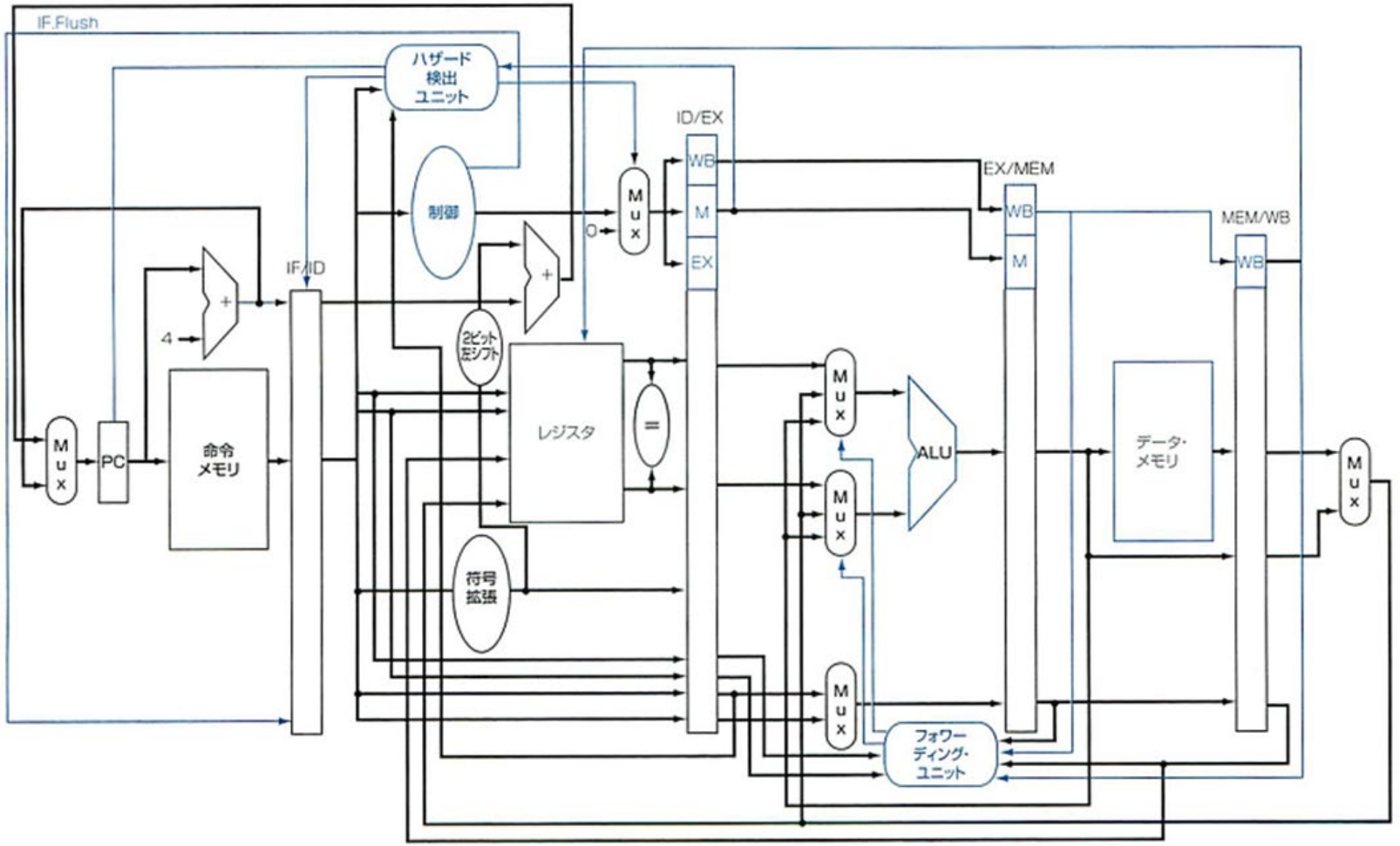
CSL
Computer Systems Laboratory

情報科学演習
プロセッサ演習
～第２週～

情報科学科
コンピュータシステム研究室

