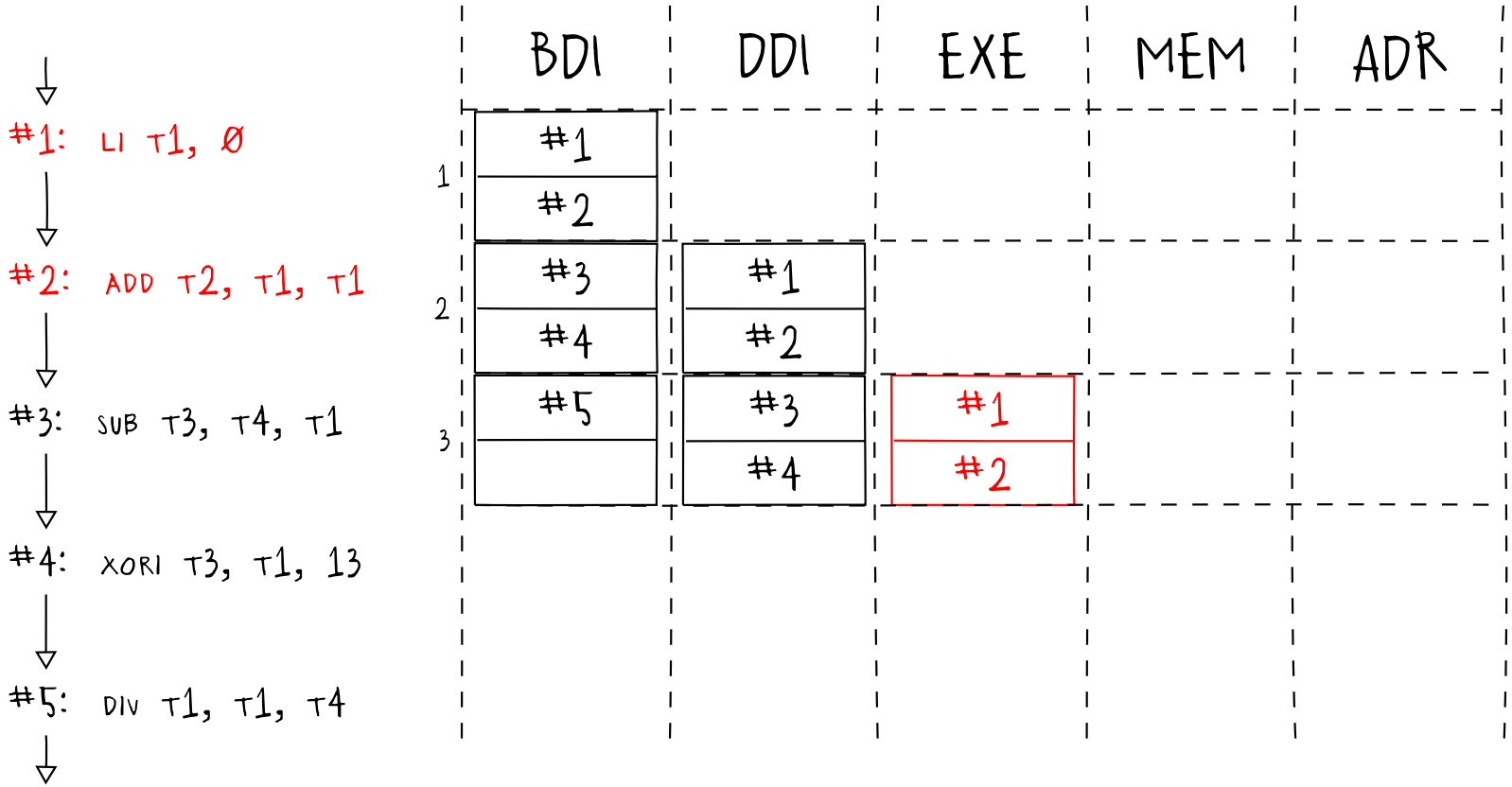
- Emissão em ordem com finalização fora de ordem
  - As instruções são executadas em ordem e o armazenamento dos resultados é feito fora de ordem



