DESDOBRAMENTO E ESCALONAMENTO DE LOOP – PIPELINE MIPS 32 BITS

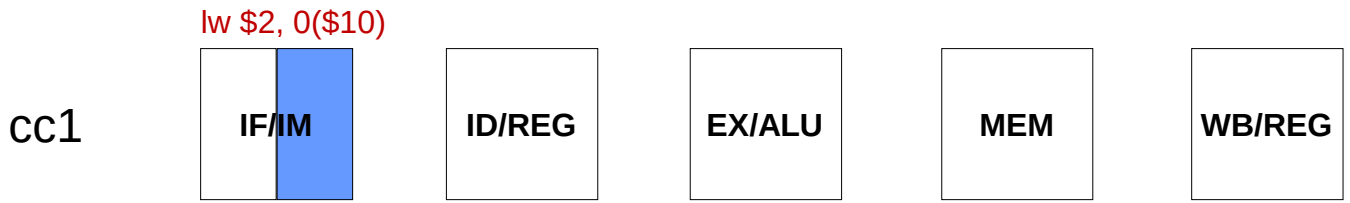
ELAINE CECÍLIA GATTO

Usando o código a seguir:

- Desdobre o código quatro vezes e escalone-o para execução rápida no pipeline padrão do MIPS sem mecanismo de forwarding.
- Considere que ele admite a instrução ADDI. Você pode considerar que o LOOP é executado para um múltiplo de quatro vezes. Suponha inicialmente que$10 = 0 e $30 = 400. Suponha também que os desvios são resolvidos no estágio MEM.
- Como o código escalonado se compara com o código não escalonado original em termos de execução—isto é, em termos de speed up? Mostre o código desdobrado e depois escalonado.

```
LOOP: Lw $2, 0($10)
      Sub $4, $2, $3
      Sw $4, 0($10)
      Addi $10, $10, 4
      Bne $10, $30, LOOP
```

ANALISANDO O CÓDIGO MIPS PARA IDENTIFICAR OS CONFLITOS/HAZARDS/DEPENDENCIAS

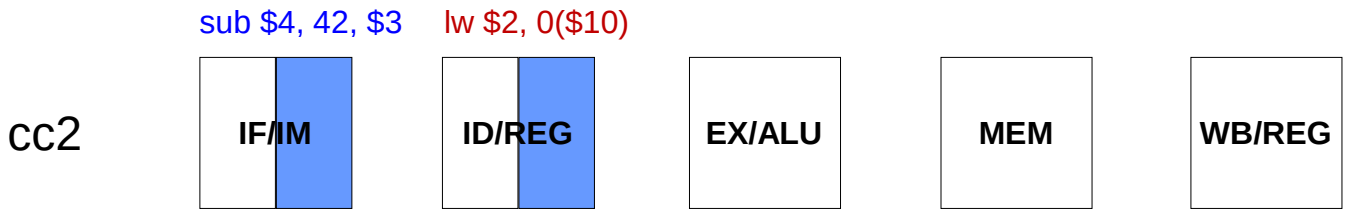


CICLO 1:

- A instrução LW entra no pipeline

O conteúdo deste material está no livro ARQUITETURA DE COMPUTADORES UMA ABORDAGEM QUANTITATIVA. Vai da página 135 a 140.

ANALISANDO O CÓDIGO MIPS PARA IDENTIFICAR OS CONFLITOS/HAZARDS/DEPENDENCIAS



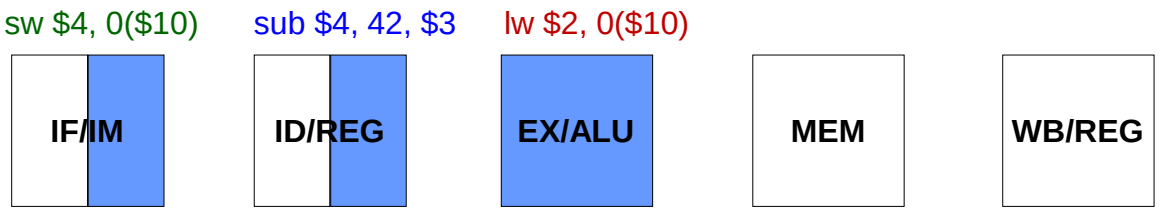
CICLO 2:

- A instrução LW está sendo codificada

- A instrução sub entra no pipeline

ANALISANDO O CÓDIGO MIPS PARA IDENTIFICAR OS CONFLITOS/HAZARDS/DEPENDENCIAS

cc3



CICLO 3:

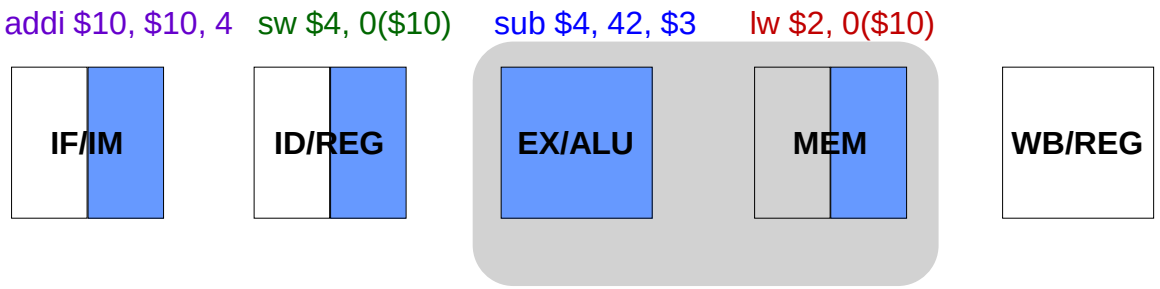
- A instrução LW está calculando o endereço de memória de onde deve pegar o operando

