Conflitos de execução

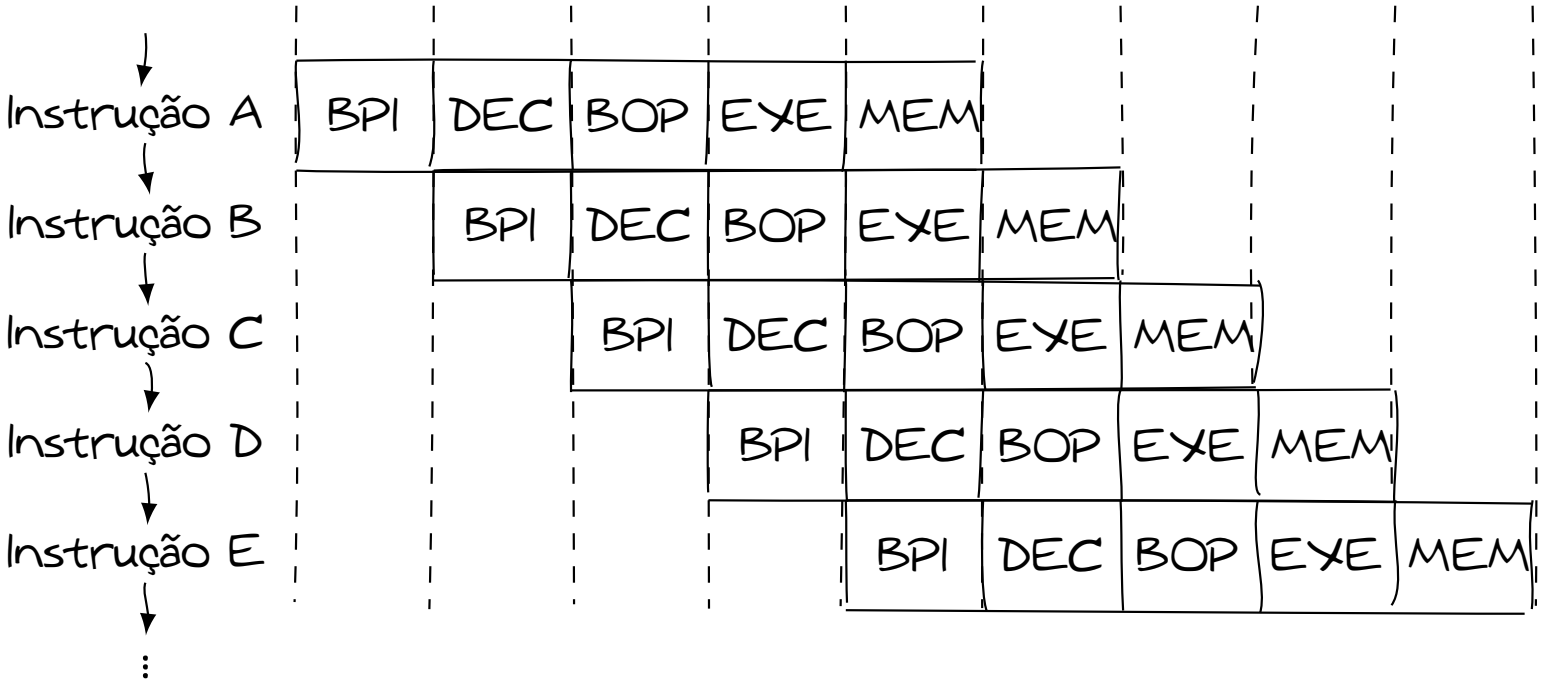
Arquitetura de Computadores

Bruno Prado

Departamento de Computação / UFS

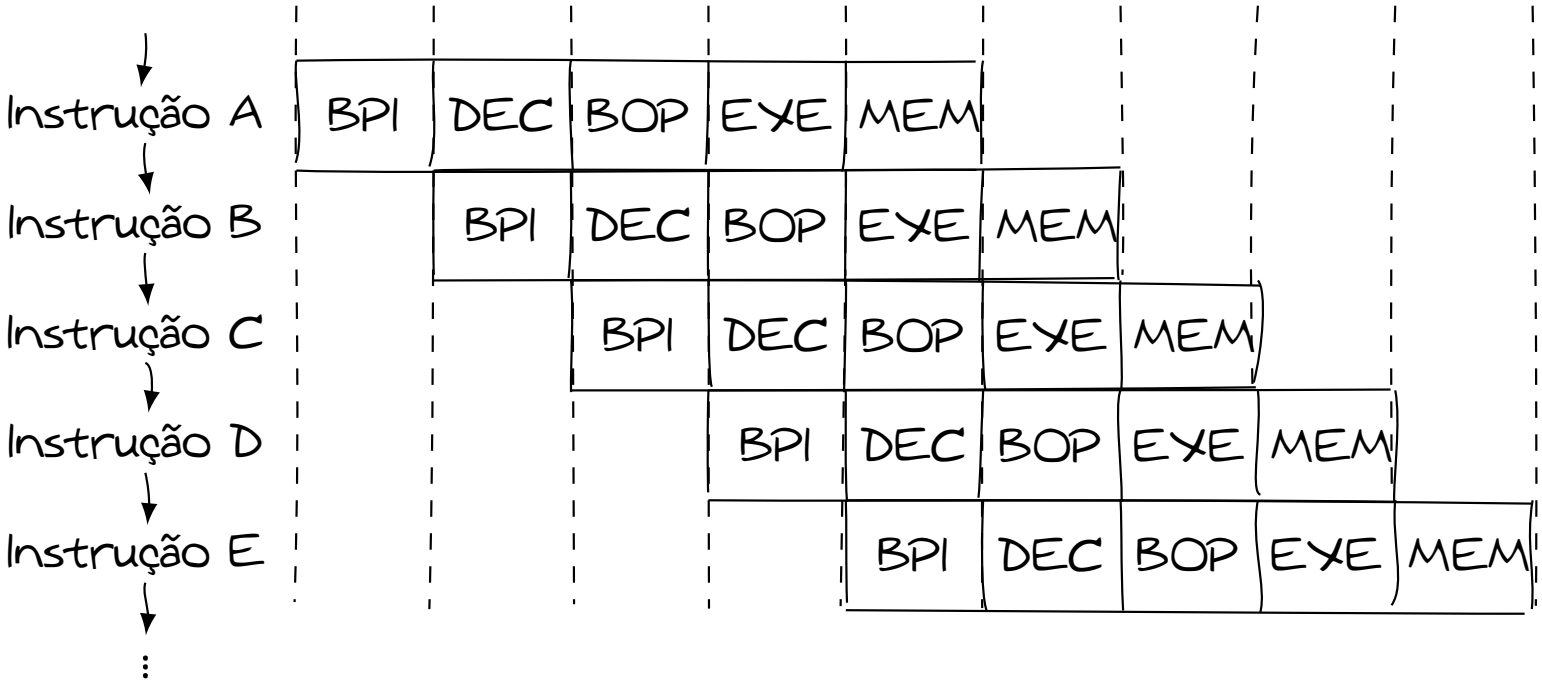
Introdução

- Implementação de arquitetura com *pipeline*
  - As instruções são executadas de forma concorrente e sobreposta nos estágios do processador



Introdução

- Implementação de arquitetura com *pipeline*
  - As instruções são executadas de forma concorrente e sobreposta nos estágios do processador



✓ Maximização da taxa de execução

✓ Melhor aproveitamento do hardware

Introdução

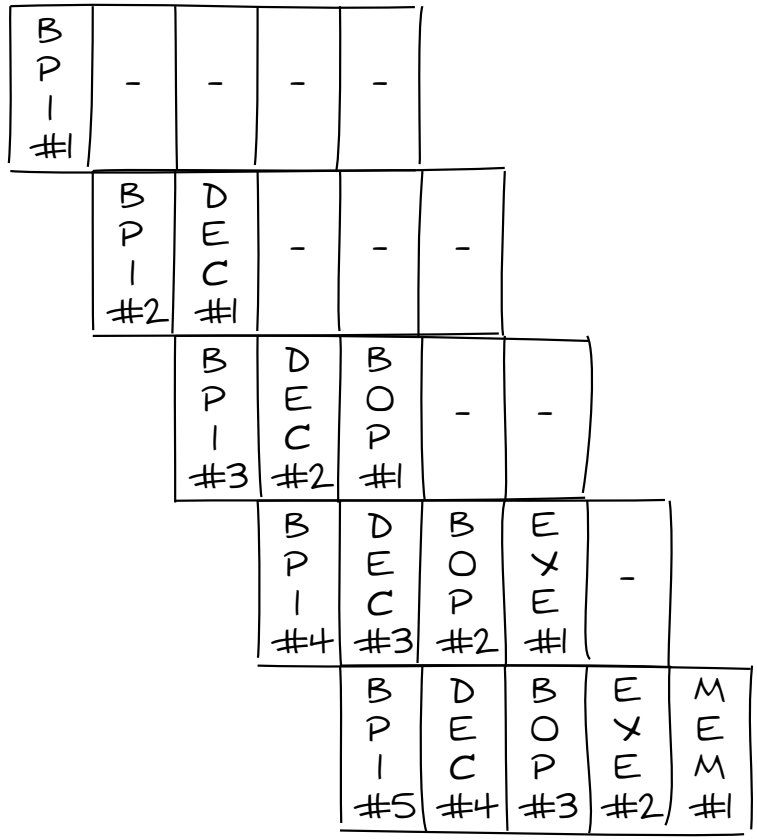
- Cenários de execução em *pipeline*
  - Ideal
    - O desempenho do processador é multiplicado pelo número de estágios utilizados no *pipeline*
    - Não existem conflitos na execução das instruções e a taxa de execução só depende da frequência de operação e da quantidade de estágios

Introdução

- Cenários de execução em *pipeline*
  - Ideal
    - O desempenho do processador é multiplicado pelo número de estágios utilizados no *pipeline*
    - Não existem conflitos na execução das instruções e a taxa de execução só depende da frequência de operação e da quantidade de estágios
  - Real
    - O desempenho é variável, sendo diretamente afetado pela sequência de instruções executadas
    - Como podem existir conflitos que precisam ser tratados, o desempenho do sistema é reduzido para manter o comportamento sequencial das instruções

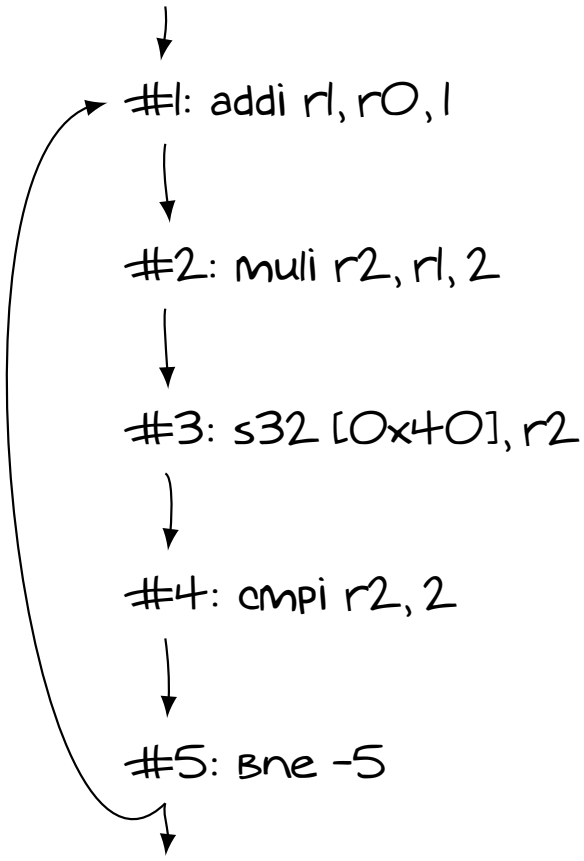
Introdução

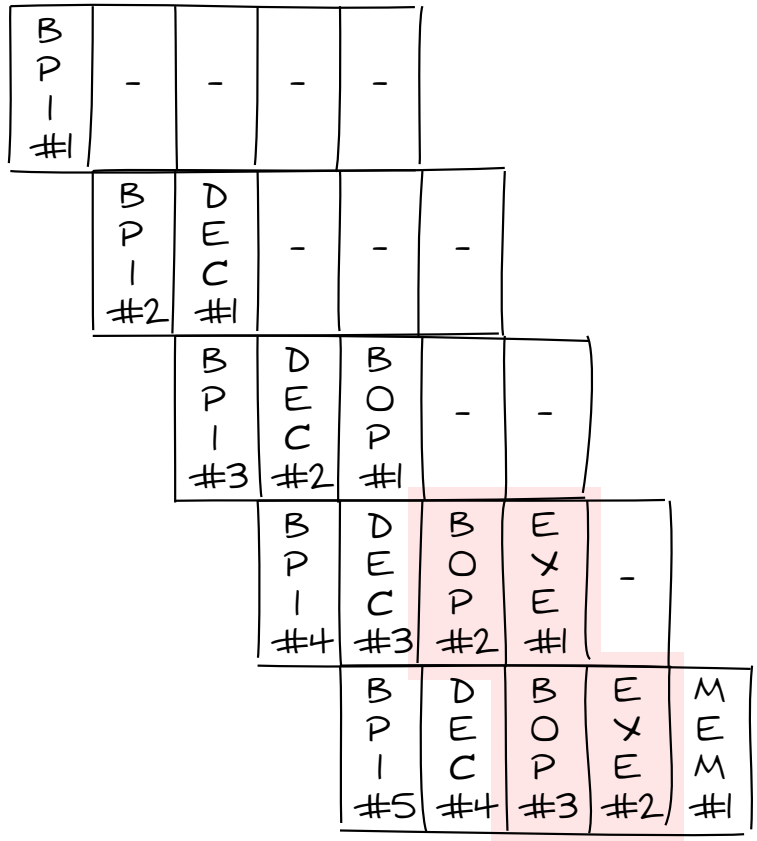
- O que são conflitos de execução no *pipeline*?
  - São situações onde uma instrução não pode executar no próximo estágio do *pipeline*
  - A resolução destes conflitos busca manter o comportamento sequencial esperado



Introdução

- O que são conflitos de execução no *pipeline*?
  - São situações onde uma instrução não pode executar no próximo estágio do *pipeline*
  - A resolução destes conflitos busca manter o comportamento sequencial esperado

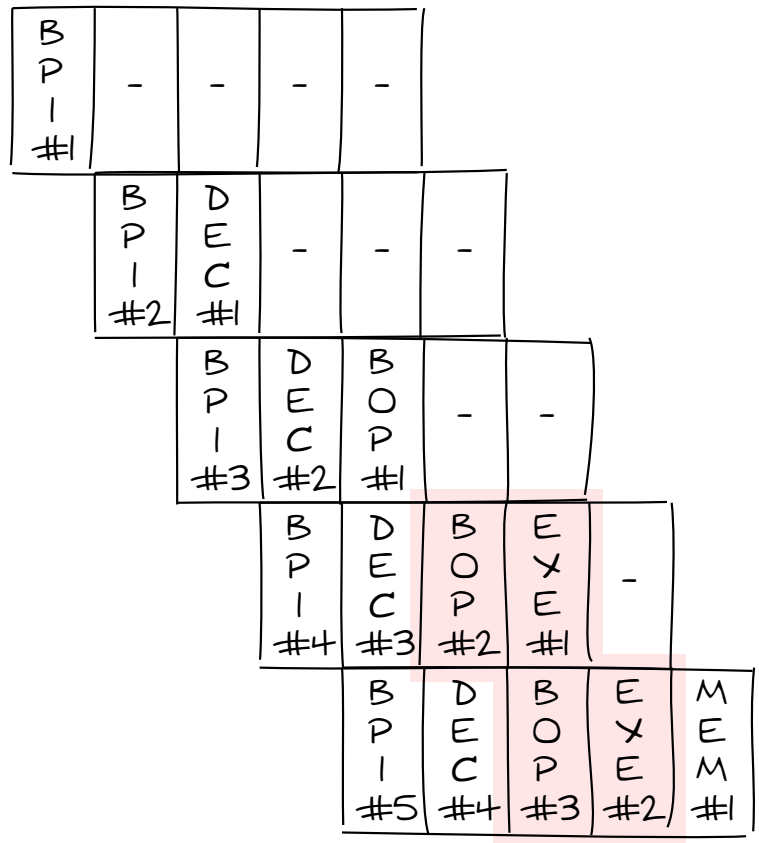




Conflito entre #1 e #2: R1 = 0 <-> 1

Introdução

- O que são conflitos de execução no *pipeline*?
  - São situações onde uma instrução não pode executar no próximo estágio do *pipeline*
  - A resolução destes conflitos busca manter o comportamento sequencial esperado



Conflito entre #2 e #3: R2 = 0 <-> 2

Introdução

- Tipos de conflitos de execução em *pipeline*
  - Estrutural
    - Limitações no projeto do processador
    - Acesso sequencial da memória para código e dados