A EXECUÇÃO É FINALIZADA EM 8 CICLOS

Superescalar

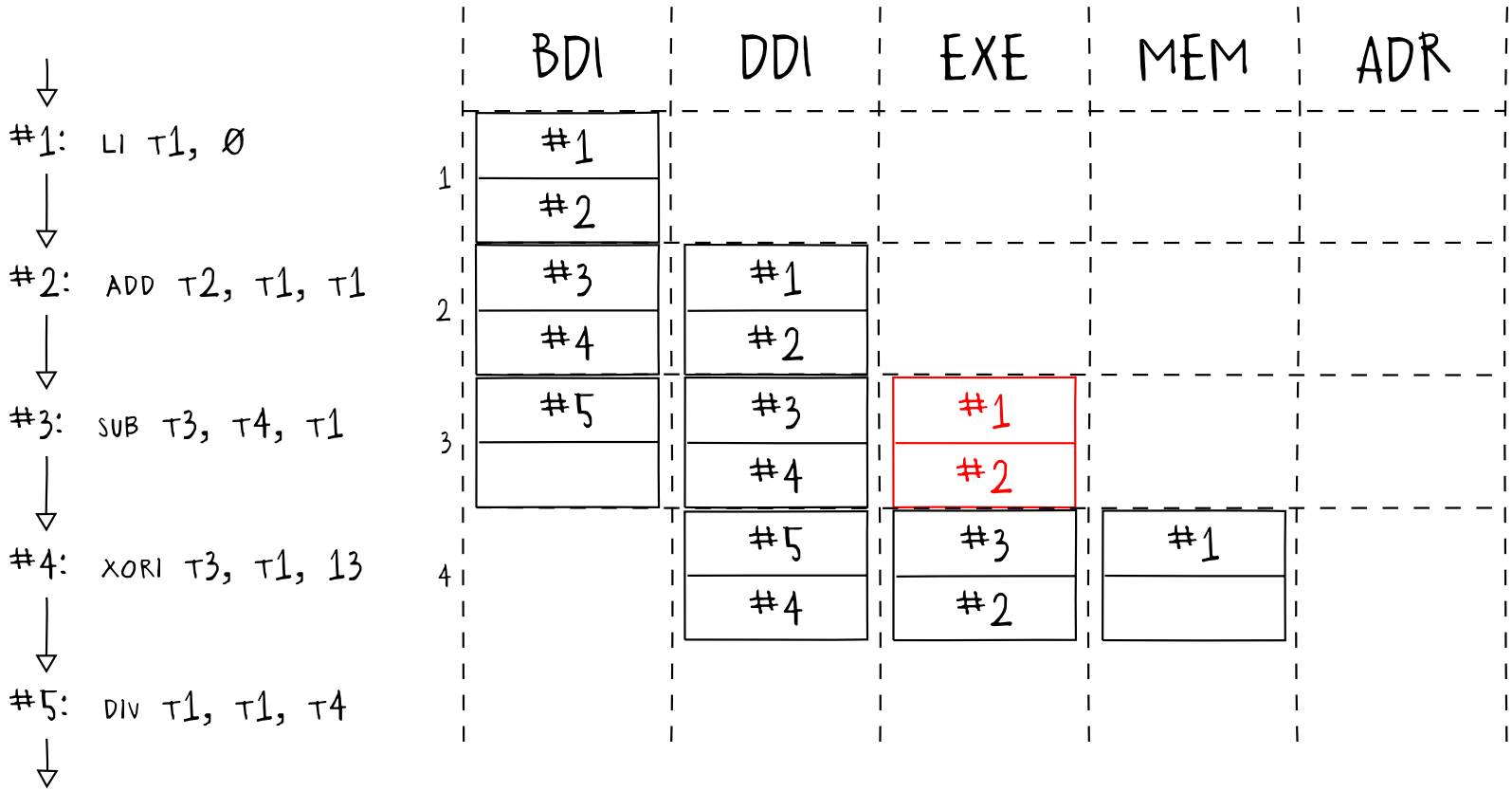
- Emissão e finalização fora de ordem
  - São utilizadas unidades de reserva para suportar a execução fora de ordem das instruções