- A instrução SUB está sendo codificada

- A instrução SW entra no pipeline

ANALISANDO O CÓDIGO MIPS PARA IDENTIFICAR OS CONFLITOS/HAZARDS/DEPENDENCIAS

cc4



CICLO 4:

- A instrução LW está buscando o valor do endereço de memória calculado

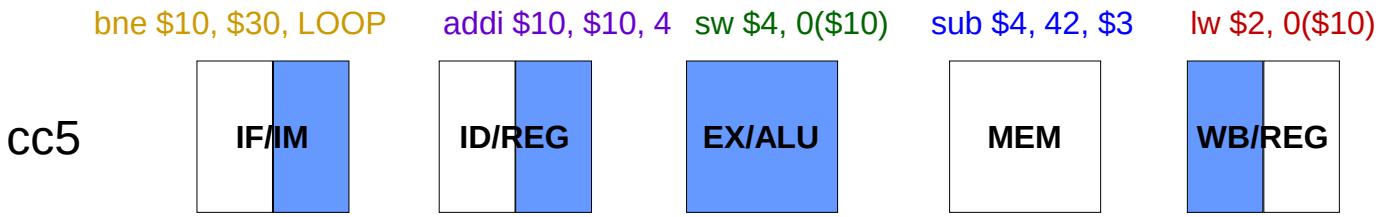
- A instrução SUB está sendo calculada: $4 = $2 - $3

- A instrução SW está sendo codificada

- A instrução ADDI entra no pipeline.

Neste ciclo temos um conflito entre as instruções LW e SUB. O valor do operando $2 que SUB precisa ainda não está disponível!

ANALISANDO O CÓDIGO MIPS PARA IDENTIFICAR OS CONFLITOS/HAZARDS/DEPENDENCIAS



CICLO 5:

- A instrução LW escreve no registrador $2 e sai do pipeline

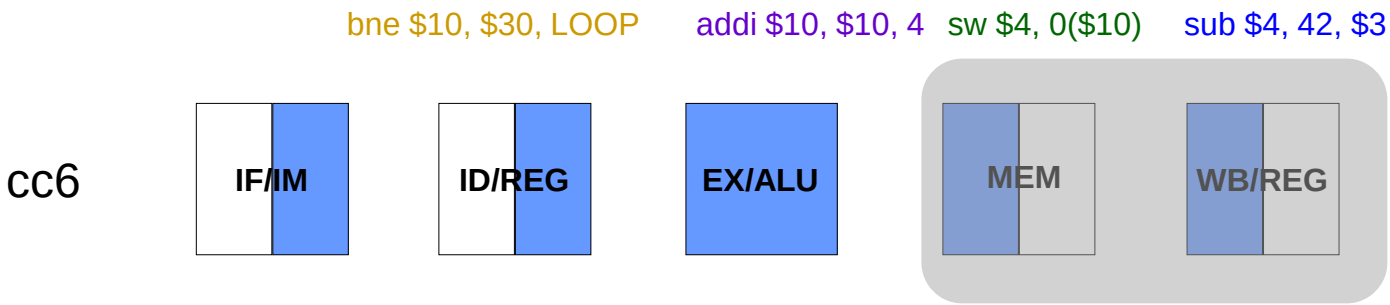
- A instrução SUB não faz nada no estágio MEM neste ciclo

- A instrução SW está calculando o endereço de memória

- A instrução ADDI está sendo codificada

- A instrução BNE entra no pipeline

ANALISANDO O CÓDIGO MIPS PARA IDENTIFICAR OS CONFLITOS/HAZARDS/DEPENDENCIAS



CICLO 6:

- A instrução SUB escreve no registrador $4 e sai do pipeline

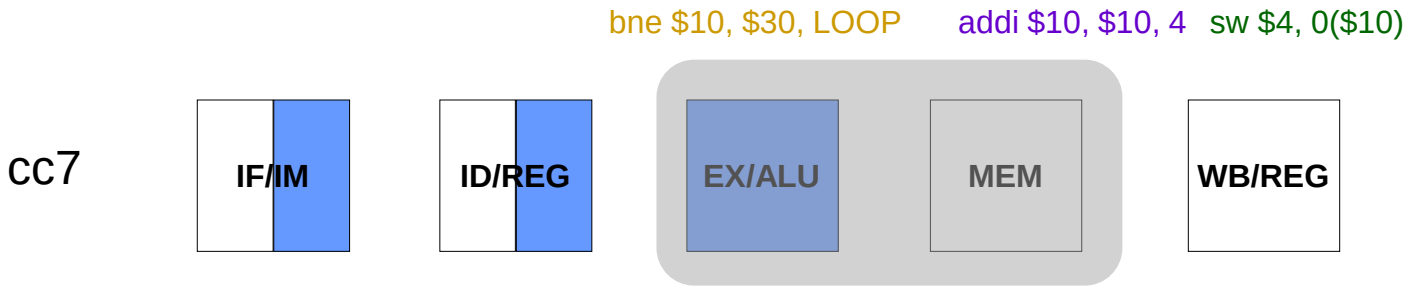
- A instrução SW escreve o valor que está no registrador $4 no endereço de memória calculado

- A instrução ADDI calcula $10 = $10 + 4

- A instrução BNE está sendo codificada

Aqui temos um problema com as instruções SUB e SW. Qual valor correto que será escrito por SW sendo que SUB ainda não terminou a escrita?