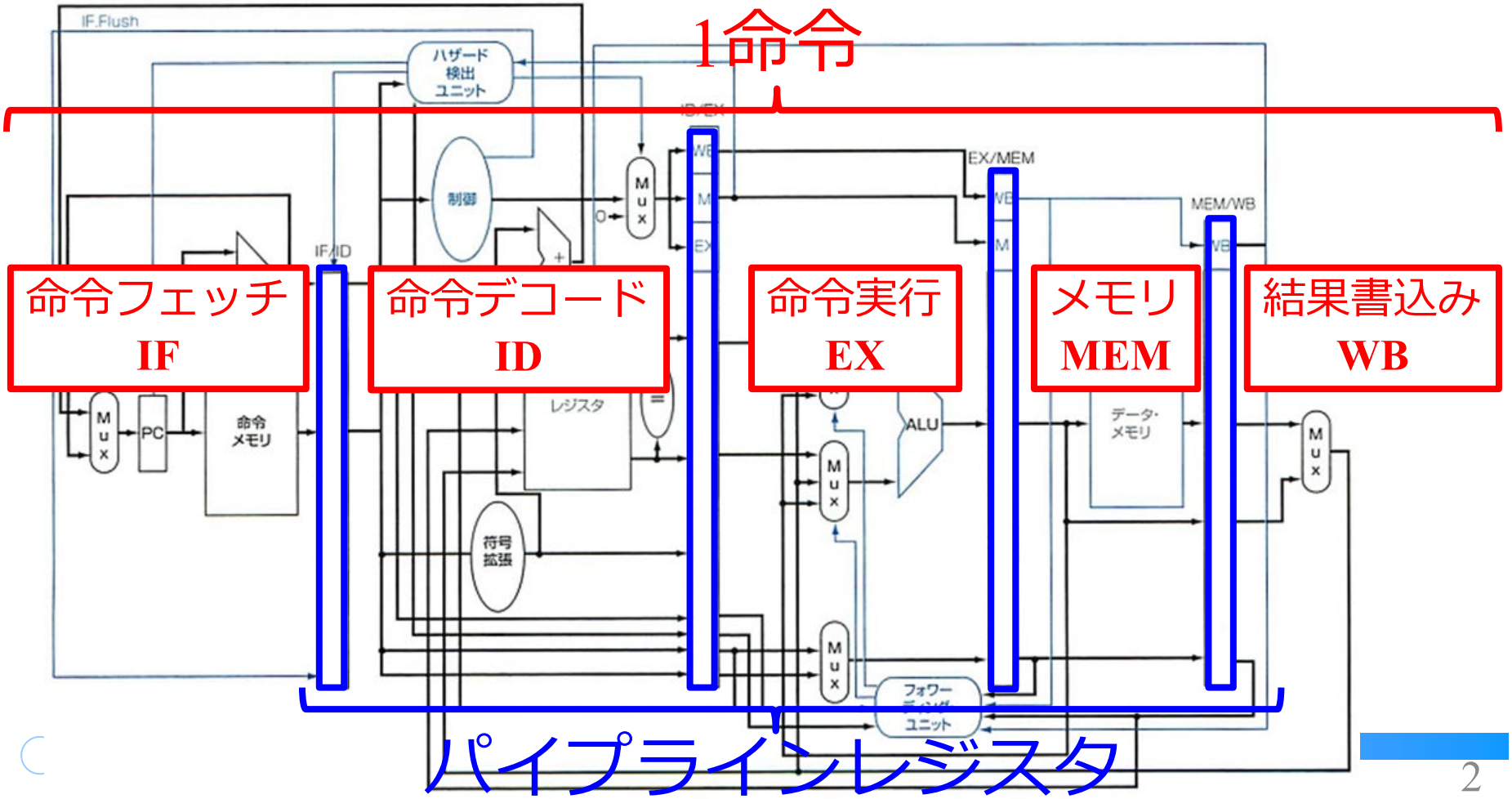
演習で使用する回路

■5段パイプラインプロセッサ



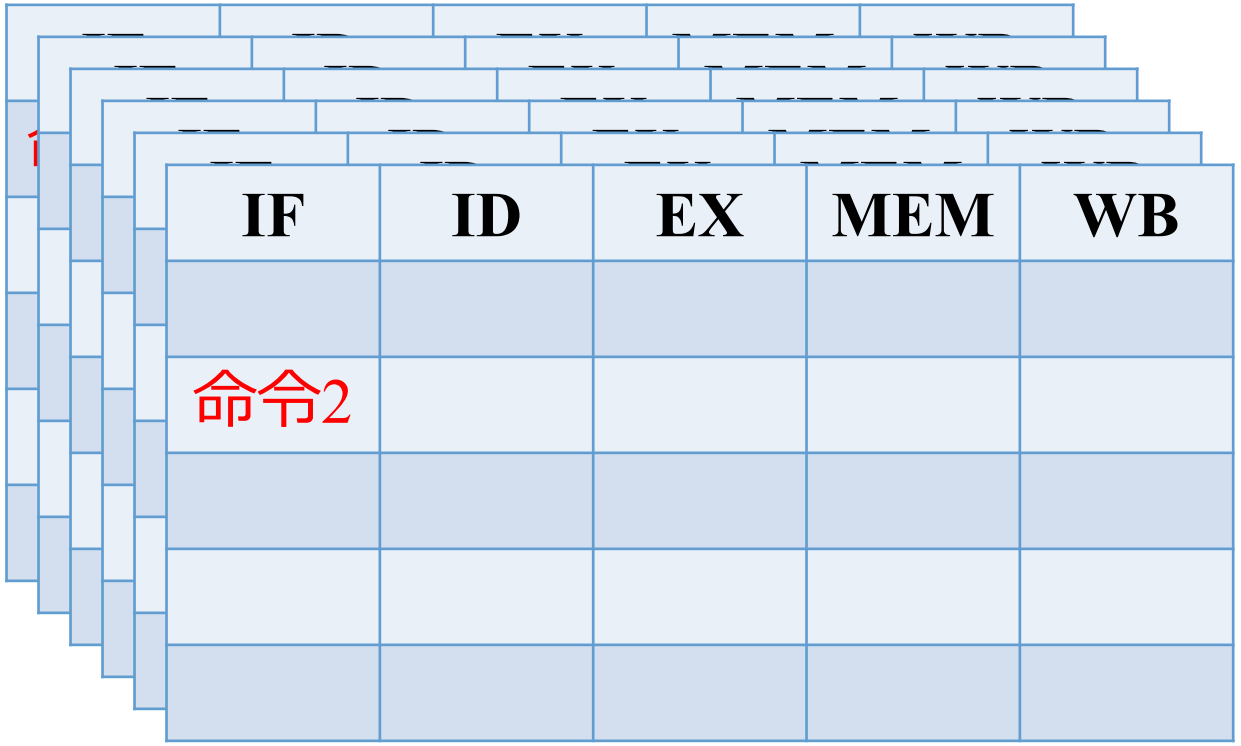