Introdução

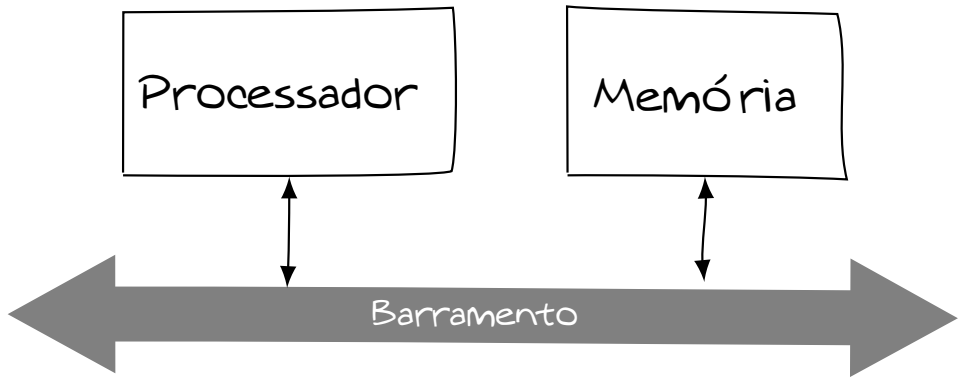
- Tipos de conflitos de execução em *pipeline*
  - Estrutural
    - Limitações no projeto do processador
    - Acesso sequencial da memória para código e dados
  - Dados
    - Dependência de dados entre instruções consecutivas
    - O comportamento sequencial deve ser preservado

Introdução

- Tipos de conflitos de execução em *pipeline*
  - Estrutural
    - Limitações no projeto do processador
    - Acesso sequencial da memória para código e dados
  - Dados
    - Dependência de dados entre instruções consecutivas
    - O comportamento sequencial deve ser preservado
  - Controle
    - Decisões baseadas em dados ainda não calculados
    - Atraso ou predição de desvio de fluxo de execução

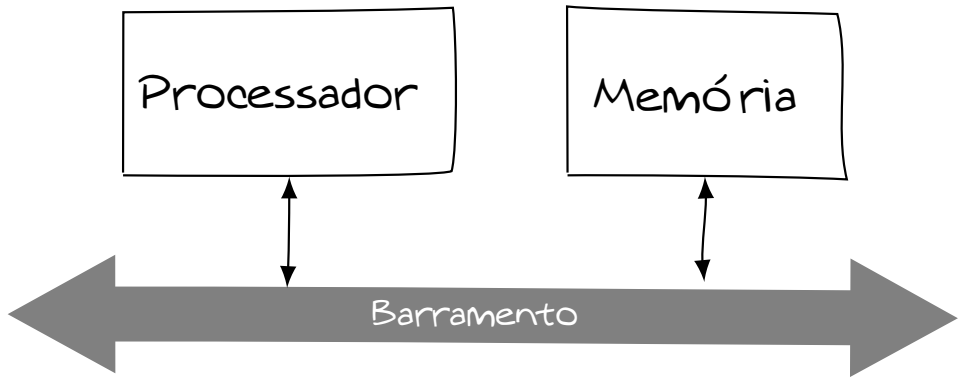
Conflito estrutural

- Limitações no projeto do sistema
  - Arquitetura Von Neumann (Princeton)



Conflito estrutural

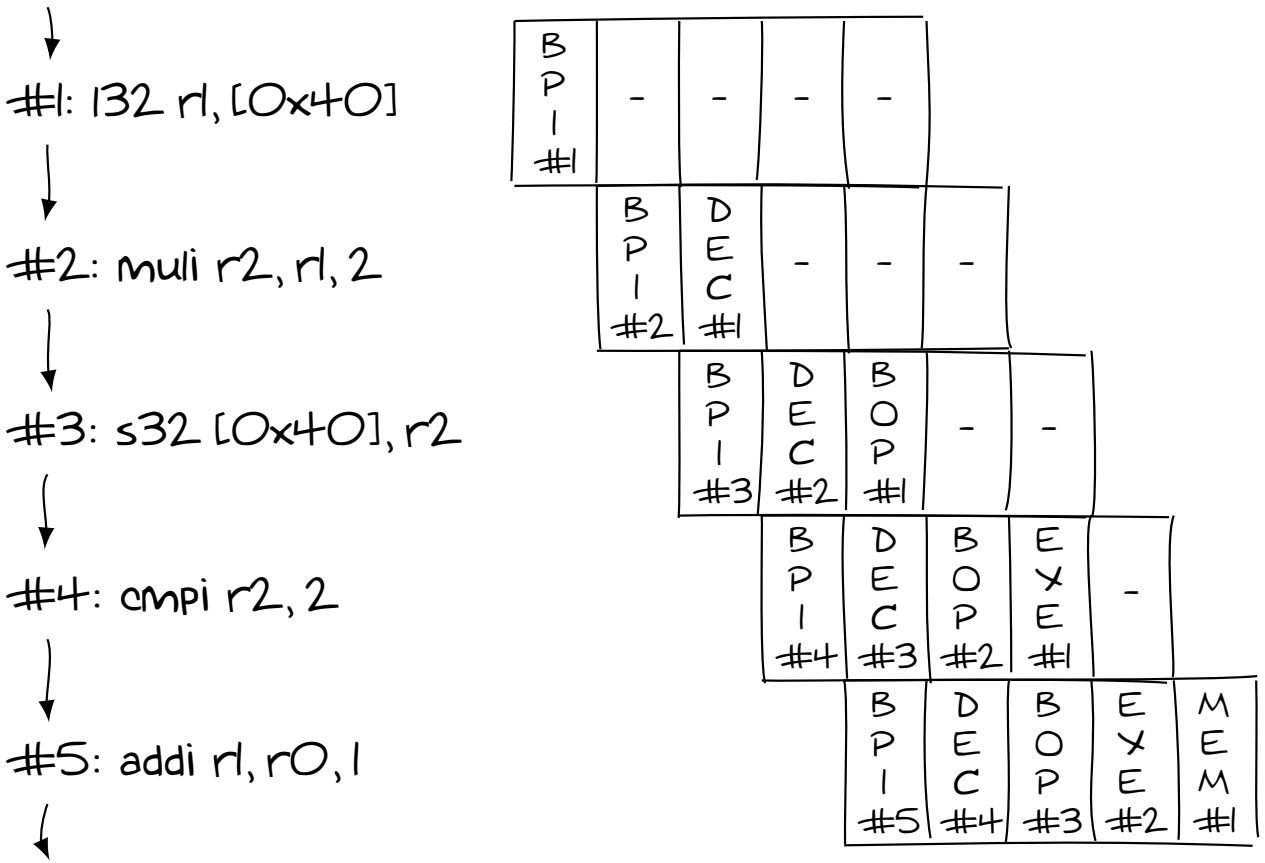
- Limitações no projeto do sistema
  - Arquitetura Von Neumann (Princeton)



Interface de memória com transações sequenciais acesso ao código e dados

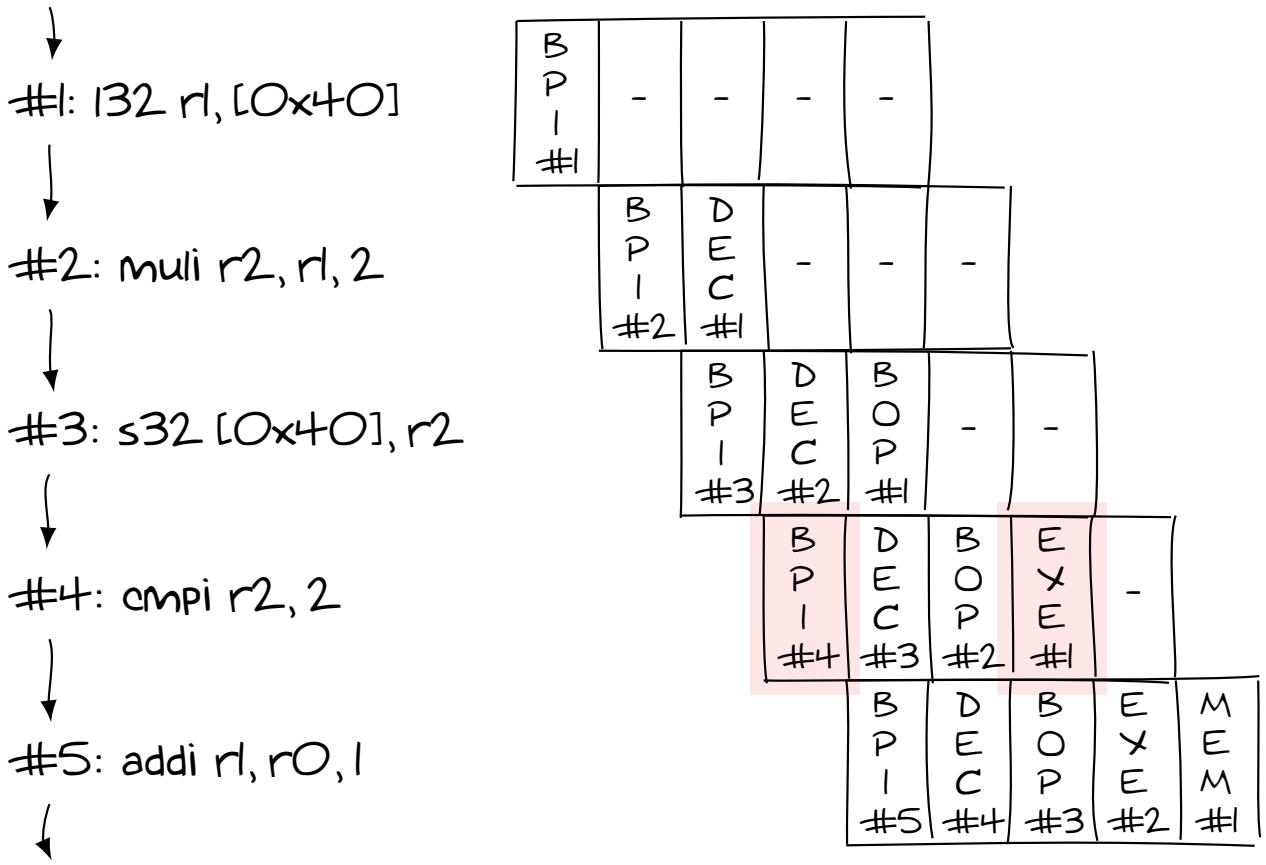
Conflito estrutural

- Limitações no projeto do sistema
  - Arquitetura Von Neumann (Princeton)



Conflito estrutural

- Limitações no projeto do sistema
  - Arquitetura Von Neumann (Princeton)



As instruções #1 e #4
acessam a memória ao mesmo tempo

Conflito estrutural

- Como resolver este conflito?

Conflito estrutural

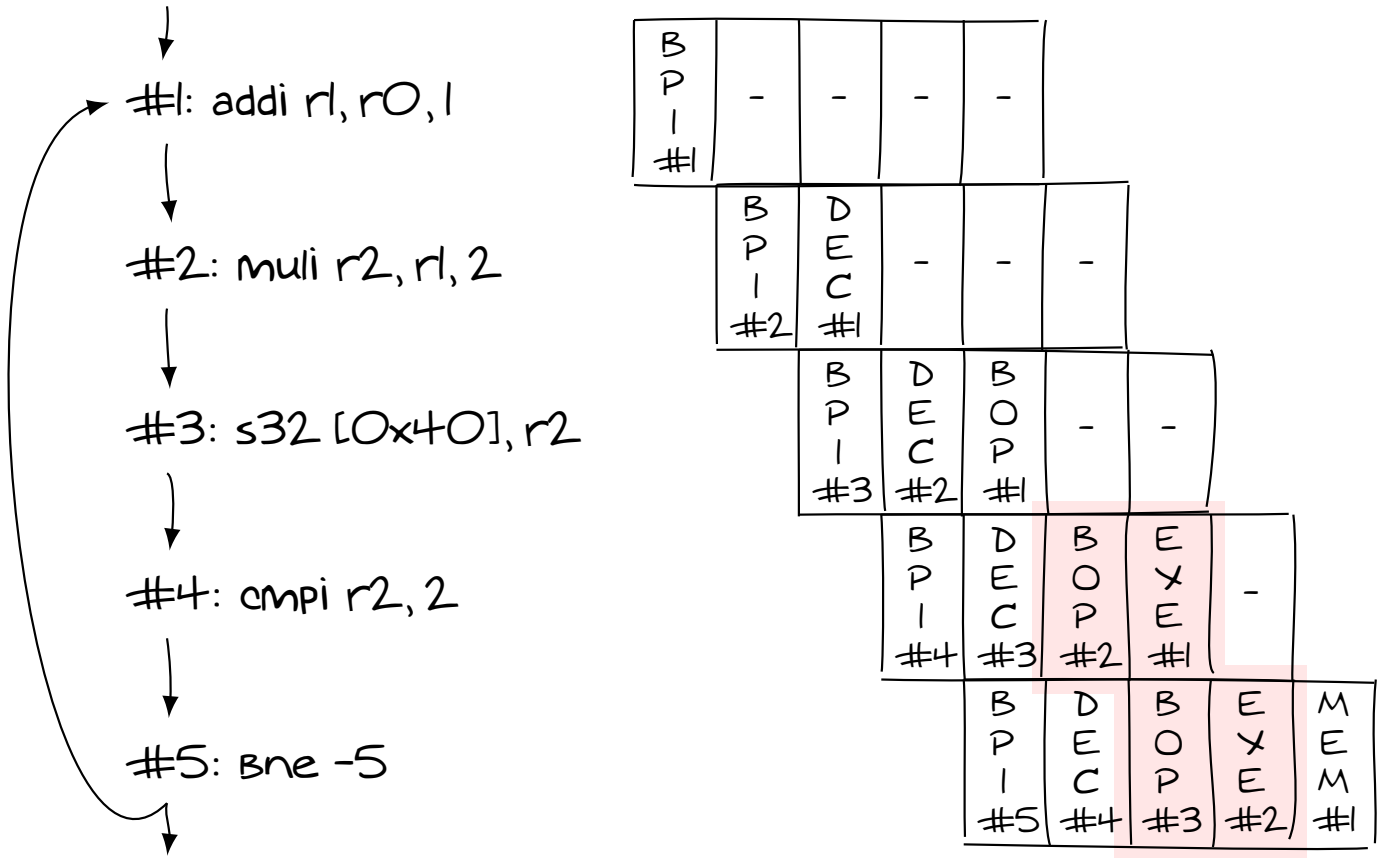
- Como resolver este conflito?
  - Sequenciamento das operações
    - Definição de qual estágio terá acesso ao barramento
    - Paralisação do *pipeline* até resolver o conflito

Conflito estrutural

- Como resolver este conflito?
  - Sequenciamento das operações
    - Definição de qual estágio terá acesso ao barramento
    - Paralisação do *pipeline* até resolver o conflito
  - Arquitetura Harvard
    - Memórias físicas separadas para código e dados
    - Acesso concorrente para instruções e dados

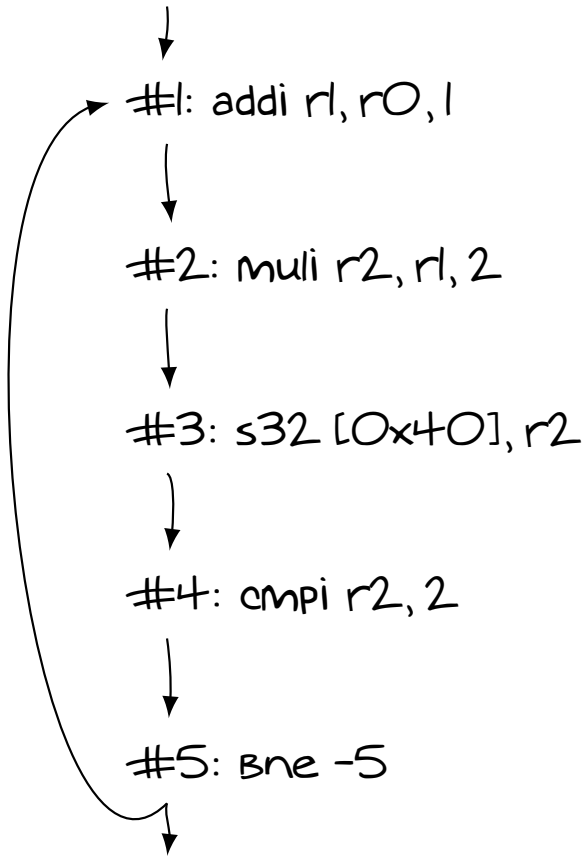
Conflito de dados

- Criado por uma dependência de dados entre instruções consecutivas executando no *pipeline*



Conflito de dados

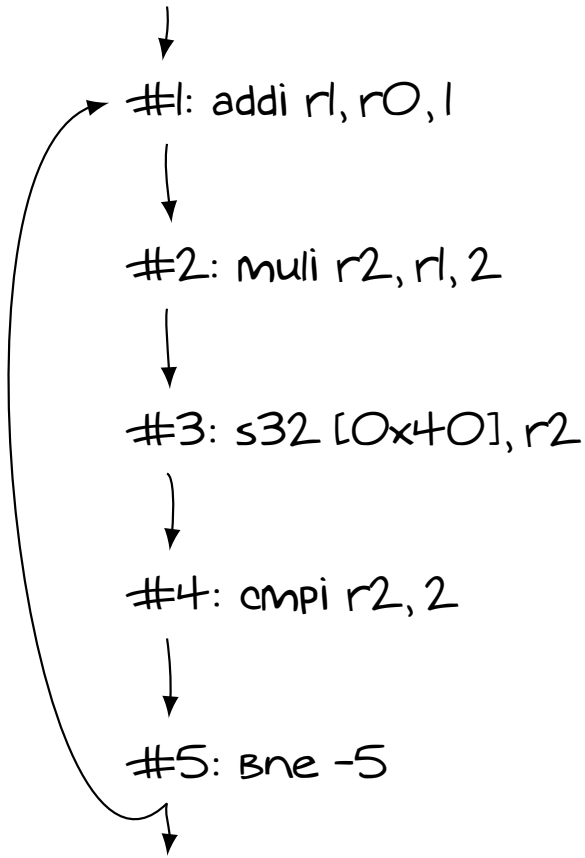
- Inserção de atrasos ou bolhas no *pipeline*