ANALISANDO O CÓDIGO MIPS PARA IDENTIFICAR OS CONFLITOS/HAZARDS/DEPENDENCIAS



CICLO 7:

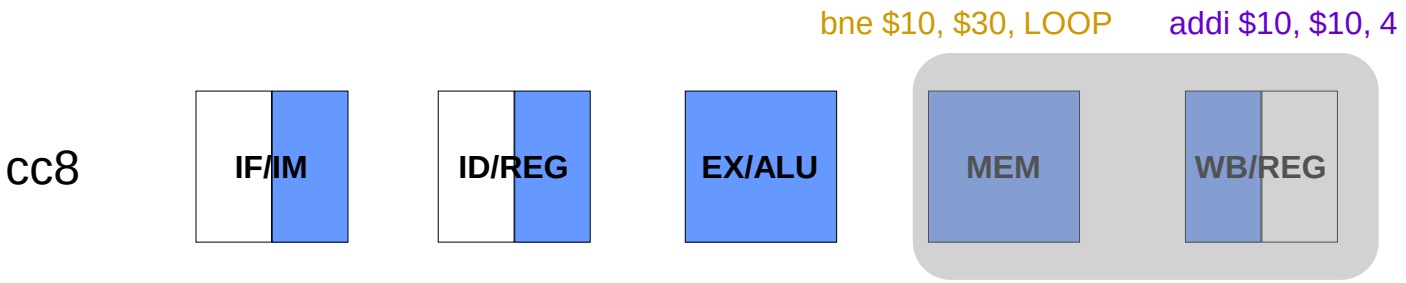
- A instrução SW não faz nada no estágio WB e sai do pipeline

- A instrução ADDI não faz nada no estágio MEM

- A instrução BNE está comparando o valores dos registradores: $10 != $30 ?

Aqui temos um problema com as instruções ADDI e BNE. O valor do registrador $10 ainda não foi escrito, então BNE não pode compará-lo!

ANALISANDO O CÓDIGO MIPS PARA IDENTIFICAR OS CONFLITOS/HAZARDS/DEPENDENCIAS



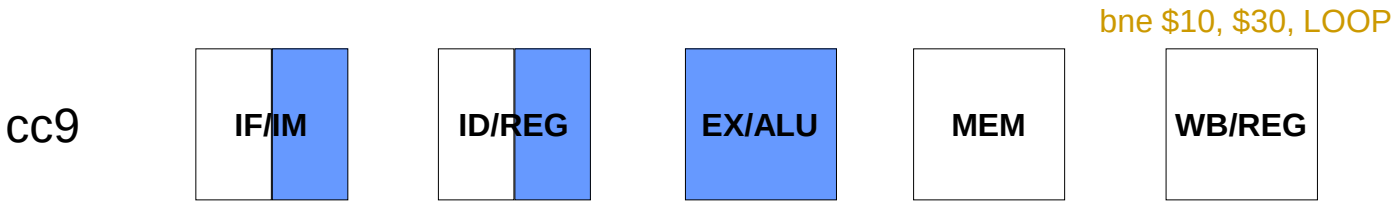
CICLO 8:

- A instrução ADDI escreve no registrador $10 e sai do pipeline

- A instrução BNE, no estágio MEM, realiza o desvio

Como temos o problema no ciclo anterior, ele persistirá aqui!