A INSTRUÇÃO #1 TEM COMO SAÍDA T1 QUE É ENTRADA PARA INSTRUÇÃO #2 (CONFLITO RAW)

Superescalar

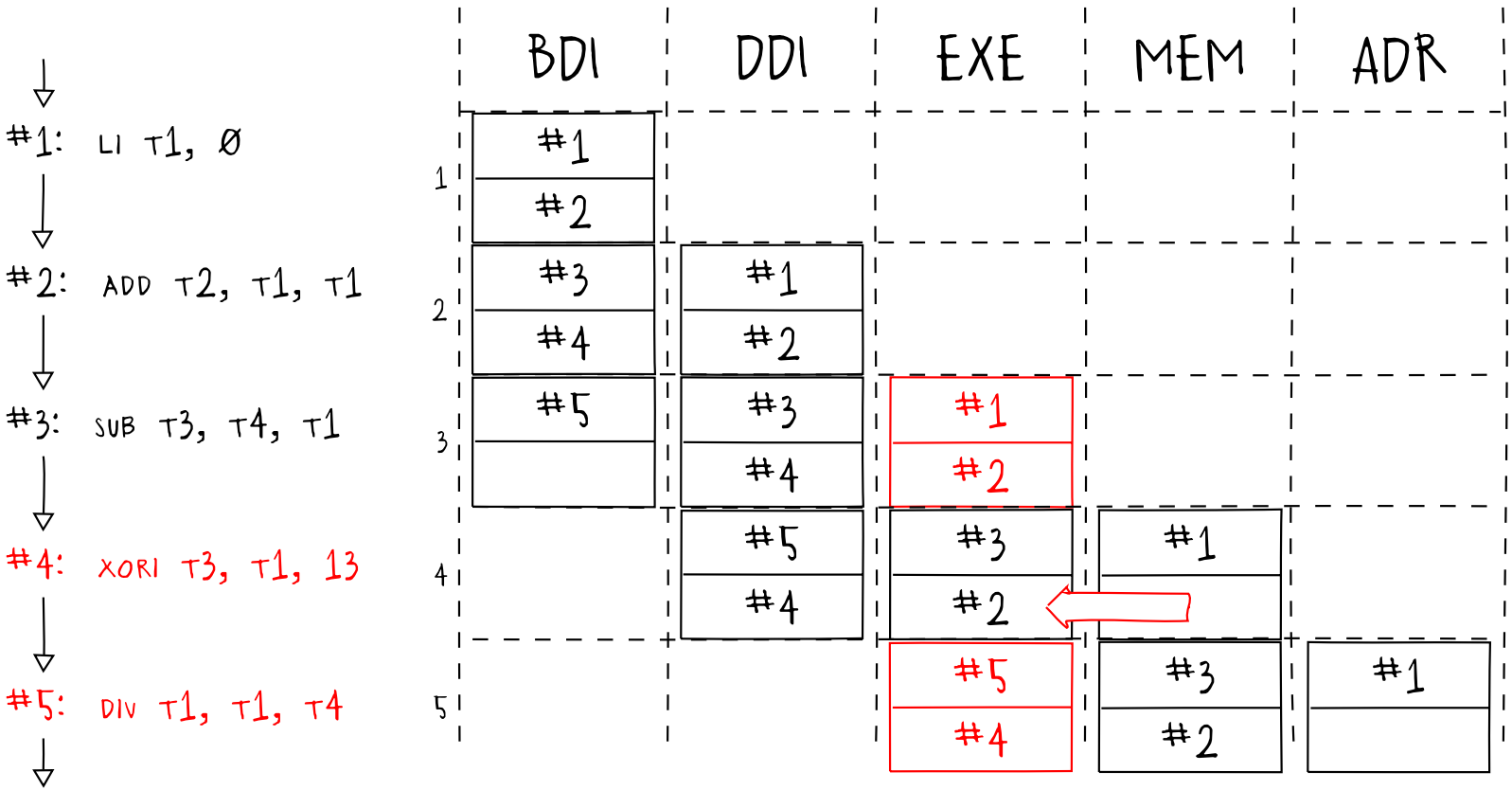
- Emissão e finalização fora de ordem
  - São utilizadas unidades de reserva para suportar a execução fora de ordem das instruções