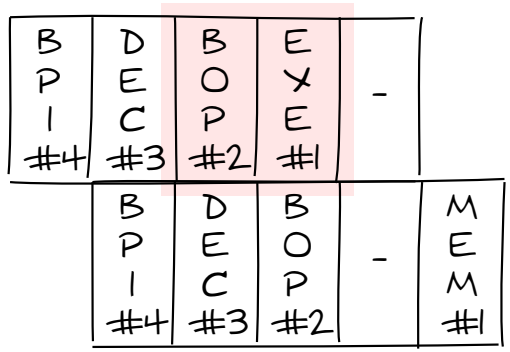


BP	DEC	BOP	EXE	
#4	#3	#2	#1	-

Conflito de dados

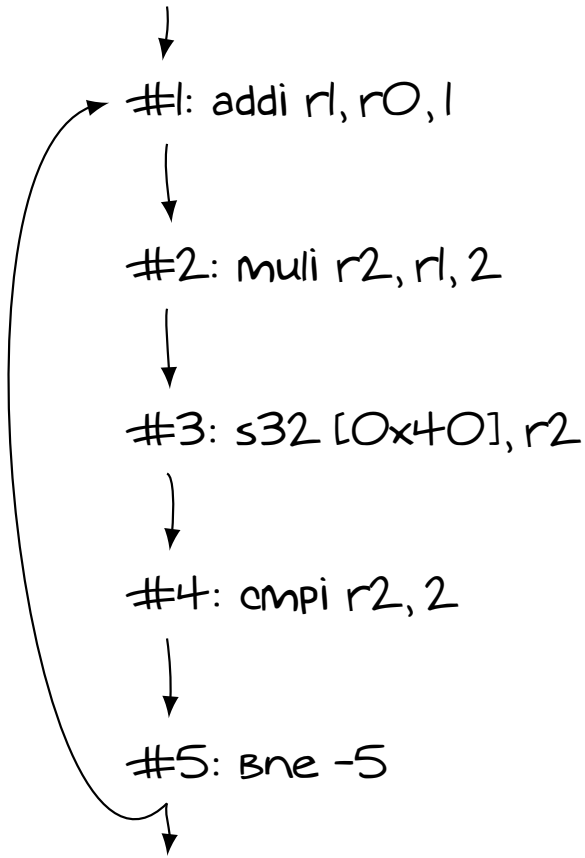
- Inserção de atrasos ou bolhas no *pipeline*

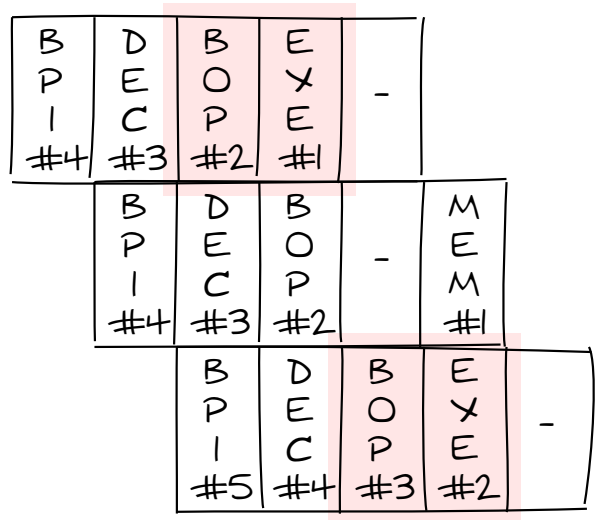




Conflito de dados

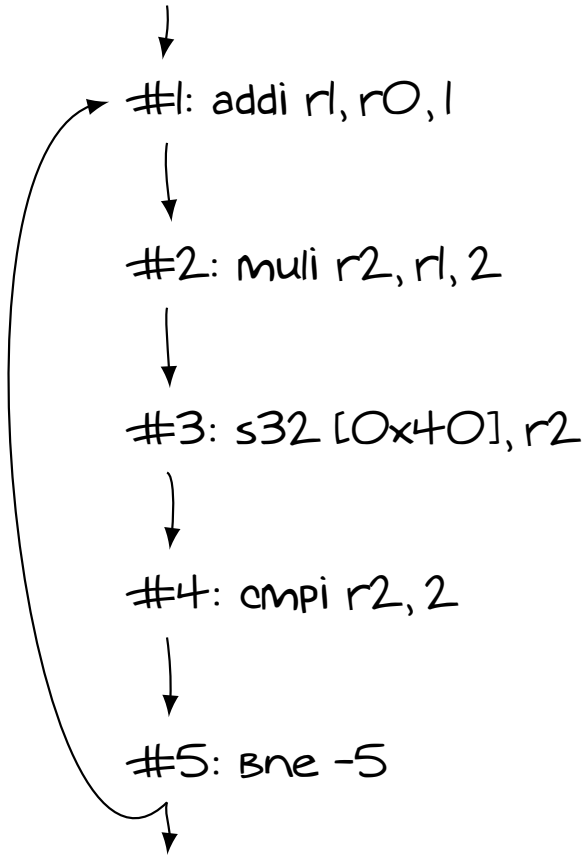
- Inserção de atrasos ou bolhas no *pipeline*

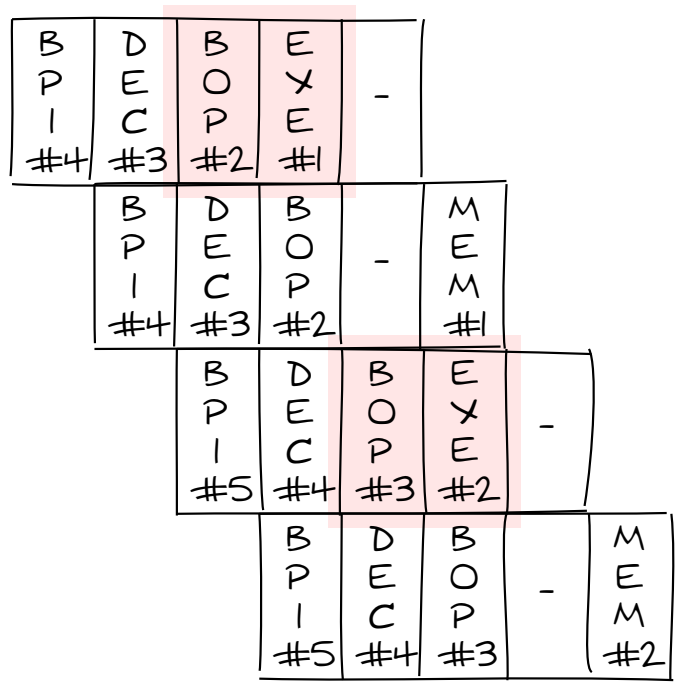




Conflito de dados

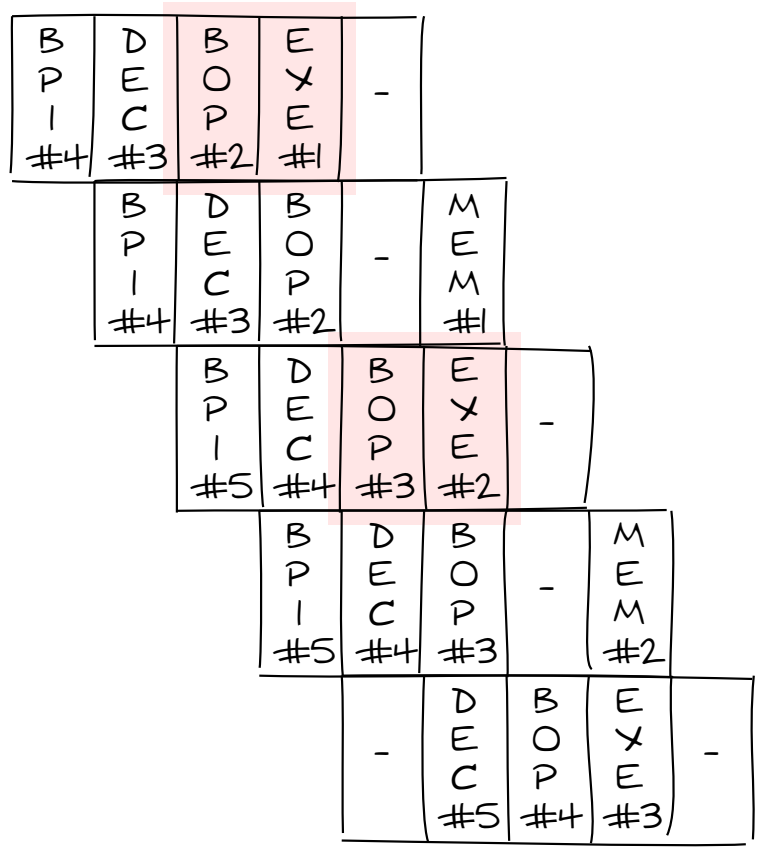
- Inserção de atrasos ou bolhas no *pipeline*





Conflito de dados

- Inserção de atrasos ou bolhas no *pipeline*

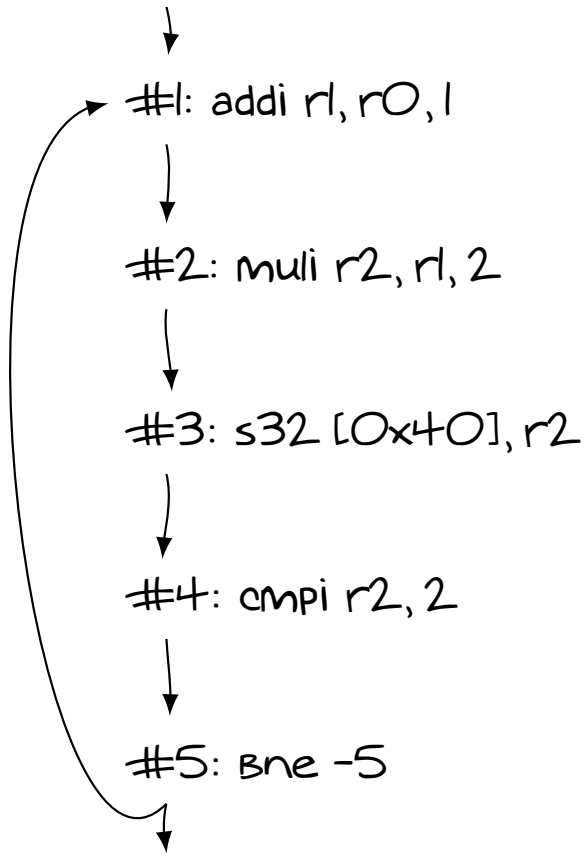


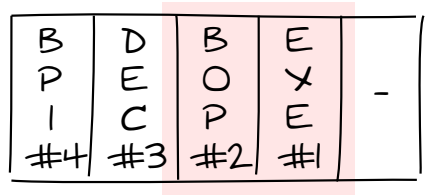
Conflito de dados

- Impacto da inserção de atrasos ou bolhas
  - ✓ É uma solução simples de implementar
  - ✗ Não é eficiente, reduzindo o desempenho