ANALISANDO O CÓDIGO MIPS PARA IDENTIFICAR OS CONFLITOS/HAZARDS/DEPENDENCIAS



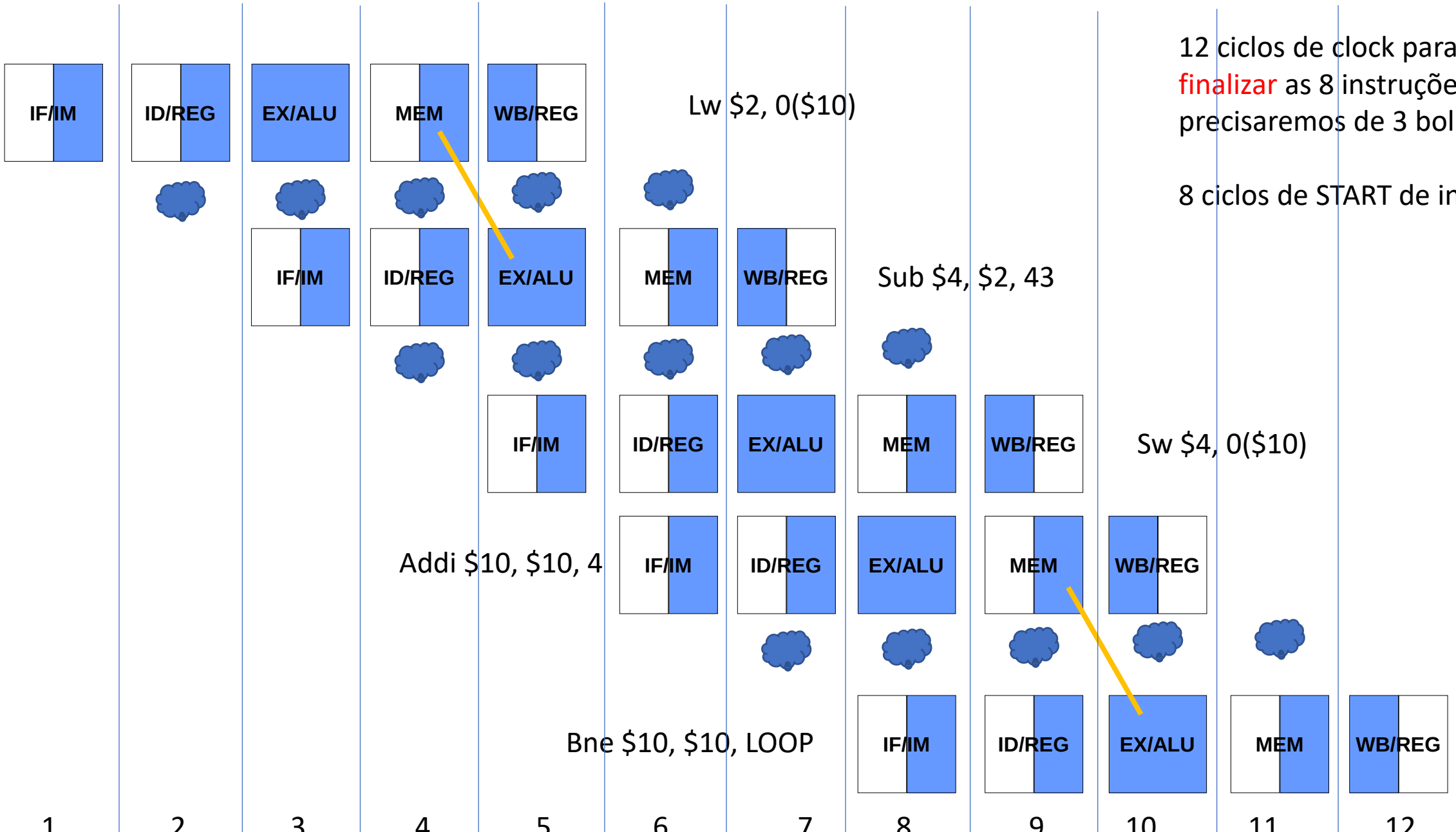
CICLO 9:

- A instrução BNE sai do pipeline

Concluímos que o código tem conflitos que precisam ser resolvidos com forwarding e bolhas. No entanto, o exercício diz que o pipeline não possui o mecanismo de forwarding. Os conflitos estão marcados no código a seguir:

```
LOOP:   Lw    $2, 0($10)
        Sub   $4, $2, $3
        Sw    $4, 0($10)
        Addi  $10, $10, 4
        Bne   $10, $30, LOOP
```

COM MECANISMO DE FORWARDING O PIPELINE DESTE CÓDIGO FICA DA SEGUINTE FORMA:



12 ciclos de clock para iniciar e finalizar as 8 instruções e precisaremos de 3 bolhas.

8 ciclos de START de instruções

SEM MECANISMO DE FORWARDING O PIPELINE DESTE CÓDIGO FICA DA SEGUINTE FORMA:

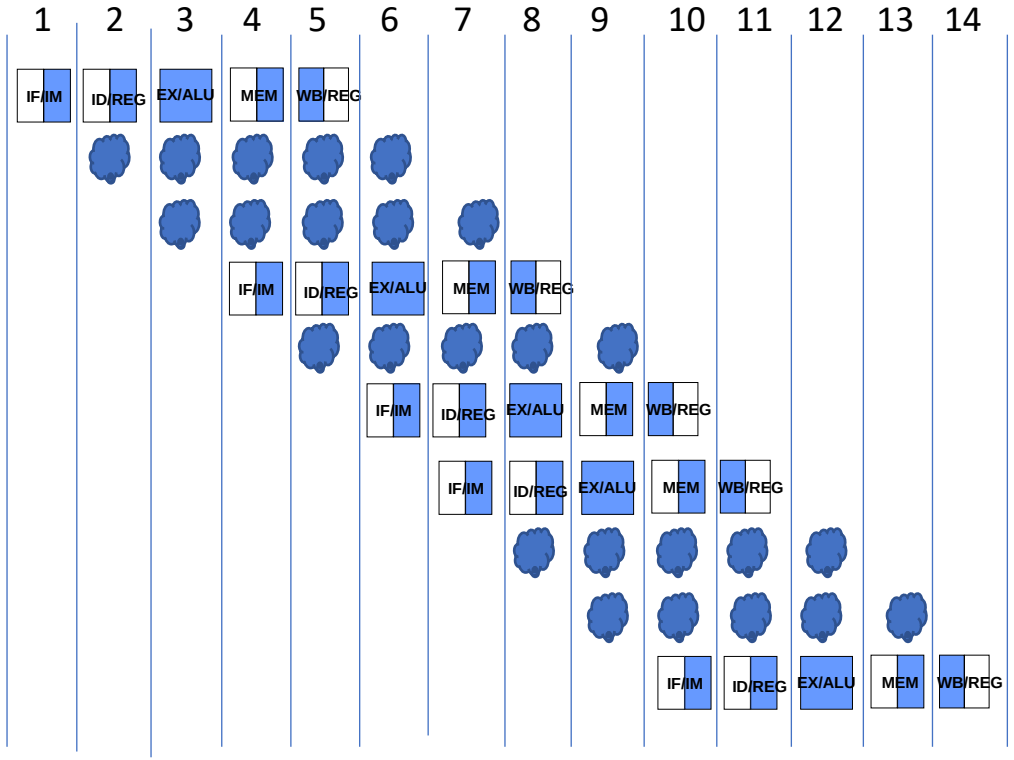
Lw $2, 0($10)

Sub $4, $2, $3

Sw $4, 0($10)

Addi $10, $10, 4

Bne $10, $10, LOOP



14 ciclos de clock para iniciar e finalizar as 10 instruções!
Precisaremos de 5 bolhas. São 10 ciclos de start.