A INSTRUÇÃO #2 FICA EM ESPERA (STALLED)
E AS INSTRUÇÕES #3 E #5 SÃO EMITIDAS FORA DE ORDEM

Superescalar

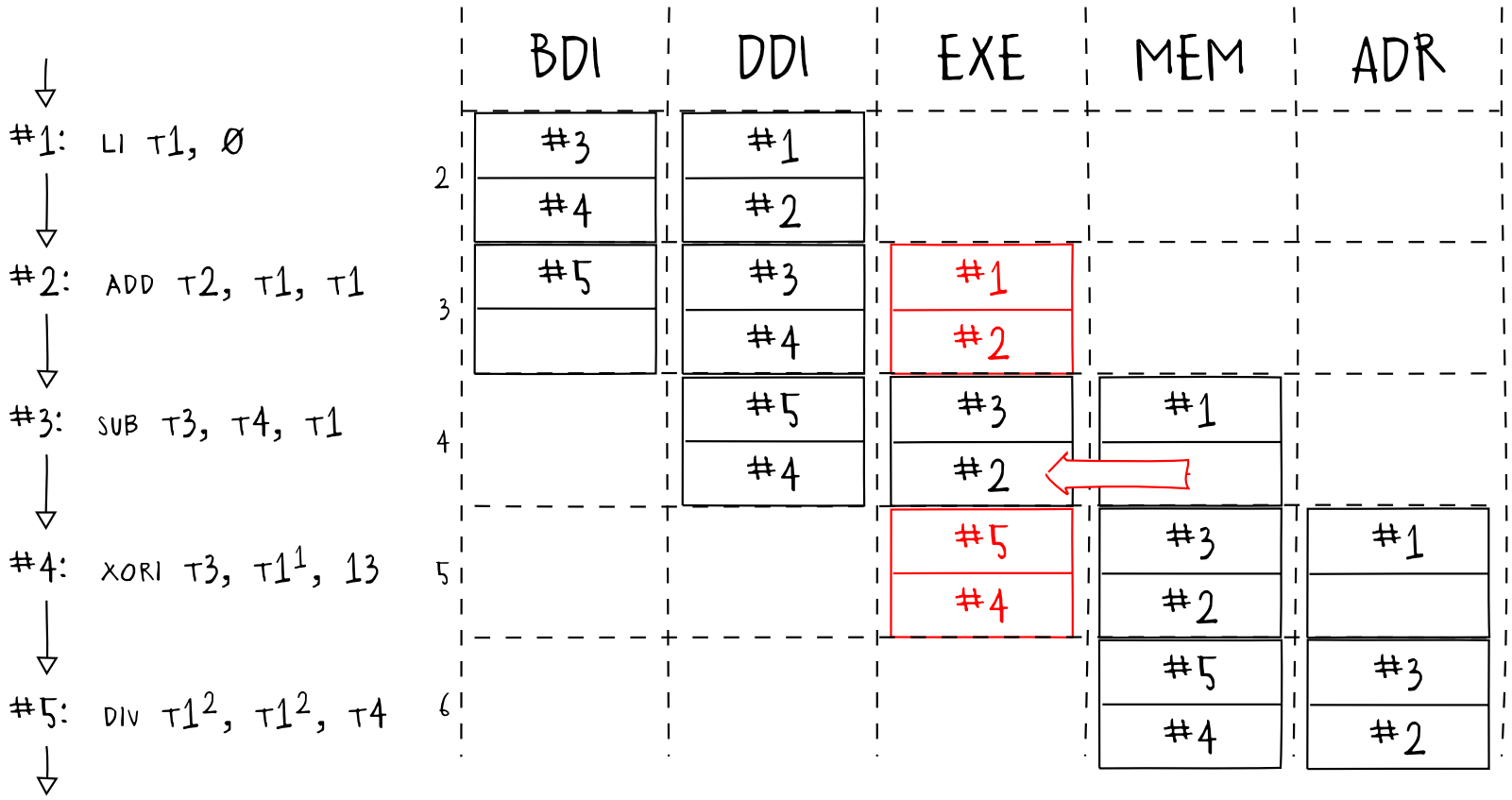
- Emissão e finalização fora de ordem
  - São utilizadas unidades de reserva para suportar a execução fora de ordem das instruções