Conflito de dados

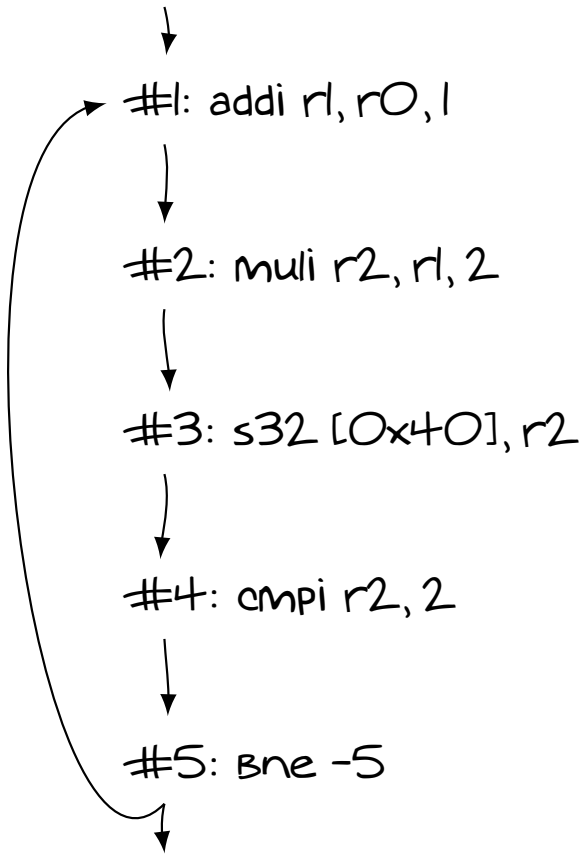
- Adiantamento de dados em registrador

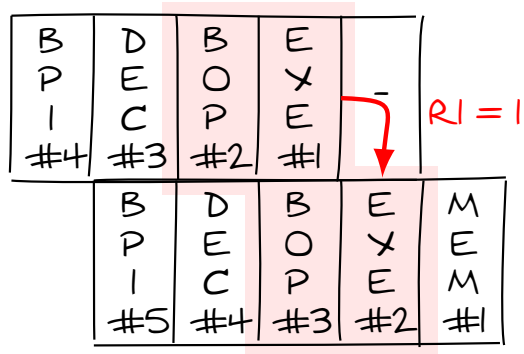




Conflito de dados

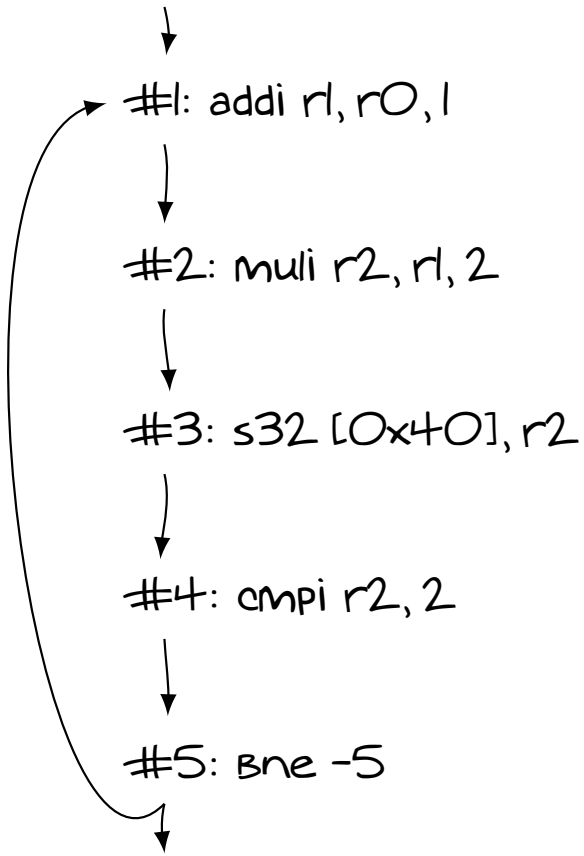
- Adiantamento de dados em registrador

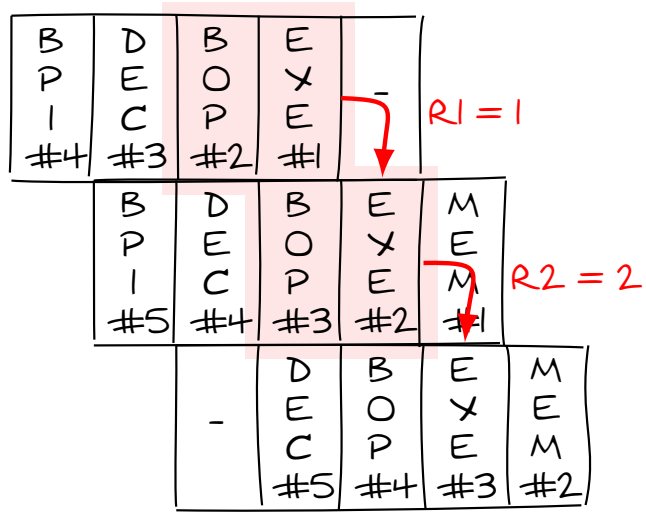




Conflito de dados

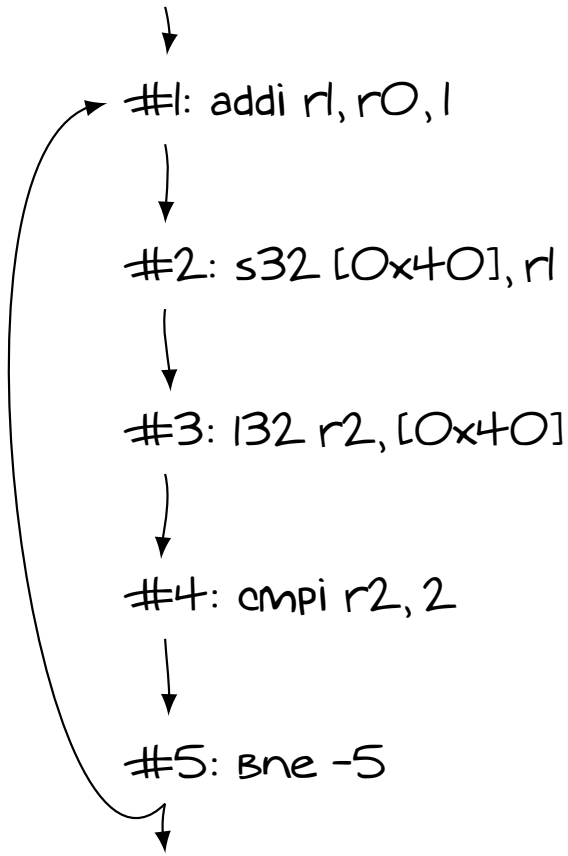
- Adiantamento de dados em registrador





Conflito de dados

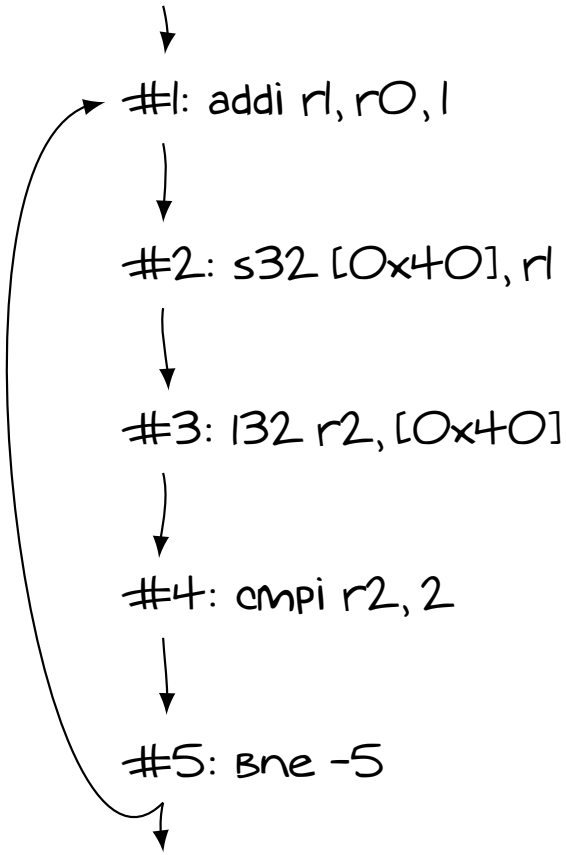
- Adiantamento de dados em memória

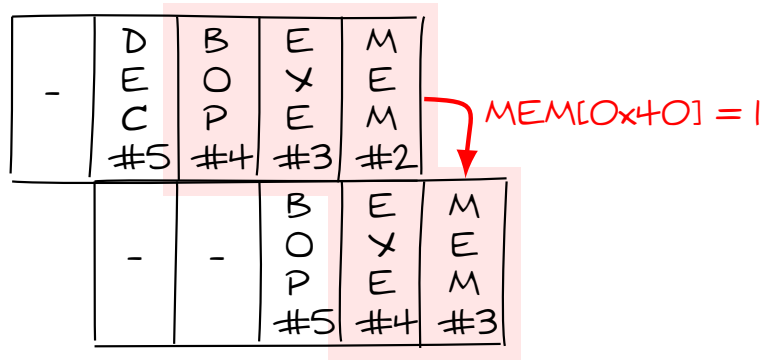


-	DEC #5	BOP #4	EXE #3	MEM #2

Conflito de dados

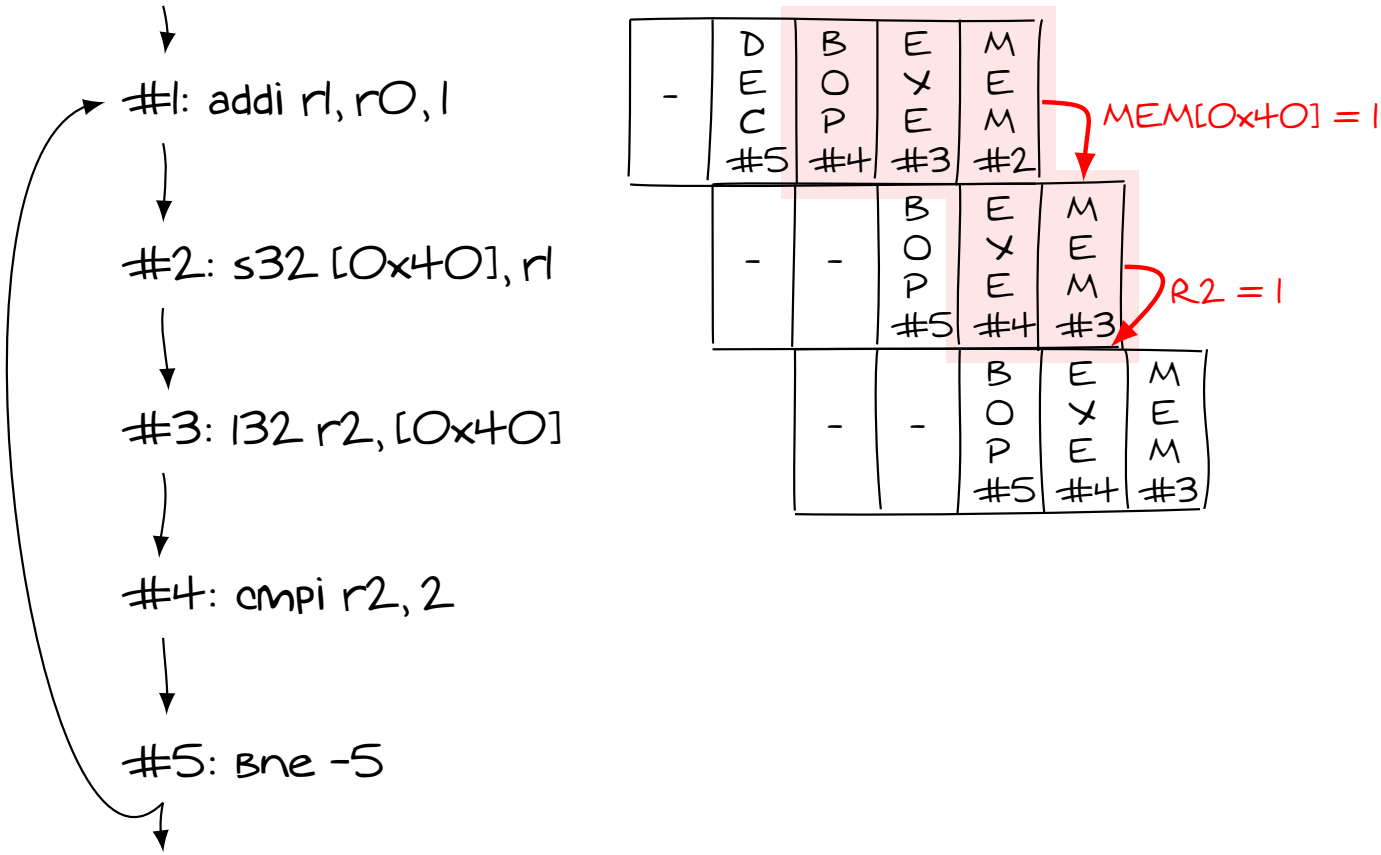
- Adiantamento de dados em memória





Conflito de dados

- Adiantamento de dados em memória



Conflito de dados

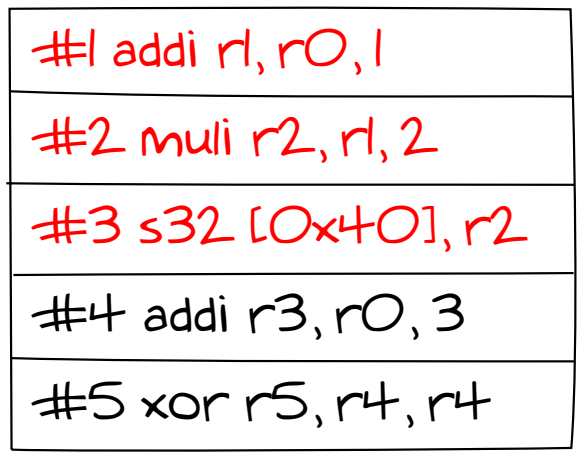
- Impacto do adiantamento de dados
  - ✓ Reduz os atrasos na execução das instruções
  - ✗ Necessita de hardware dedicado nos estágios

Conflito de dados

- Impacto do adiantamento de dados
  - ✓ Reduz os atrasos na execução das instruções
  - ✗ Necessita de hardware dedicado nos estágios
- Arquitetura *load-store*
  - Realiza o uso intensivo dos registradores nas operações, reduzindo os acessos à memória
  - Sem dependência de dados em memória, é evitada a inserção de bolhas entre as instruções no *pipeline*

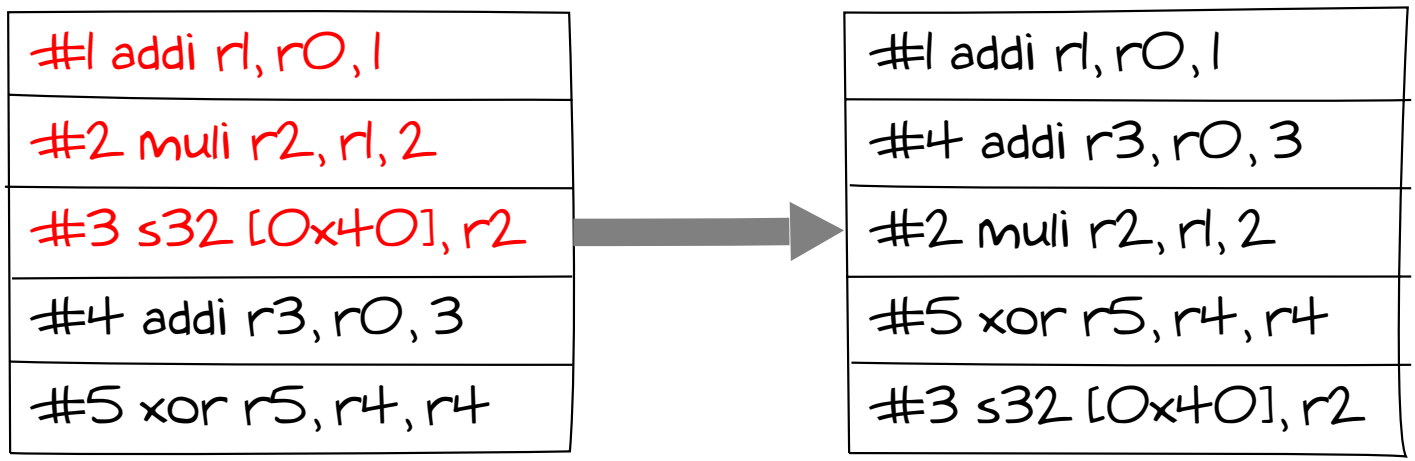
Conflito de dados

- Papel do compilador no conflito de dados
  - Apesar do processador ser capaz de resolver os conflitos no *pipeline*, o código gerado pelo compilador pode eliminar a ocorrência dos conflitos



Conflito de dados

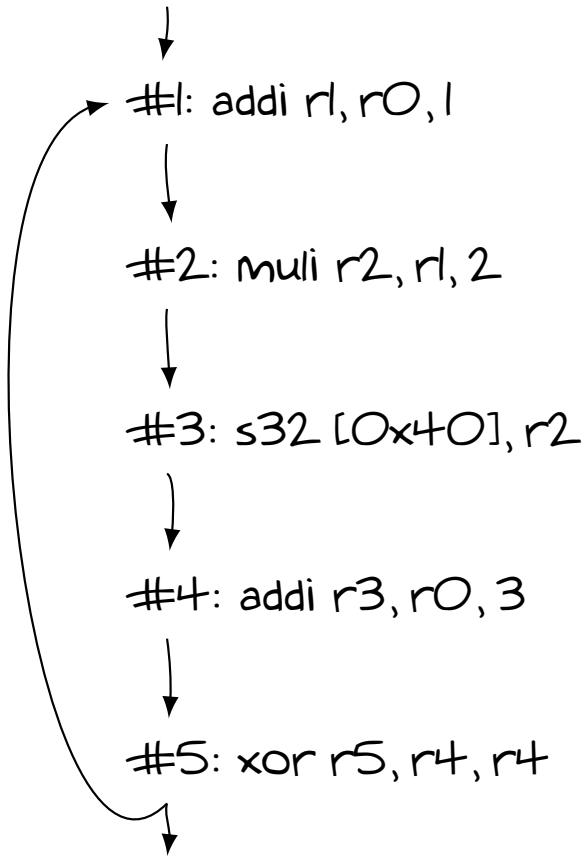
- Papel do compilador no conflito de dados
  - Apesar do processador ser capaz de resolver os conflitos no *pipeline*, o código gerado pelo compilador pode eliminar a ocorrência dos conflitos

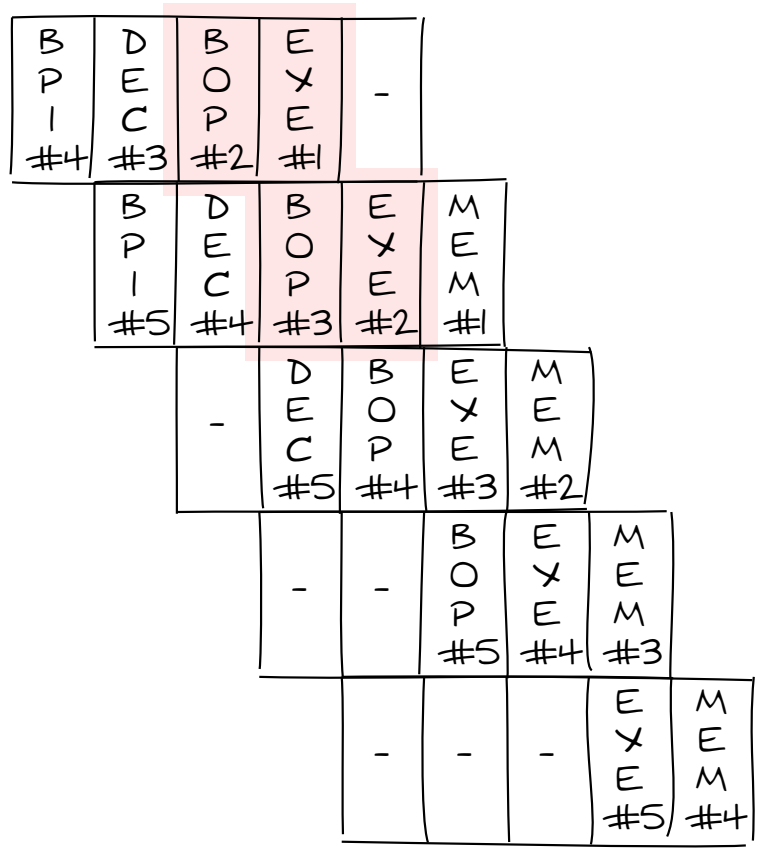


A reorganização das instruções #1, #2 e #3 elimina os conflitos sem impacto no desempenho

Conflito de dados

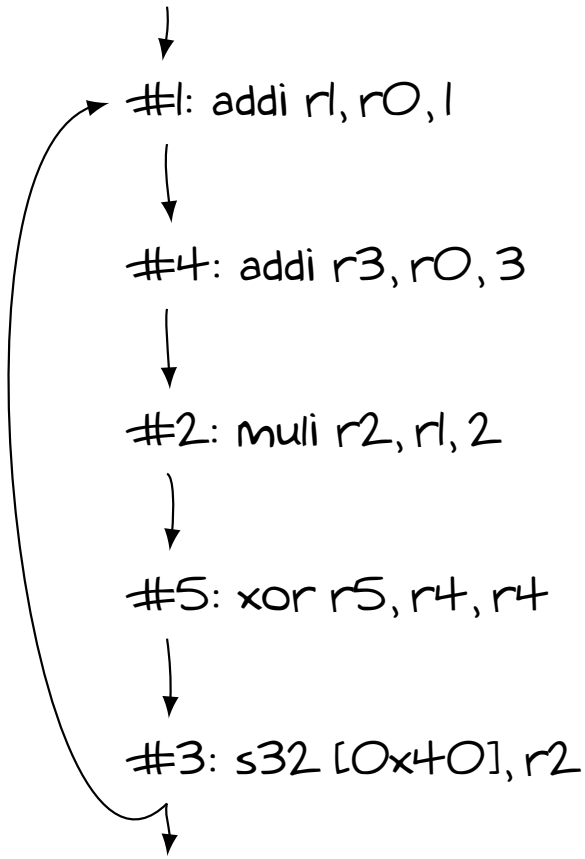
- Papel do compilador no conflito de dados
  - Código sem reorganização

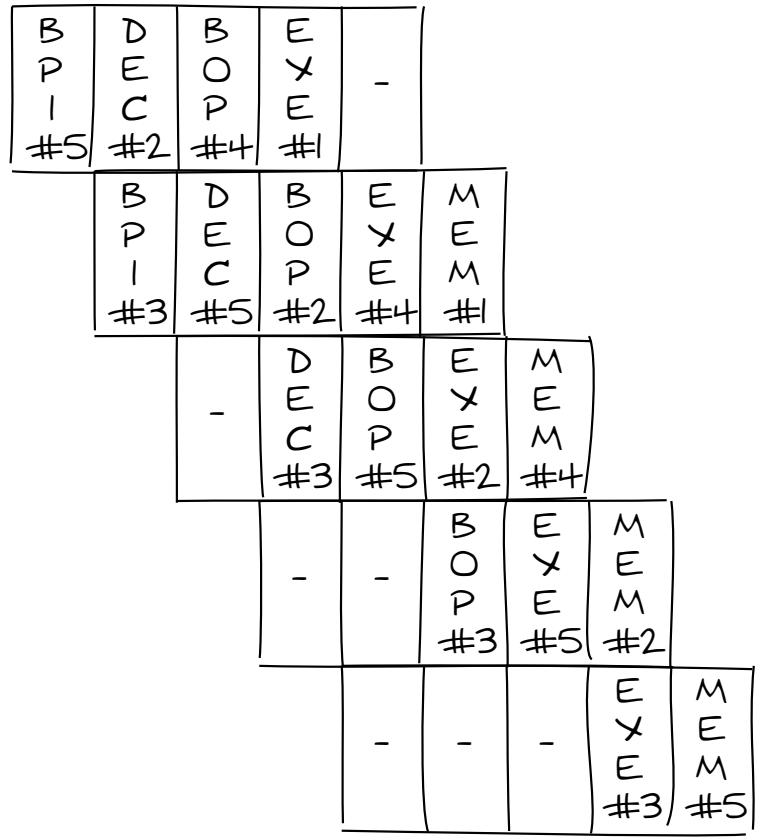




Conflito de dados

- Papel do compilador no conflito de dados
  - Código com reorganização





Conflito de dados

- Quem é responsável por tratar conflitos no *pipeline*?
  - Hardware
    - Durante a popularização do projeto em *pipeline* no final dos anos 70, os compiladores eram muito limitados