COM MECANISMO DE FORWARDING, E SEM QUALQUER ESCALONAMENTO, O CÓDIGO FICA DA SEGUINTE FORMA:

```
LOOP:   1. LW     $2, 9($10)
        2. BOLHA
        3. SUB    $4, $2, $3
        4. BOLHA
        5. SW     $4, 0($10)
        6. ADDI   $10, $10, 4
        7. BOLHA
        8. BNE    $10, $30, LOOP
```

SEM MECANISMO DE FORWARDING, E SEM QUALQUER ESCALONAMENTO, O CÓDIGO FICA DA SEGUINTE FORMA:

```
LOOP:   1. LW     $2, 9($10)
        2. BOLHA
        3. BOLHA
        4. SUB    $4, $2, $3
        5. BOLHA
        6. SW     $4, 0($10)
        7. ADDI   $10, $10, 4
        8. BOLHA
        9. BOLHA
       10. BNE    $10, $30, LOOP
```

Estes códigos são apenas reflexo dos diagramas dos slides anteriores

ESCALONAMENTO:

- Técnica usada para manter o pipeline cheio;
- Encontra sequências de instruções não relacionadas que podem ser sobrepostas;
- Evita bolhas:
  - Uma instrução dependente precisa ser separada da instrução de origem por uma distância em ciclos de clock igual à latência do pipeline dessa instrução de origem

COM MECANISMO DE FORWARDING, E COM ESCALONAMENTO, O CÓDIGO FICA DA SEGUINTE FORMA:

```
LOOP:   1. LW     $2, 0($10)
        2. ADDI   $10, $10, 4
        3. SUB    $4, $2, $3
        4. BOLHA
        5. SW     $4, 8($10)
        6. BNE    $10, $30, LOOP
```

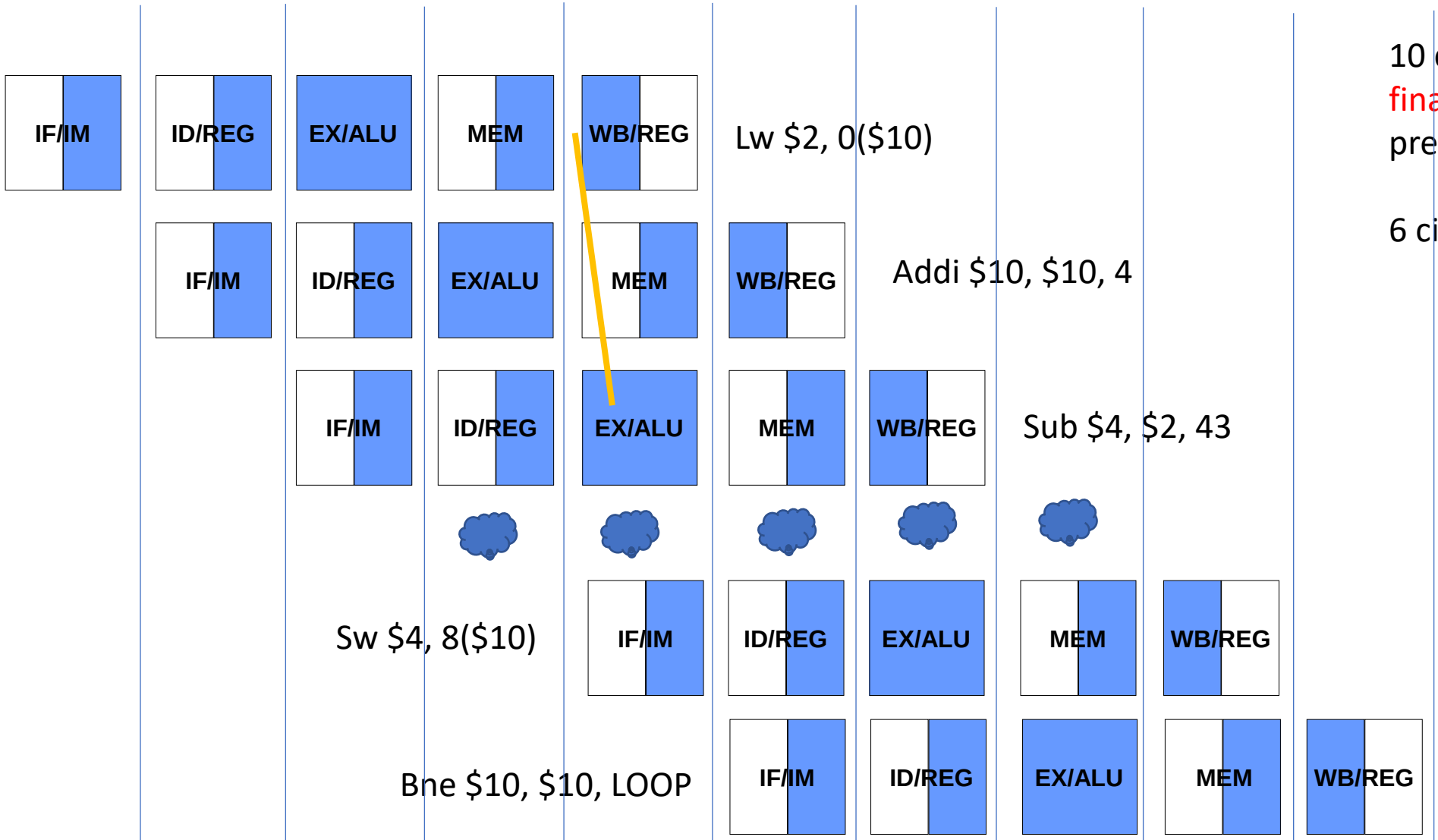
SEM MECANISMO DE FORWARDING, E COM ESCALONAMENTO, O CÓDIGO FICA DA SEGUINTE FORMA:

```
LOOP:   1. LW     $2, 0($10)
        2. ADDI   $10, $10, 4
        3. BOLHA
        4. SUB    $4, $2, $3
        5. BOLHA
        6. BOLHA
        7. SW     $4, 8($10)
        8. BNE    $10, $30, LOOP
```