É FEITO O ADIANTAMENTO PARA A INSTRUÇÃO #2 E É DETECTADO UM CONFLITO WAR ENTRE #5 E #4

Superescalar

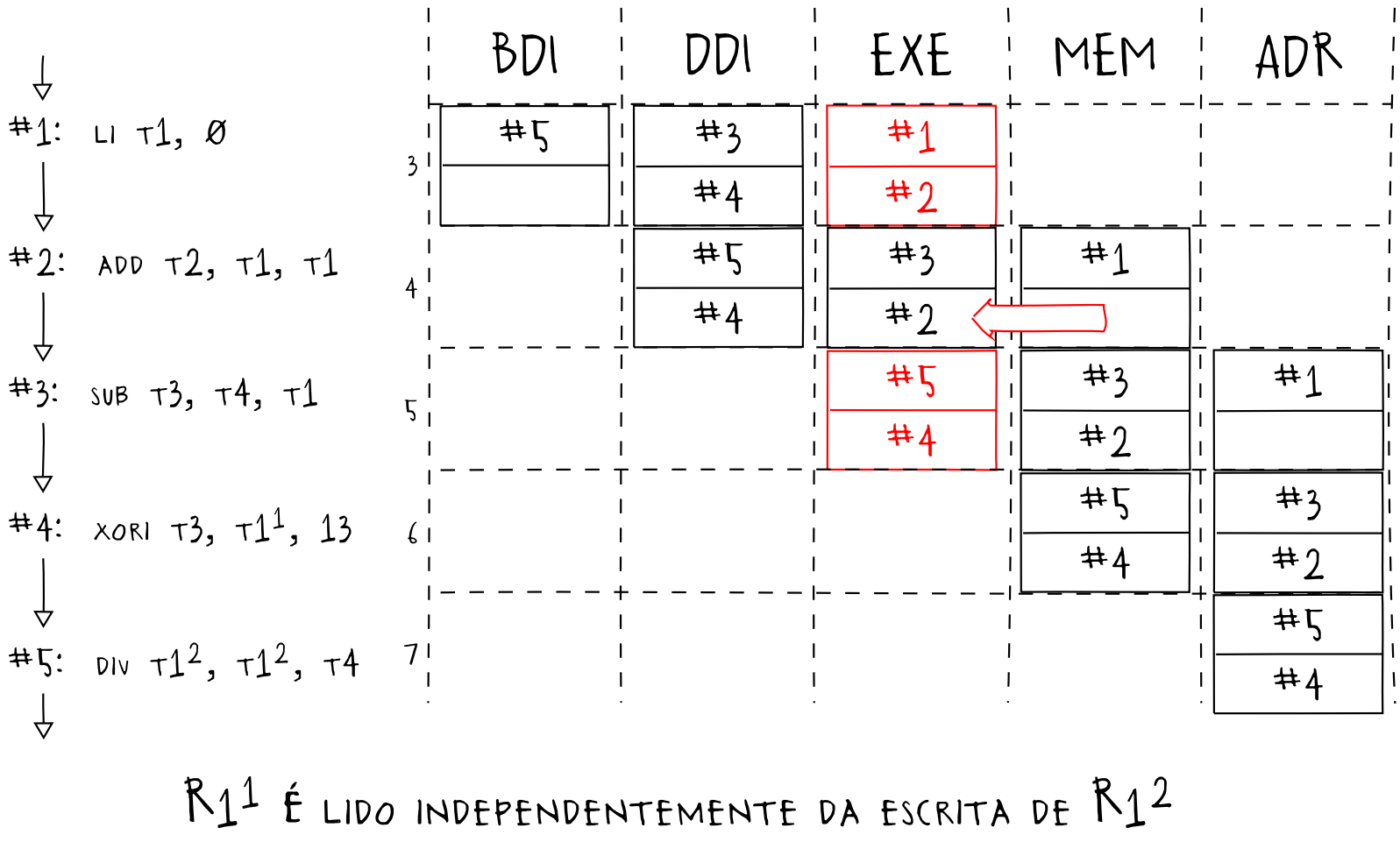
- Emissão e finalização fora de ordem
  - São utilizadas unidades de reserva para suportar a execução fora de ordem das instruções