Conflito de dados

- Quem é responsável por tratar conflitos no *pipeline*?
  - Hardware
    - Durante a popularização do projeto em *pipeline* no final dos anos 70, os compiladores eram muito limitados
    - O projeto do hardware era mais complexo para detectar e tratar conflitos em tempo de execução

Conflito de dados

- Quem é responsável por tratar conflitos no *pipeline*?
  - Hardware
    - Durante a popularização do projeto em *pipeline* no final dos anos 70, os compiladores eram muito limitados
    - O projeto do hardware era mais complexo para detectar e tratar conflitos em tempo de execução
    - Em última instância, o processador precisa garantir o comportamento correto na execução do software

Conflito de dados

- Quem é responsável por tratar conflitos no *pipeline*?
  - Hardware
    - Durante a popularização do projeto em *pipeline* no final dos anos 70, os compiladores eram muito limitados
    - O projeto do hardware era mais complexo para detectar e tratar conflitos em tempo de execução
    - Em última instância, o processador precisa garantir o comportamento correto na execução do software
  - Software
    - Maior flexibilidade para otimizações e melhorias

Conflito de dados

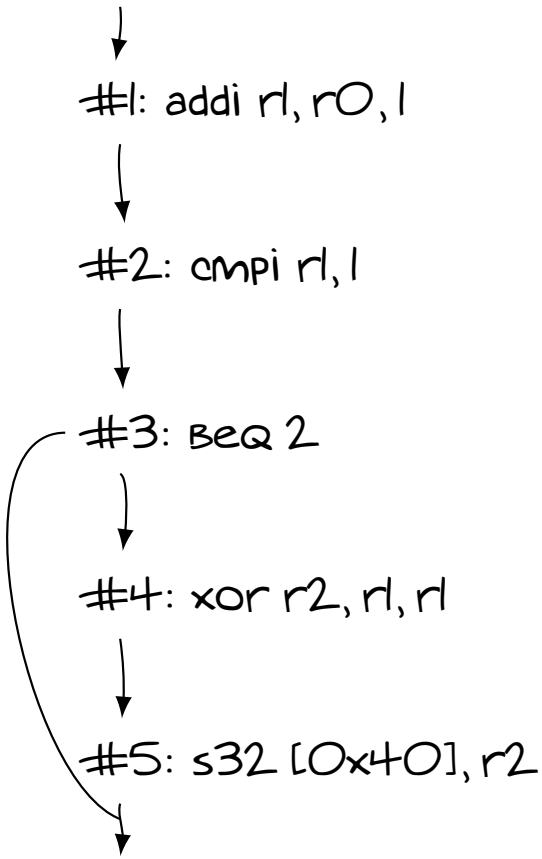
- Quem é responsável por tratar conflitos no *pipeline*?
  - Hardware
    - Durante a popularização do projeto em *pipeline* no final dos anos 70, os compiladores eram muito limitados
    - O projeto do hardware era mais complexo para detectar e tratar conflitos em tempo de execução
    - Em última instância, o processador precisa garantir o comportamento correto na execução do software
  - Software
    - Maior flexibilidade para otimizações e melhorias
    - Ferramentas e técnicas de compilação avançadas

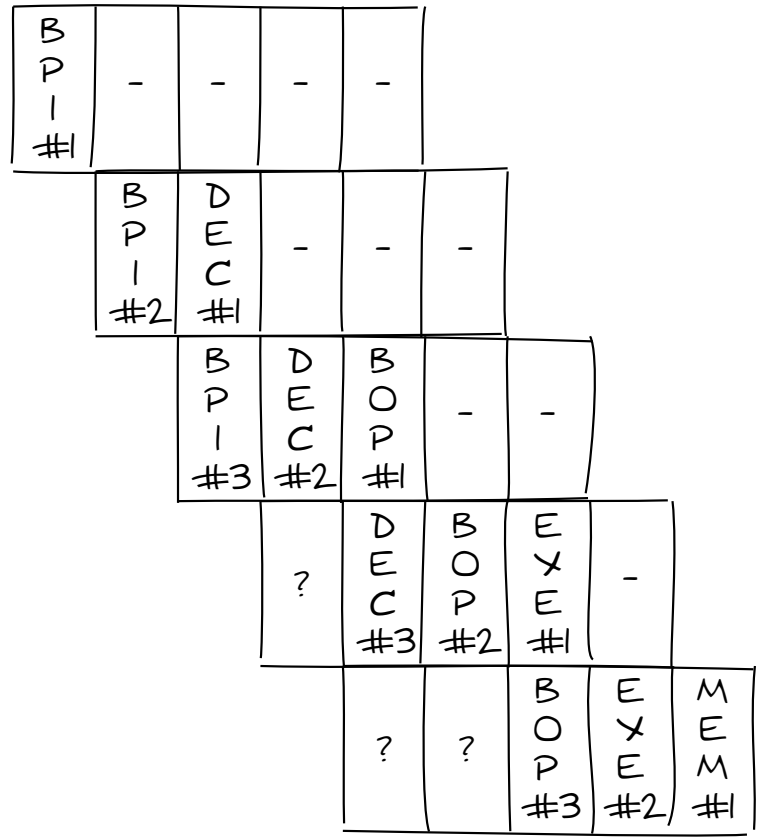
Conflito de dados

- Quem é responsável por tratar conflitos no *pipeline*?
  - Hardware
    - Durante a popularização do projeto em *pipeline* no final dos anos 70, os compiladores eram muito limitados
    - O projeto do hardware era mais complexo para detectar e tratar conflitos em tempo de execução
    - Em última instância, o processador precisa garantir o comportamento correto na execução do software
  - Software
    - Maior flexibilidade para otimizações e melhorias
    - Ferramentas e técnicas de compilação avançadas
    - Simplificação do projeto de processador, delegando para o compilador tarefas de tratamento de conflito

Conflito de controle

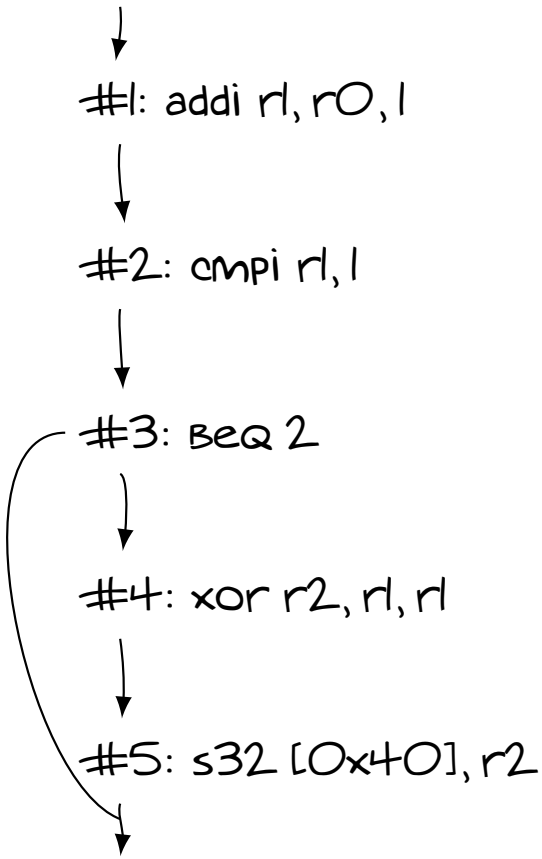
- É decorrente da execução de instruções de desvio de fluxo que são dependentes de condições que ainda serão calculadas ou modificadas no *pipeline*

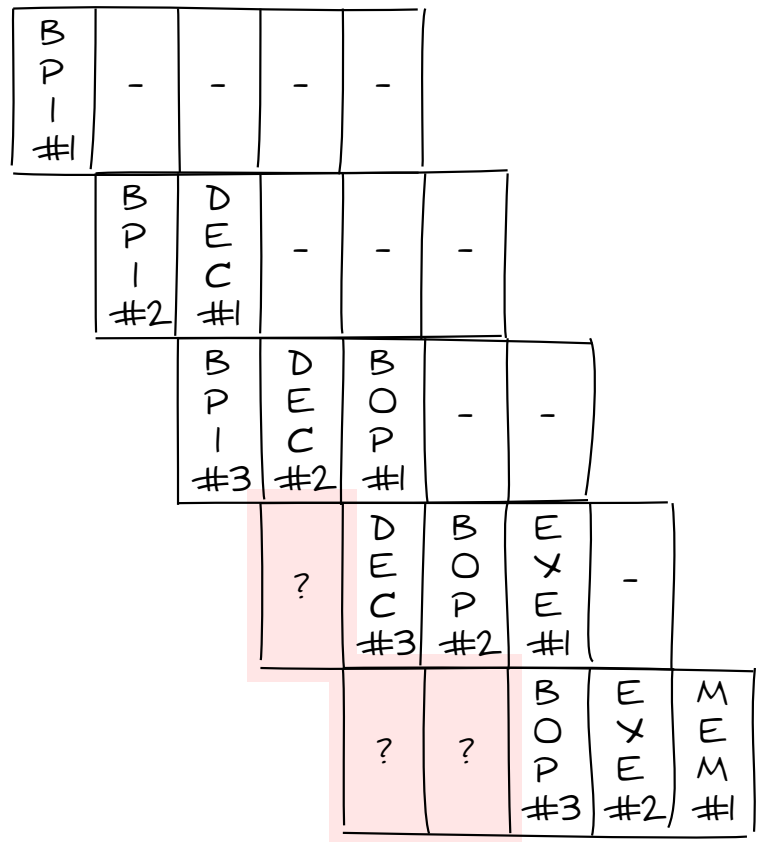




Conflito de controle

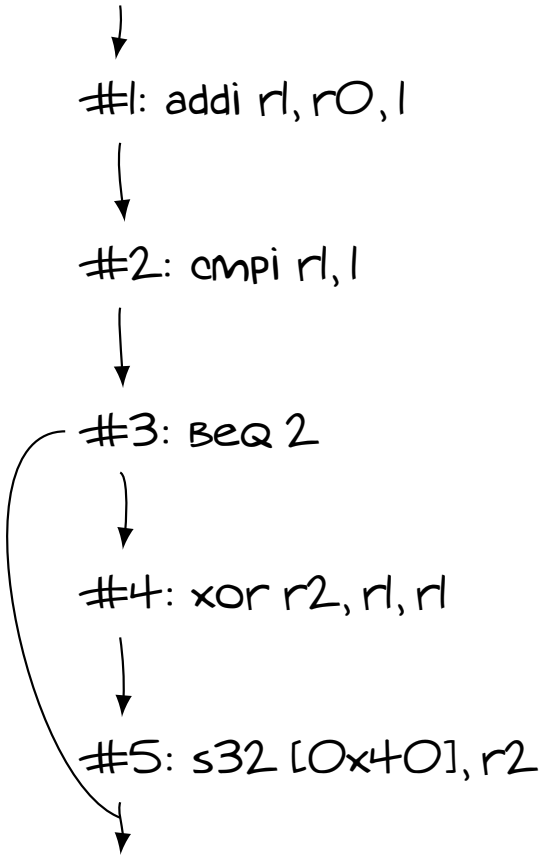
- É decorrente da execução de instruções de desvio de fluxo que são dependentes de condições que ainda serão calculadas ou modificadas no *pipeline*

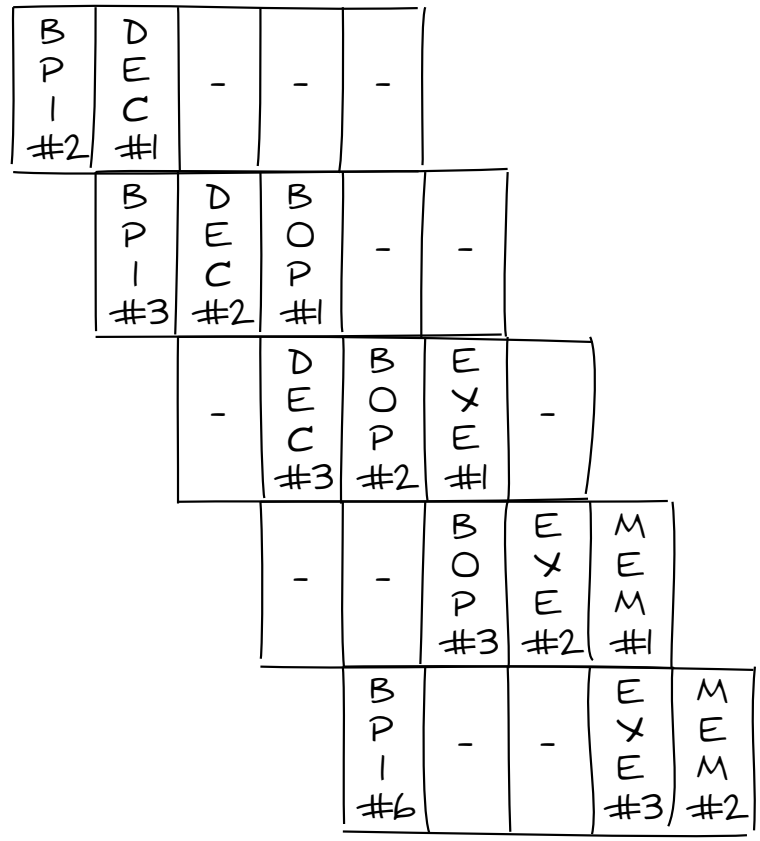




Conflito de controle

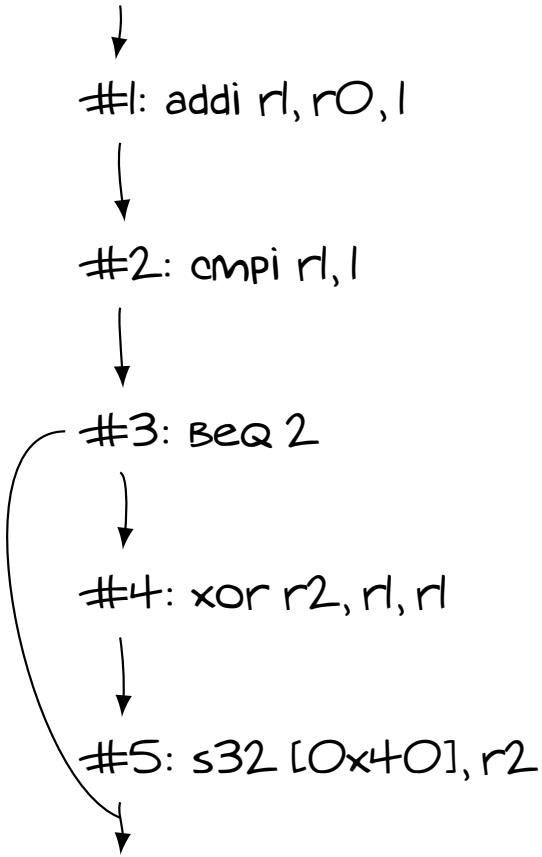
- Inserção de atrasos ou bolhas no *pipeline*

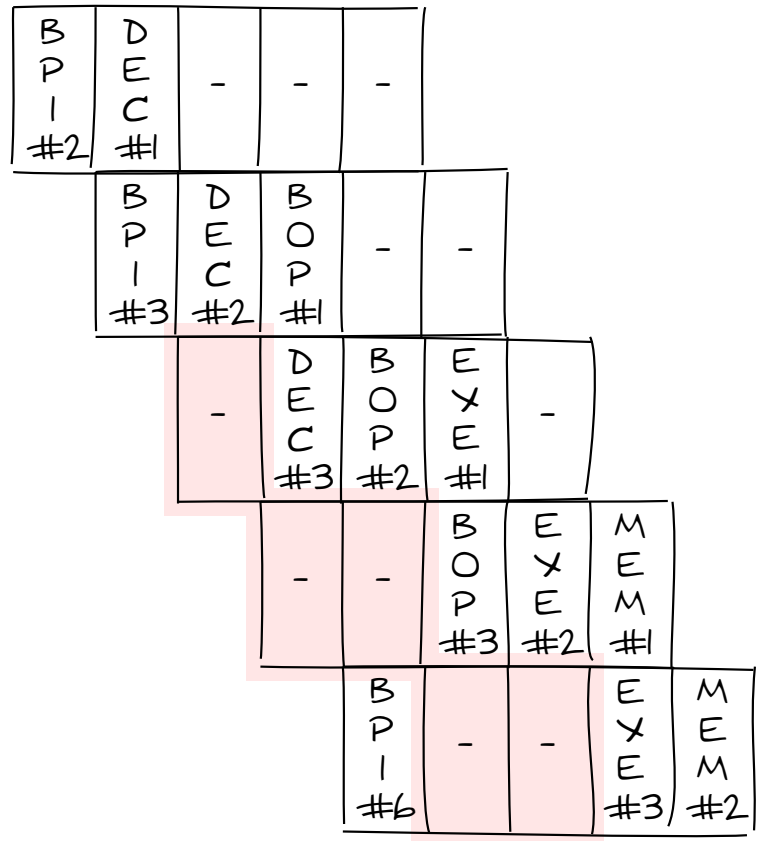




Conflito de controle

- Inserção de atrasos ou bolhas no *pipeline*





Conflito de controle

- Impacto da inserção de atrasos ou bolhas
  - ✓ Resolve o conflito de forma simples
  - ✗ Ineficiente e reduz muito o desempenho
  - ✗ Necessita de hardware dedicado nos estágios

Conflito de controle

- Como prever de forma eficiente quais instruções serão executadas após o desvio no *pipeline*?