Vamos ver como fica o diagrama pra esses dois códigos!

O 8 na instrução SW em ambos os códigos, é por conta que pulamos 2 instruções: 0 + 4 + 4. A bolha não conta! Para escalonar precisamos prestar atenção nestes detalhes!

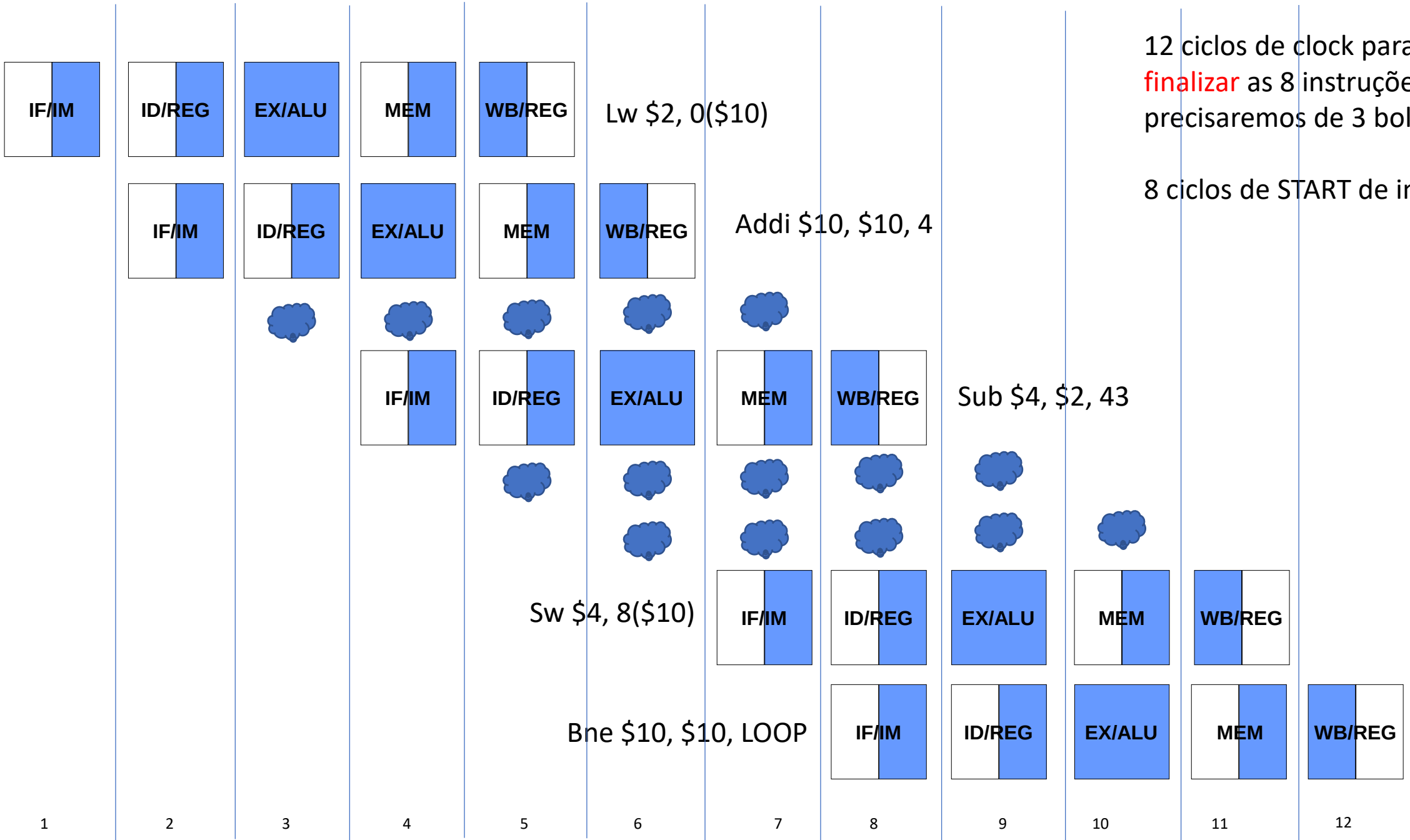
COM MECANISMO DE FORWARDING E COM ESCALONAMENTO



10 ciclos de clock para iniciar e finalizar as 6 instruções e precisaremos de 1 bolha.

6 ciclos de START de instruções

SEM MECANISMO DE FORWARDING E COM ESCALONAMENTO



12 ciclos de clock para iniciar e finalizar as 8 instruções e precisaremos de 3 bolhas.