AO INVÉS DE SEQUENCIALIZAR #5 E #4,
É FEITO O RENOMEAMENTO DE T1 (WAR)

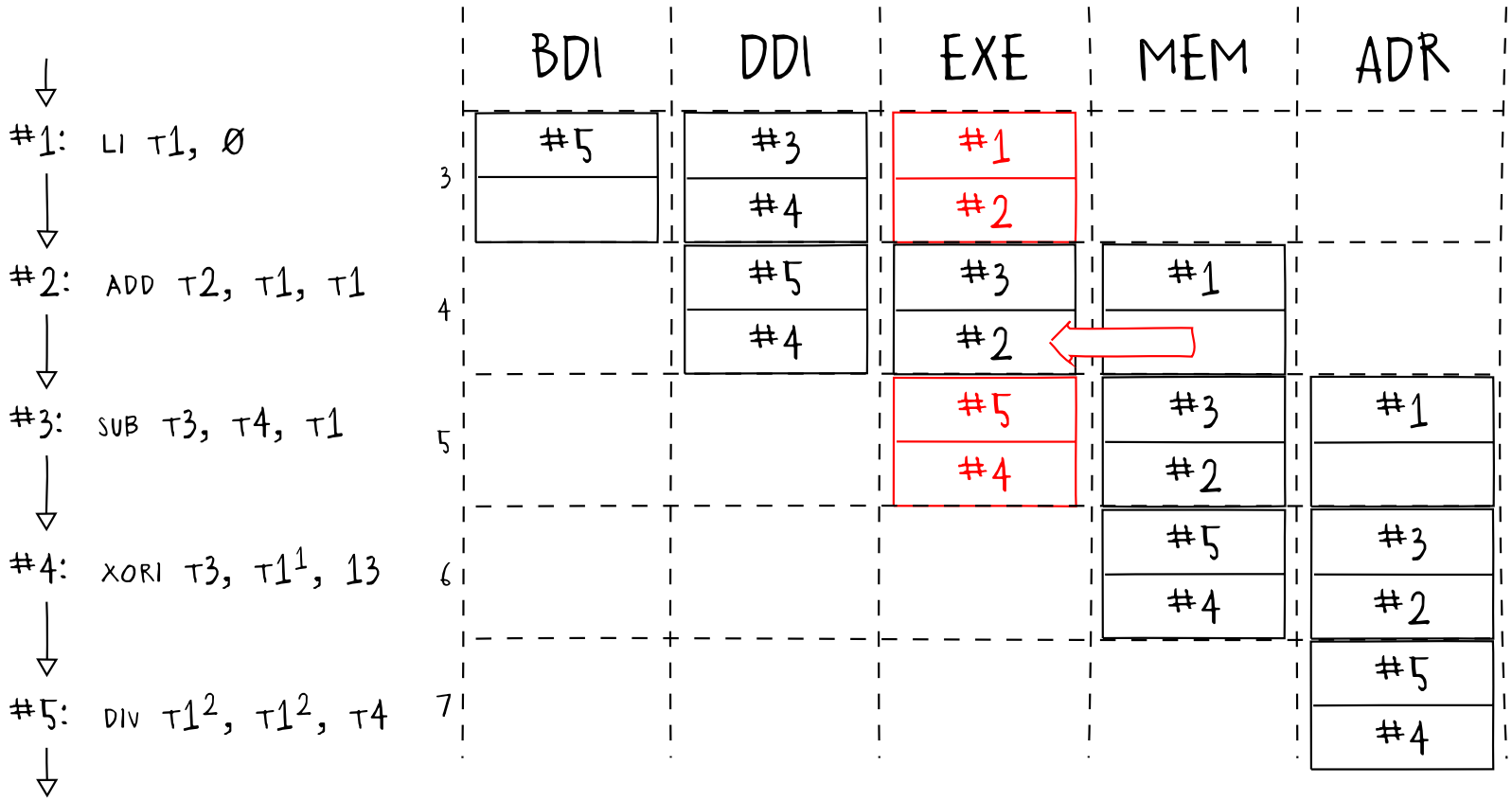
Superescalar

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



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*): replica as operações de cada iteração, removendo as instruções de desvio, entretanto, a quantidade de iterações deve ser previamente conhecida

Superescalar

- Técnicas estáticas de compilação do software para explorar o paralelismo de instruções
  - Desenrolamento de laço (*loop unrolling*): replica as operações de cada iteração, removendo as instruções de desvio, entretanto, a quantidade de iterações deve ser previamente conhecida
  - *Very Long Instruction Word* (VLIW): escalona as instruções em um pacote de tamanho fixo, sem dependências entre as operações e que podem ser executadas paralelamente

Superescalar

- Desenrolamento de laços (*loop unrolling*)

```
1 // Multiplicação escalar de vetor
2 void mulsv(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 sem otimização

```
1  # mulsv(a0 = 7, a1 = 0x80008000, a2 = 100)
2  mulsv:
3      init:
4          # i = t0 = 0
5          li t0, 0
6      loop:
7          # i < n
8          bge t0, a2, end
9          # V[i] = k * V[i]
10         sra t1, t0, 2
11         add t1, t1, a1
12         lw t2, 0(t1)
13         mul t2, a0, t2
14         sw t2, 0(t1)
15         # i++
16         addi t0, t0, 1
17         j loop
18     end:
19         ret
```

Superescalar

- Desenrolamento de laços (*loop unrolling*)
  - Código de montagem com otimização

```
1   # mulsv(a0 = 7, a1 = 0x80008000, a2 = 100)
2   mulsv:
3       loop0:
4           # V[0] = k * V[0]
5           lw t0, 0(a1)
6           mul t0, a0, t0
7           sw t0, 0(a1)
…       ...
498     loop99:
499         # V[99] = k * V[99]
500         lw t0, 396(a1)
501         mul t0, a0, t0
502         sw t0, 396(a1)
503     ret
```

As instruções de desvio são eliminadas,
porém são emitidas muito mais instruções

Superescalar

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

```
1  lw t0, 0(a0)
2  lw t1, 4(a0)
3  lw t2, 8(a0)
4  lw t3, 12(a0)
5  xori t0, t0, 3
6  xori t1, t1, 5
7  xori t2, t2, 8
8  xori t3, t3, 13
9  sw t0, 0(a0)
10 sw t1, 4(a0)
11 sw t2, 8(a0)
12 sw t3, 12(a0)
```

$\Longrightarrow$

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 para explorar o paralelismo, com utilização de instruções *nop* quando não conseguir preencher o pacote

Superescalar

- *Very Long Instruction Word* (VLIW)
  - Aumento do tamanho do código gerado para explorar o paralelismo, com utilização de instruções *nop* quando não conseguir preencher o pacote
  - Problemas de incompatibilidade binária em famílias de processadores compatíveis, mas com diferentes organizações e quantidade de unidades de processamento

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
    - 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 obtidos

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 obtidos
  - 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 mais robusto pela redundância
    - Em oposição à exploração do paralelismo implícito entre as instruções, o multiprocessamento depende que o software seja paralelizado para utilizar os núcleos de processamento