Conflito de controle

- Como prever de forma eficiente quais instruções serão executadas após o desvio no *pipeline*?
  - **Estaticamente**: instruções de atraso (*delay slot*) são executadas até resolver as condições do desvio

Conflito de controle

- Como prever de forma eficiente quais instruções serão executadas após o desvio no *pipeline*?
  - **Estaticamente**: instruções de atraso (*delay slot*) são executadas até resolver as condições do desvio
  - **Dinamicamente**: executando os desvios de forma especulativa, baseando-se no histórico de desvios

Conflito de controle

- Inserção de instruções de atraso (*delay slot*)
  - As operações que independem do controle de fluxo condicional ou iterativo são sempre executadas

```
1  // Biblioteca padrão
2  #include <stdlib.h>
3  // Função principal
4  int main() {
5      // Declaração de variáveis
6      int a = rand(), b, c, d = 10;
7      // Operação and
8      b = d & d;
9      // Controle condicional
10     if(a == 0) a = 1;
11     else a = 0;
12     // Operação or
13     c = d | d;
14     // Retorno sem erros
15     return 0;
16 }
```

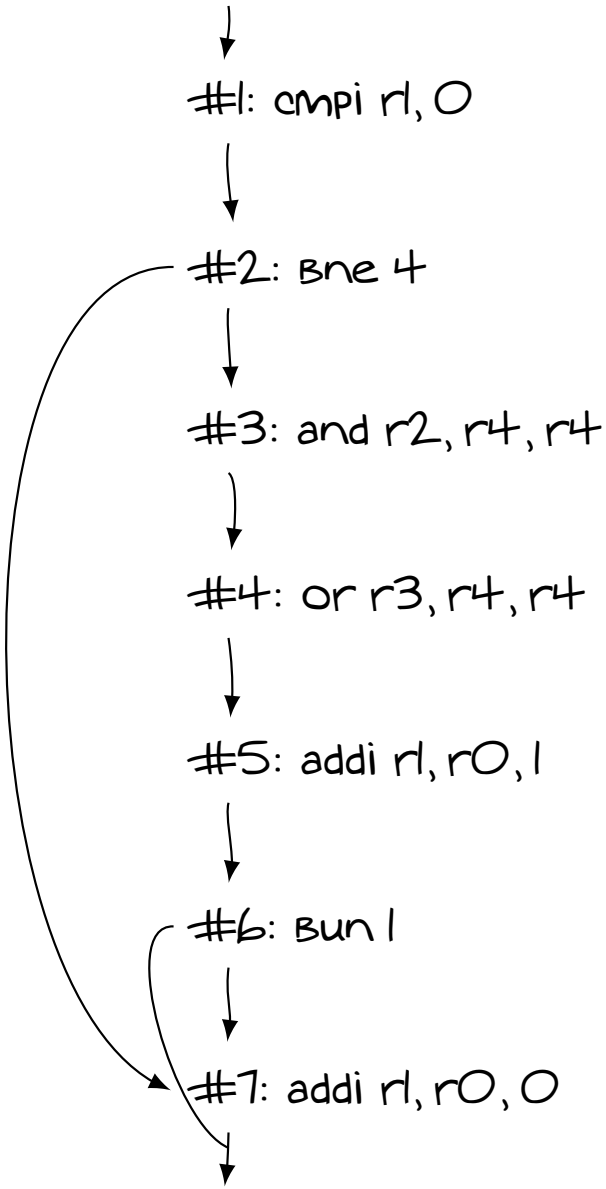
Conflito de controle

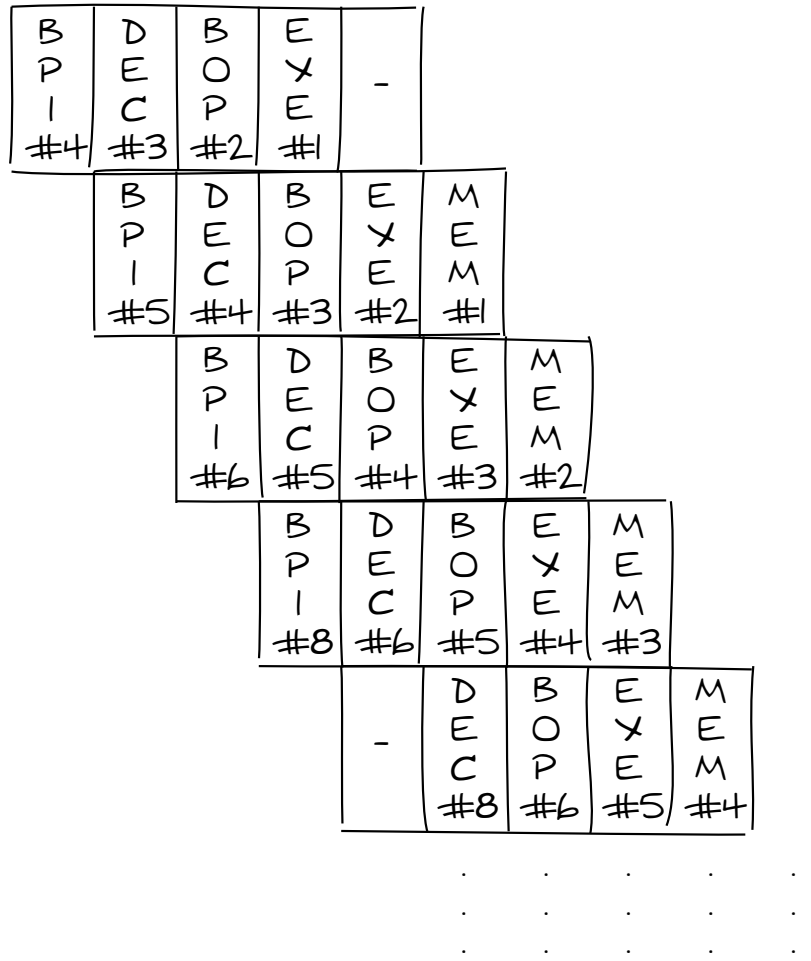
- Inserção de instruções de atraso (*delay slot*)
  - As operações que independem do controle de fluxo condicional ou iterativo são sempre executadas

```
1  // Biblioteca padrão
2  #include <stdlib.h>
3  // Função principal
4  int main() {
5      // Declaração de variáveis
6      int a = rand(), b, c, d = 10;
7      // Operação and
8      b = d & d;
9      // Controle condicional
10     if(a == 0) a = 1;
11     else a = 0;
12     // Operação or
13     c = d | d;
14     // Retorno sem erros
15     return 0;
16 }
```

Conflito de controle

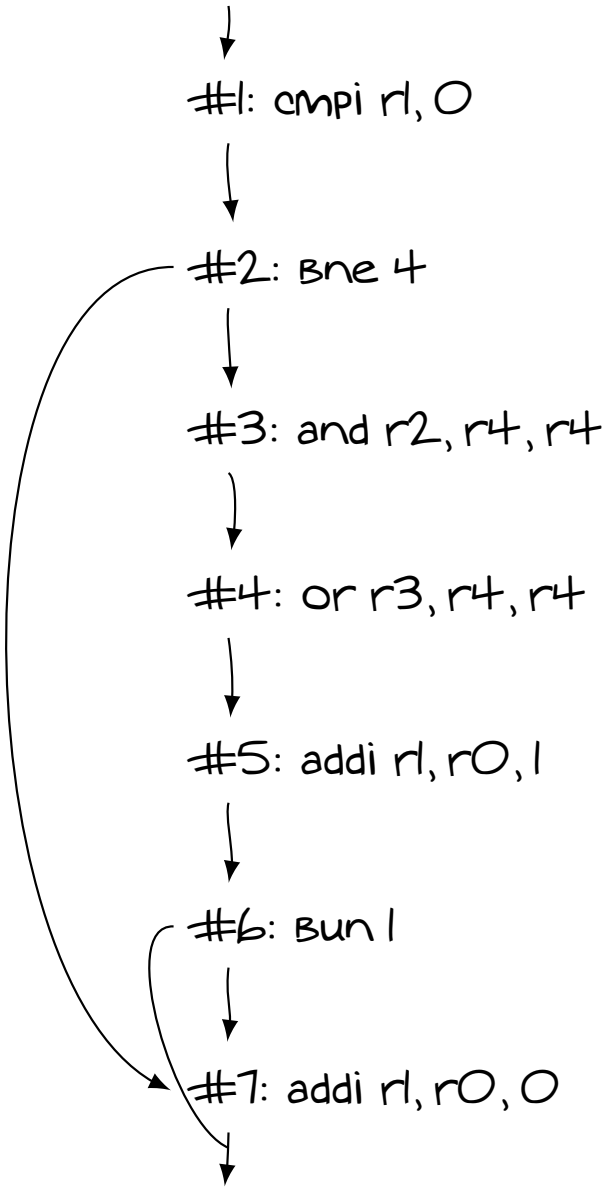
- Inserção de instruções de atraso (*delay slot*)

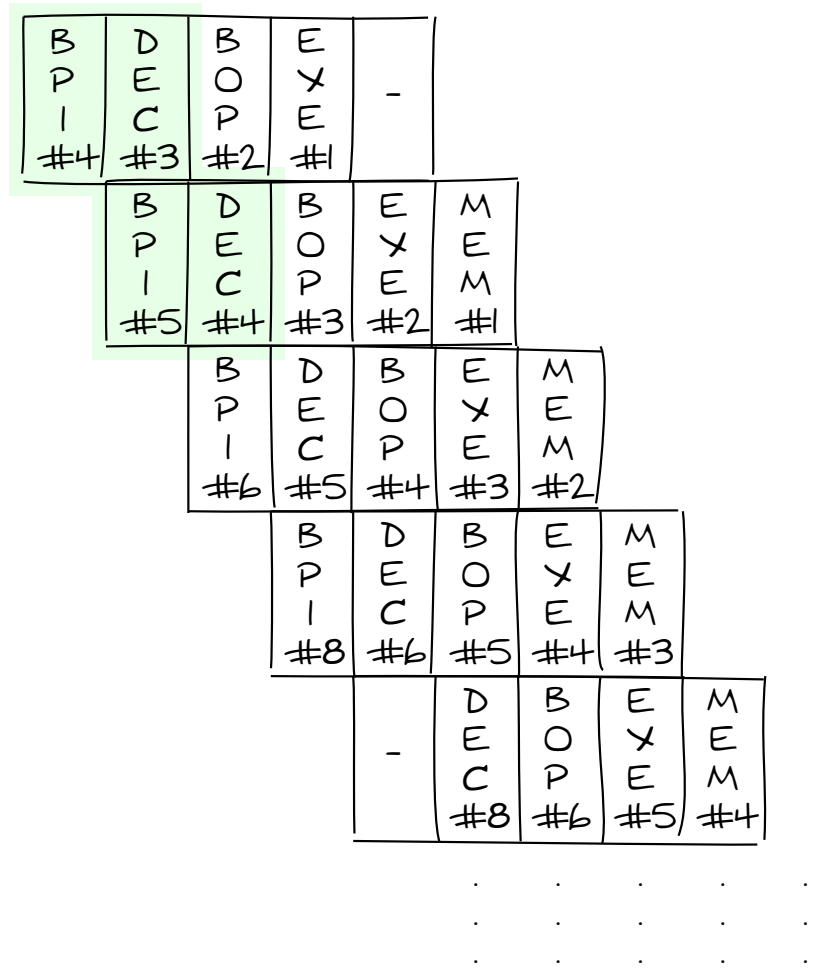




Conflito de controle

- Inserção de instruções de atraso (*delay slot*)





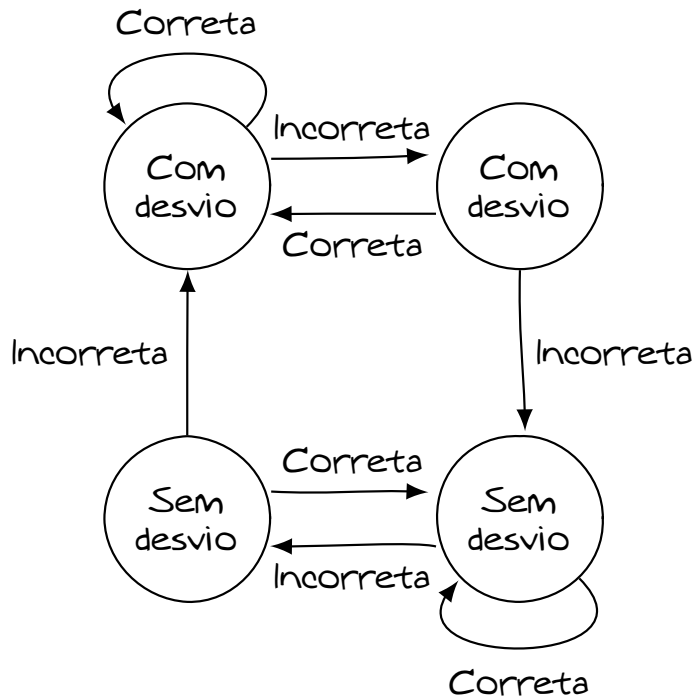
Conflito de controle

- Inserção de instruções de atraso (*delay slot*)
  - Esta técnica precisa ser suportada pela arquitetura e as instruções escalonadas pelo compilador
  - Quando o compilador não consegue alocar operações que sejam independentes do controle de fluxo, são inseridas instruções **nop** que são equivalentes as bolhas geradas pelo *pipeline*

Conflito de controle

- Previsão dinâmica de desvio
  - Tabela de história de desvios de 2 bits

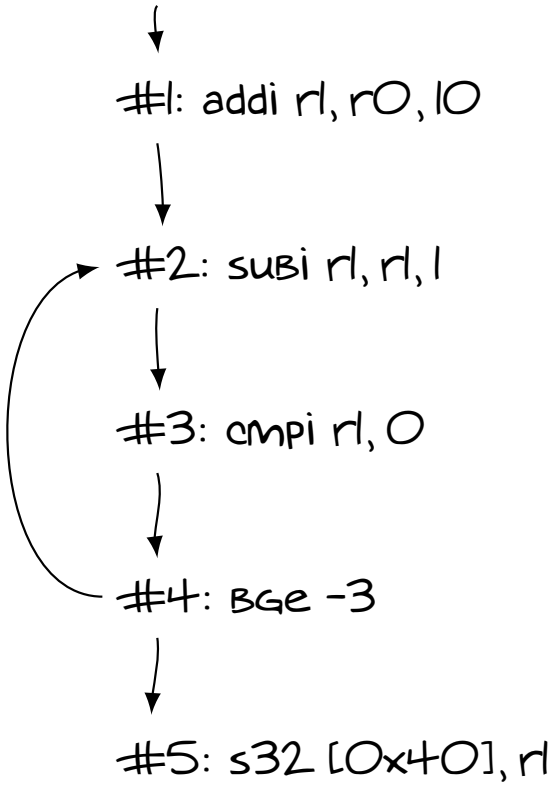
Código	Estimativa	Correta	Incorreta
00	Sem desvio	00	01
01	Sem desvio	00	10
10	Com desvio	10	11
11	Com desvio	10	00