8 ciclos de START de instruções

DESDOBRANDO O LOOP EM 4 CÓPIAS

```
LOOP:  1. Lw    $2, 0($10)
       2. Sub   $4, $2, $3
       3. Sw    $4, 0($10)

       4. Lw    $6, 8($10)
       5. Sub   $8, $6, $3
       6. Sw    $8, 8($10)

       7. Lw    $12, 16($10)
       8. Sub   $14, $12, $3
       9. Sw    $14, 16($10)

      10. Lw    $16, 24($10)
      11. Sub   $18, $16, $3
      12. Sw    $18, 24($10)

      13. Addi  $10, $10, 32
      14. Bne   $10, $30, LOOP
```

Para desdobrar o código você precisa fazer uma cópia do CORPO do LOOP renomeando os registradores e modificando o índice que acompanha o endereço base da instrução de load/store.

As instruções referentes ao incremento do LOOP e à decisão do DESVIO não entram na cópia!

O código à esquerda já é o código desdobrado. Cada parte replicada está com uma cor diferente para destacar as mudanças.

Agora temos 14 instruções que vão encher o pipeline e com isso as bolhas não serão mais necessárias.

Ainda assim, do jeito que está aqui teremos bolhas. Vamos reordenar para evitar isto!

DESDOBRANDO O LOOP EM 4 CÓPIAS

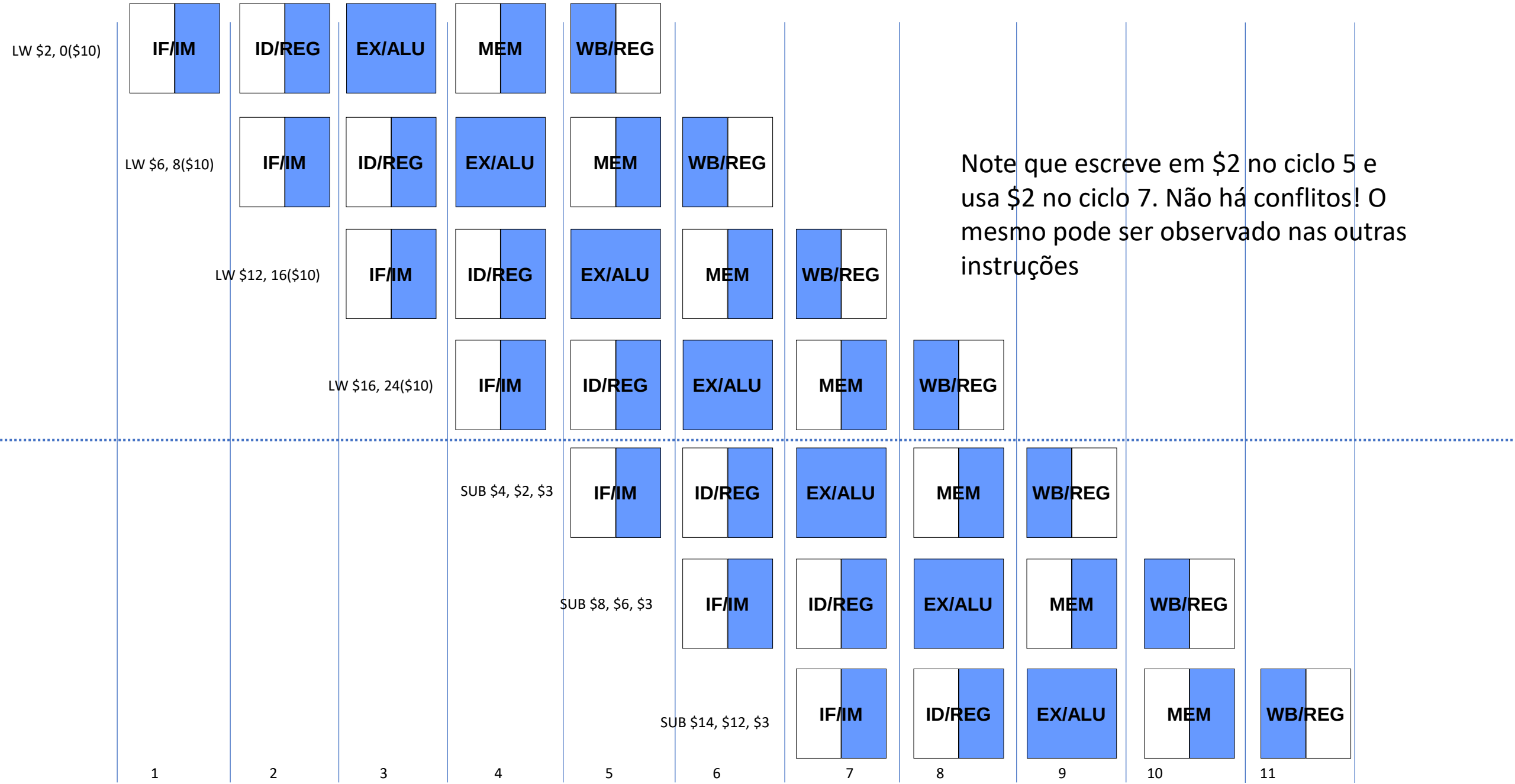
```
LOOP:  1. Lw   $2, 0($10)
       2. Lw   $6, 8($10)
       3. Lw   $12, 16($10)
       4. Lw   $16, 24($10)

       5. Sub  $4, $2, $3
       6. Sub  $8, $6, $3
       7. Sub  $14, $12, $3
       8. Sub  $18, $16, $3

       9. Sw   $4, 0($10)
      10. Sw   $8, 8($10)
      11. Sw   $14, 16($10)
      12. Sw   $18, 24($10)

      13. Addi  $10, $10, 4
      14. Bne   $10, $30, LOOP
```

Depois de desdobrar o código, reordene de forma que todas as instruções de LOAD estejam no inicio e todas as instruções de STORE no final.