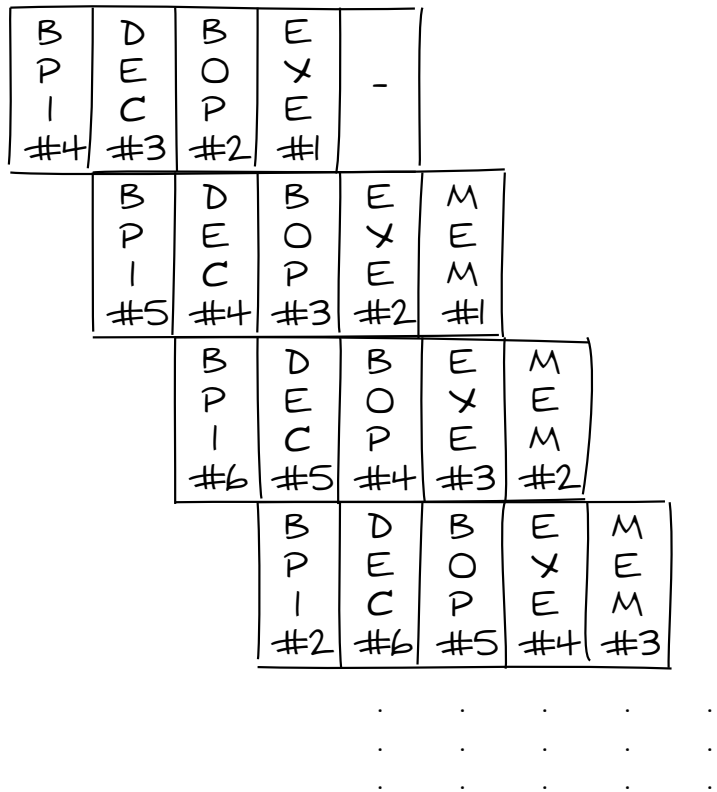


Os preditores podem ser locais,
globais ou a combinação de ambos

Conflito de controle

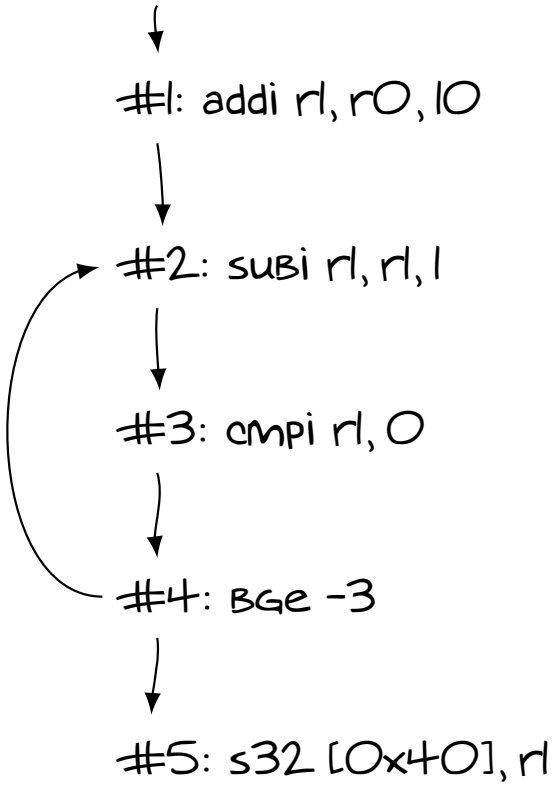
- Predição dinâmica de desvio

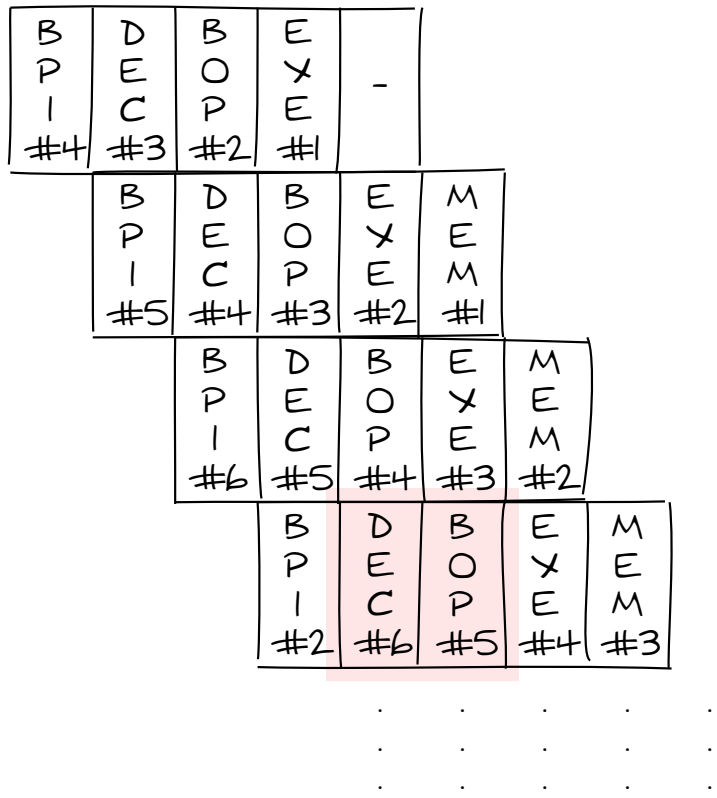




Conflito de controle

- Predição dinâmica de desvio

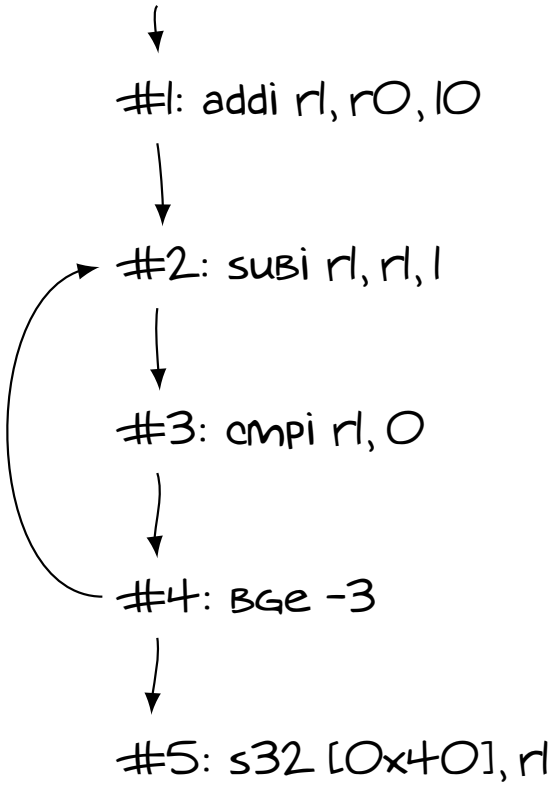


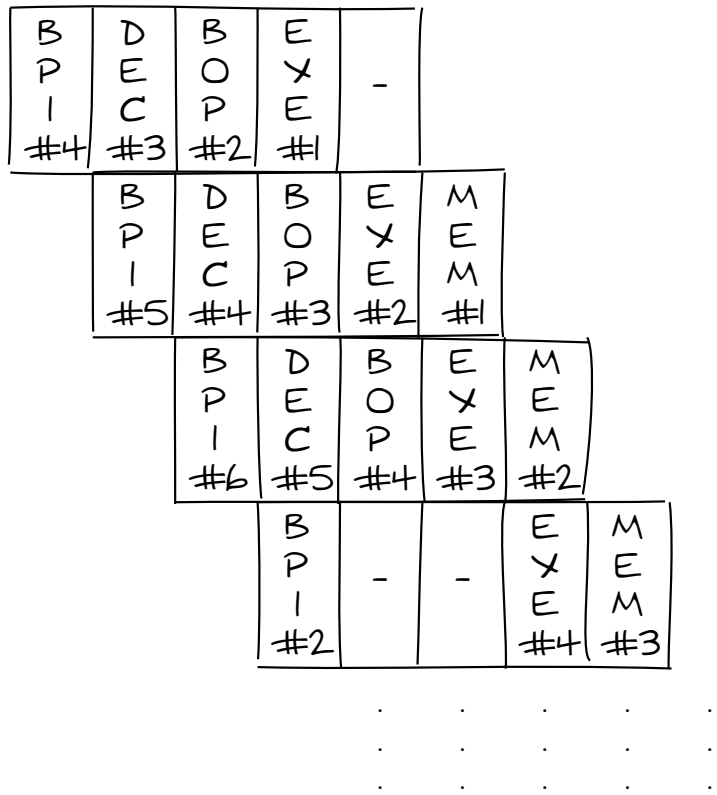


Iteração 1 (predição incorreta)

Conflito de controle

- Predição dinâmica de desvio

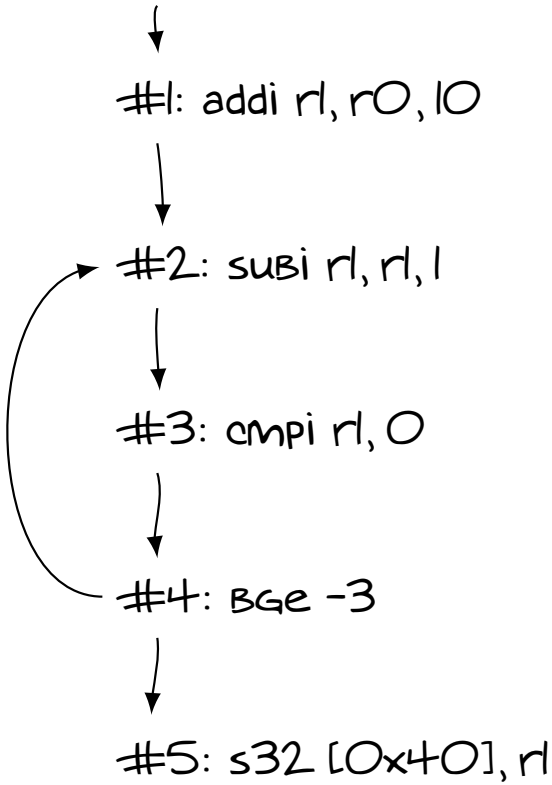


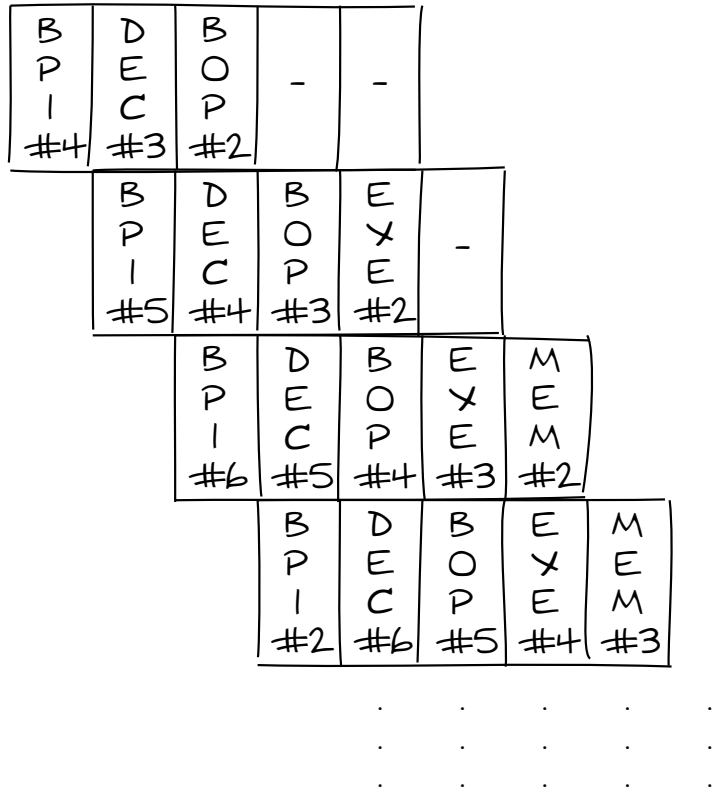


Iteração 1 (predição incorreta)

Conflito de controle

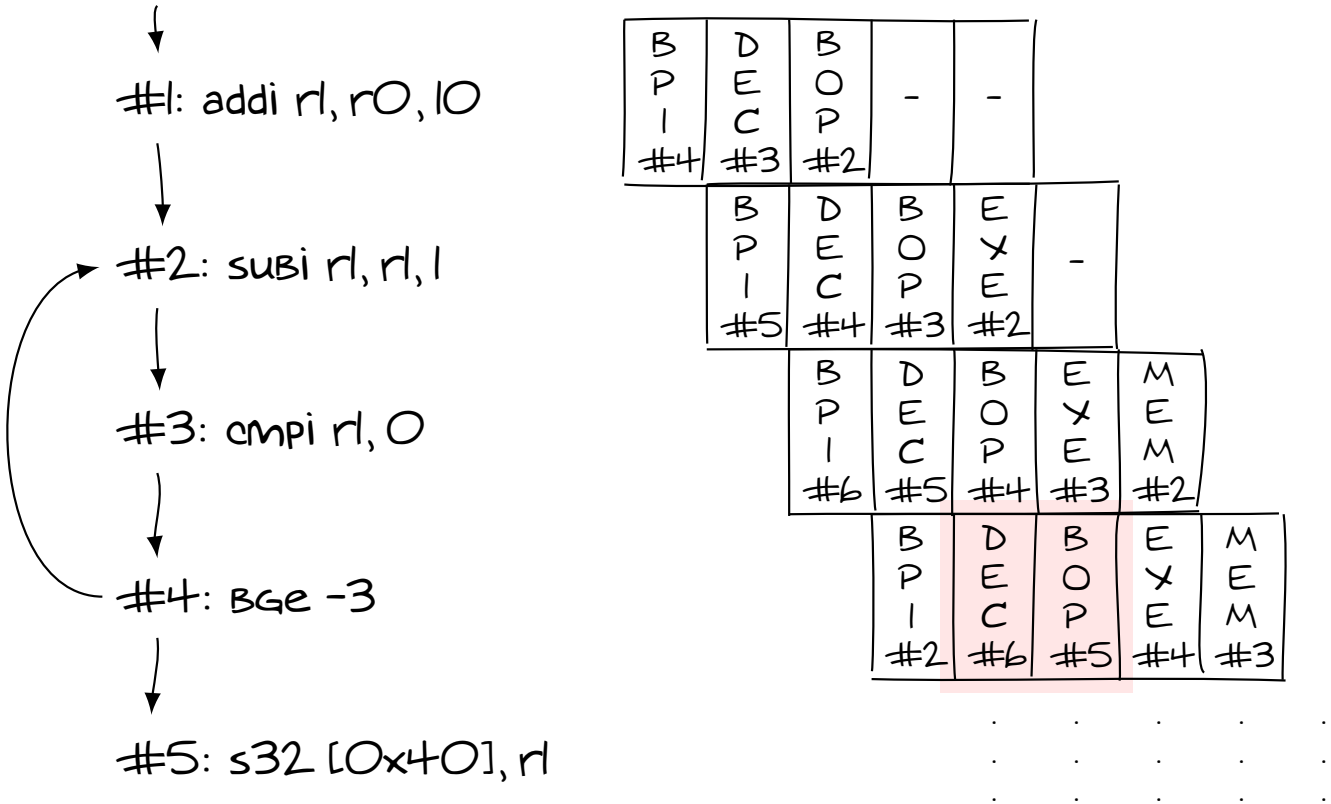
- Predição dinâmica de desvio





Conflito de controle

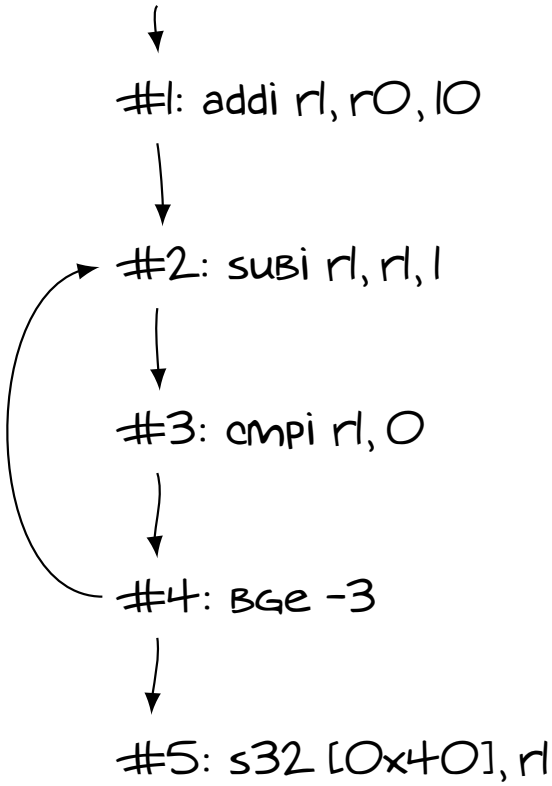
- Predição dinâmica de desvio

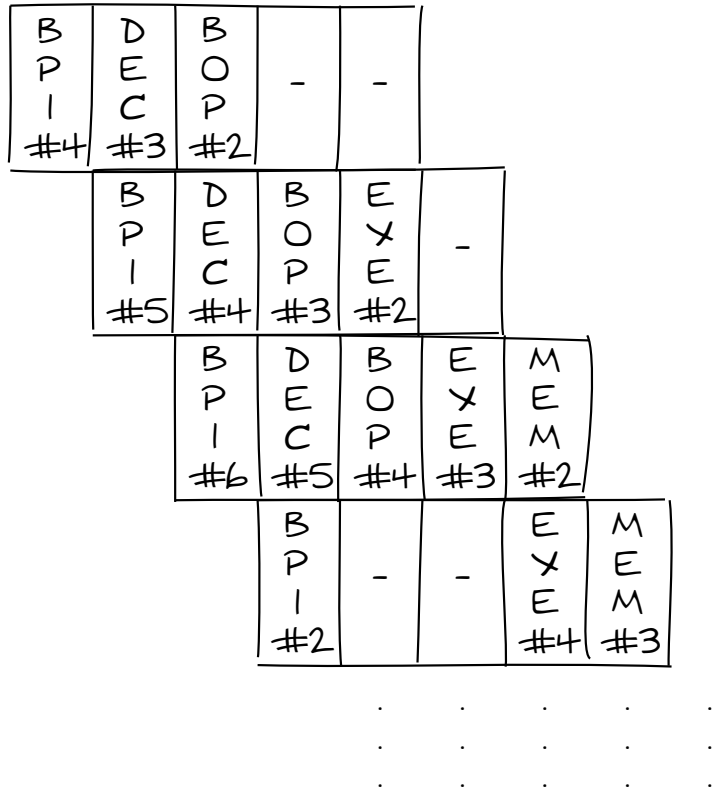


Iteração 2 (predição incorreta)

Conflito de controle

- Predição dinâmica de desvio

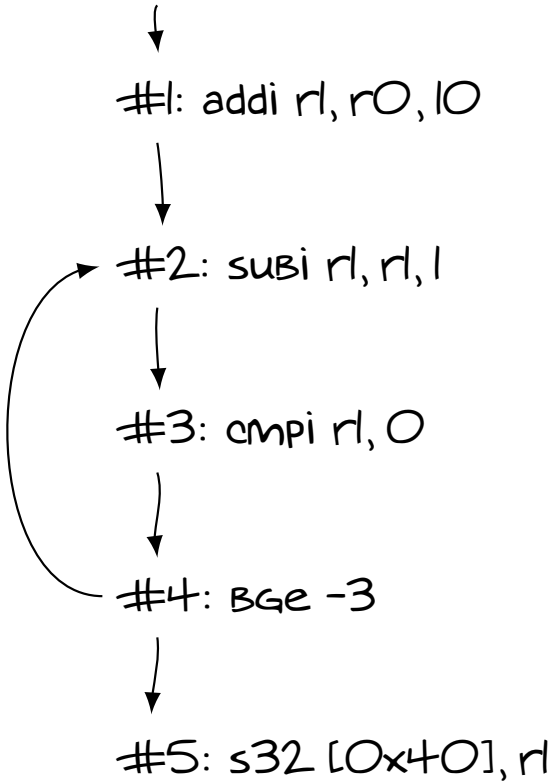


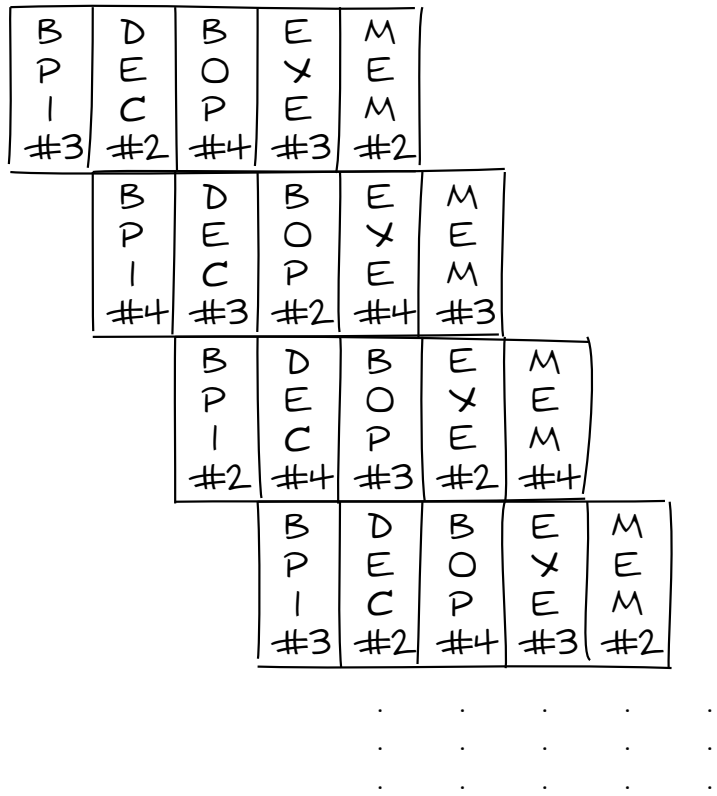


Iteração 2 (predição incorreta)

Conflito de controle

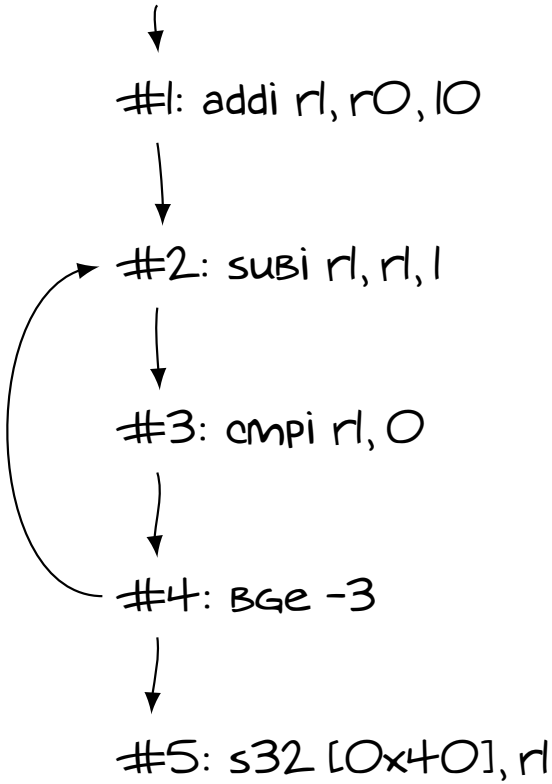
- Predição dinâmica de desvio

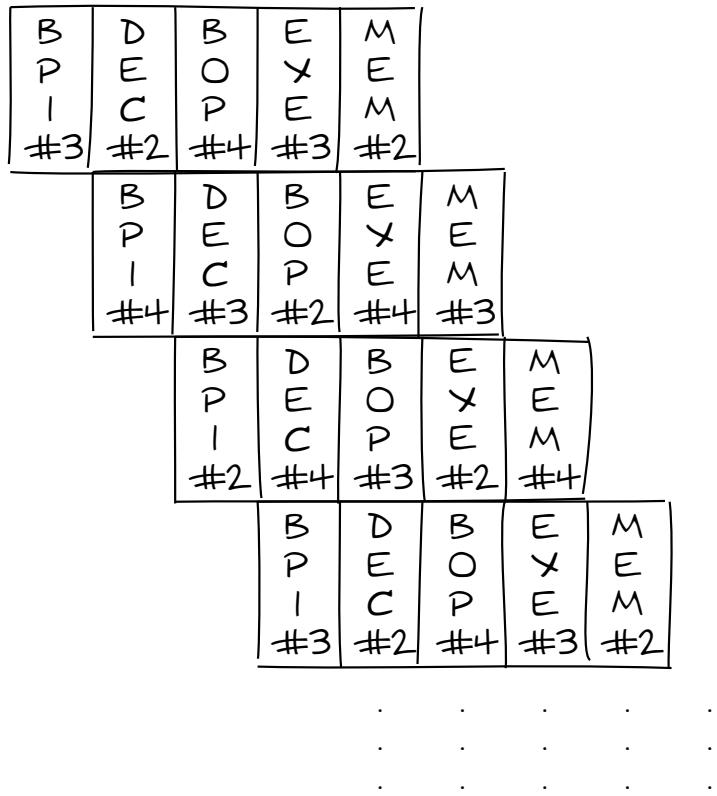




Conflito de controle

- Predição dinâmica de desvio

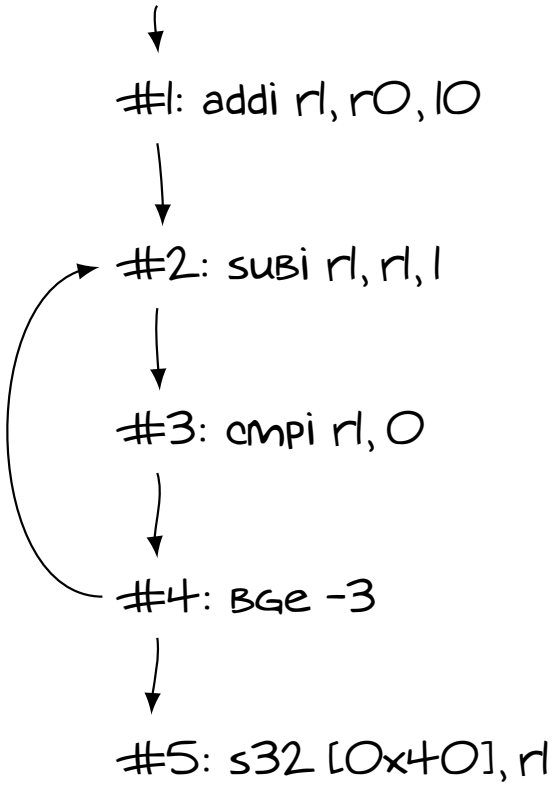


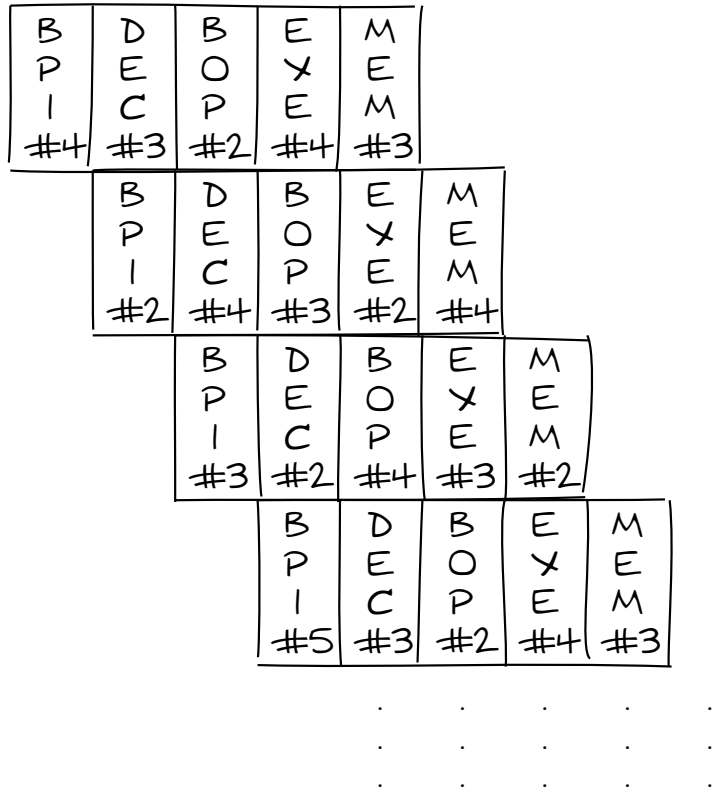


Iterações 3 até 9 (predição correta)

Conflito de controle

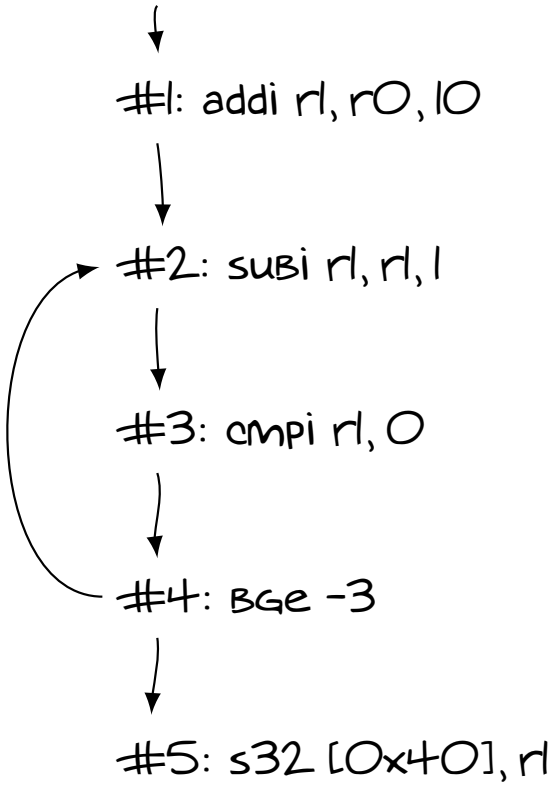
- Predição dinâmica de desvio

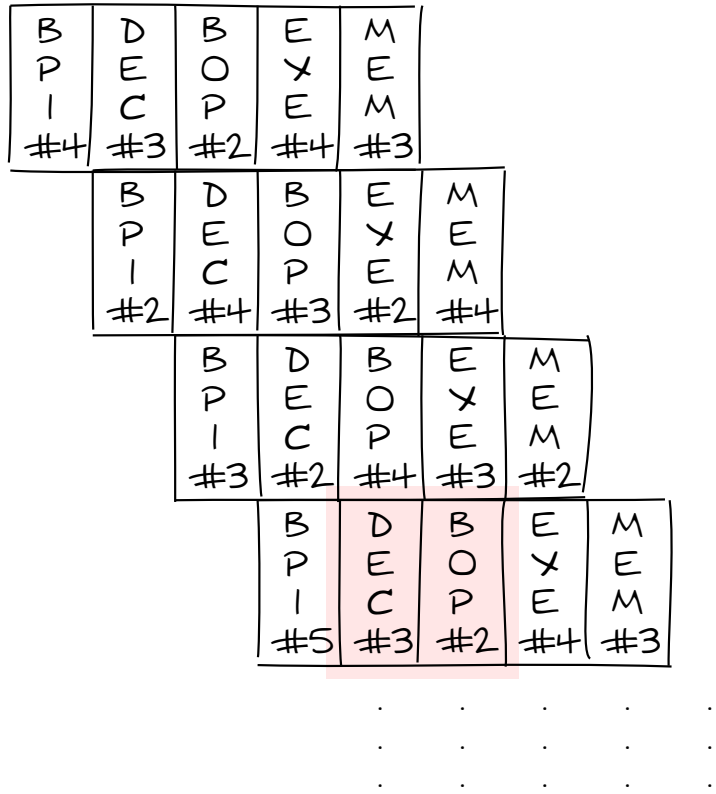




Conflito de controle

- Predição dinâmica de desvio

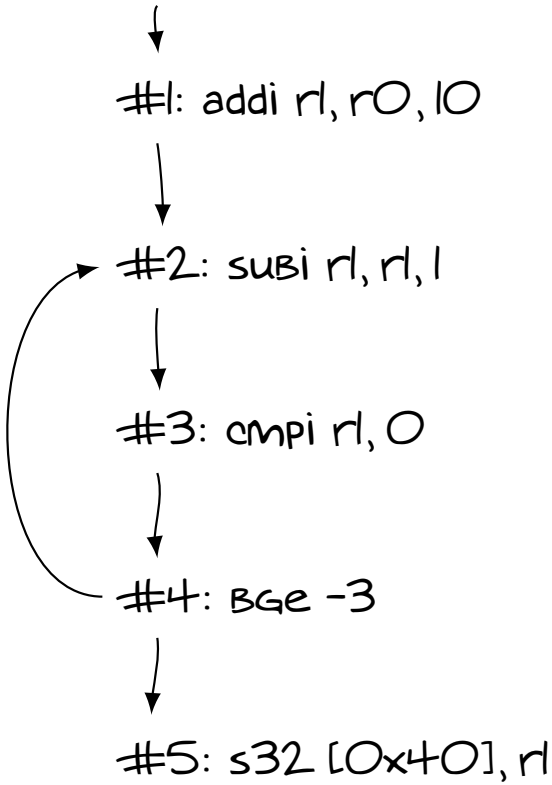


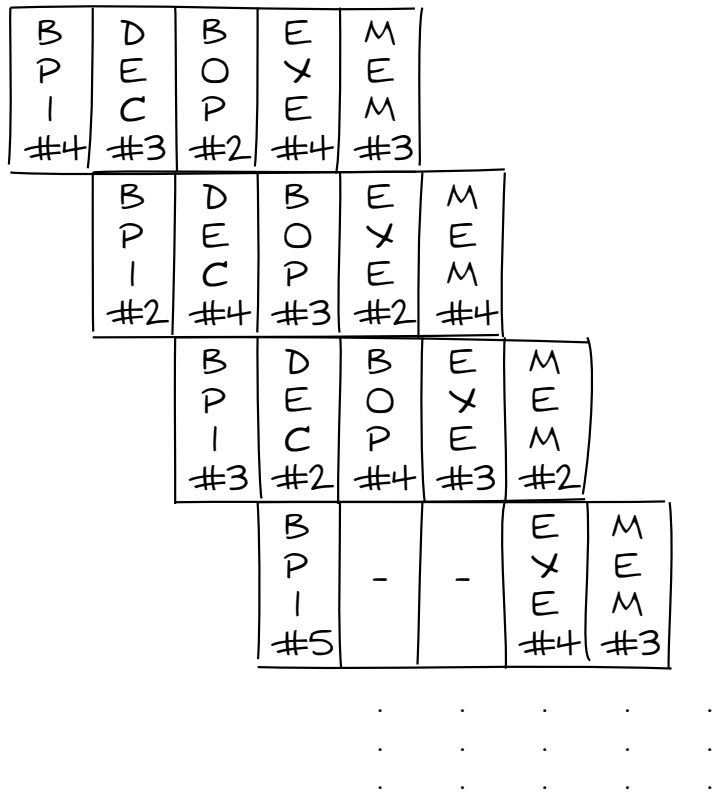


Iteração 10 (predição incorreta)

Conflito de controle

- Predição dinâmica de desvio





Iteração 10 (predição incorreta)

Conflito de controle

- Predição dinâmica de desvio
  - A máquina de estados de predição muda sua previsão após dois resultados sequenciais incorretos
  - Técnicas mais complexas de predição acertam com mais de 90% de precisão a ocorrência dos desvios

Exercício

- Execute o programa abaixo no *pipeline* de 5 estágios, considerando que o processador utiliza uma organização de memória Von Neumann
  - O comportamento deve ser igual ao multiciclo, mostrando o tratamento dos diferentes conflitos
  - Represente graficamente os estágios do *pipeline*

```
1  // R1 = 0
2  mov r1, 0
3  // MEM[0x100] = R1
4  s32 [0x40], r1
5  // R1++
6  addi r1, r1, 1
7  // R1 ? 10
8  cmpi r1, 10
9  // Desvio condicional
10 blt -4
11 // Fim
12 int 0
```