DESDOBRAMENTO:

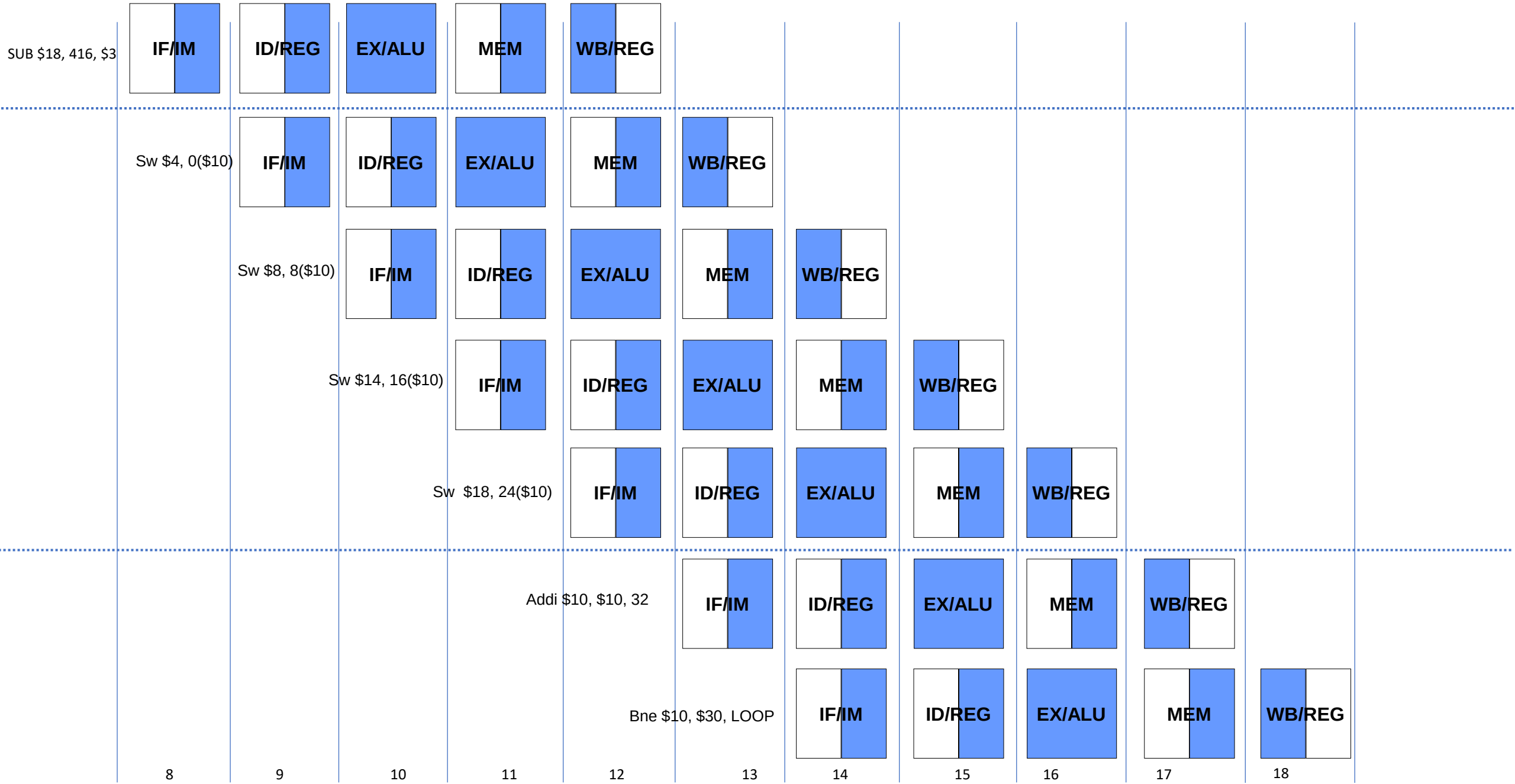
- É uma técnica para aumentar o número de instruções relativas às instruções de desvio
- Replica o corpo do LOOP várias vezes e ajusta o final
- Remove as instruções de desvio do meio do caminho
- Permite que instruções de diferentes iterações sejam escalonadas juntas
- Necessário renomear os registradores
- Considerar apenas o corpo do LOOP para a replicação
- Evita bolhas

DIAGRAMA CÓDIGO DESDOBRADO E ESCALONADO



Note que escreve em $2 no ciclo 5 e usa $2 no ciclo 7. Não há conflitos! O mesmo pode ser observado nas outras instruções

DIAGRAMA CÓDIGO DESDOBRADO E ESCALONADO



- Desdobramos o código em quatro vezes
- Consideramos que o circuito não tem forwarding
- Consideramos que os desvios são resolvidos no estágio MEM
- Consideramos que o circuito admite a instrução ADDI
- Mostramos o código desdobrado e escalonado
- Como o código escalonado se compara com o código não escalonado original em termos de execução, isto é, em termos de speed up?

Código Original

14 ciclos de clock para iniciar e finalizar as 10 instruções! Precisaremos de 5 bolhas. São necessário 10 ciclos de clock de start.

Código Desdobrado e Escalonado

18 ciclos de clock para iniciar e finalizar as 14 instruções! Não precisaremos de bolhas. São necessários 14 ciclos de clock de start.