演習で使用する回路

■5段パイプラインプロセッサ



非パイプラインの場合

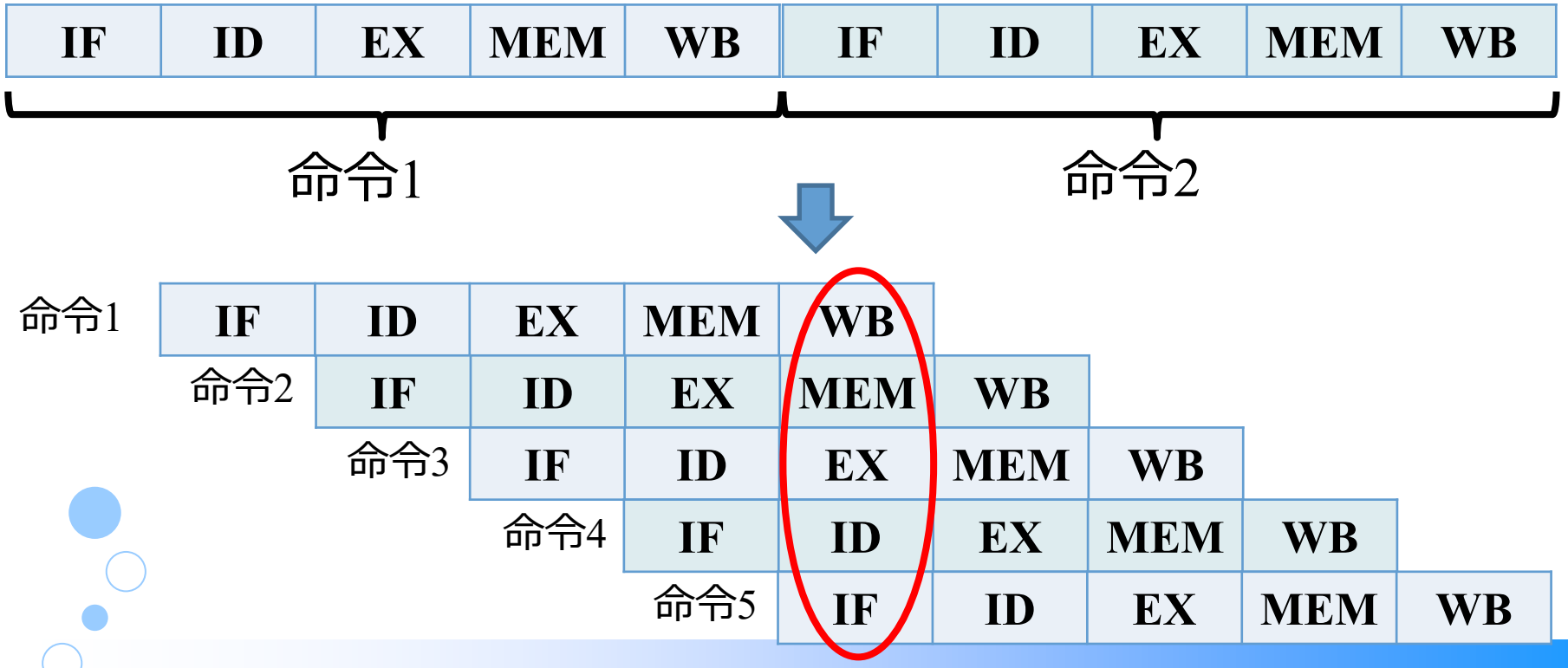
■命令の流れ



各ステージの実行中に他のステージは未使用

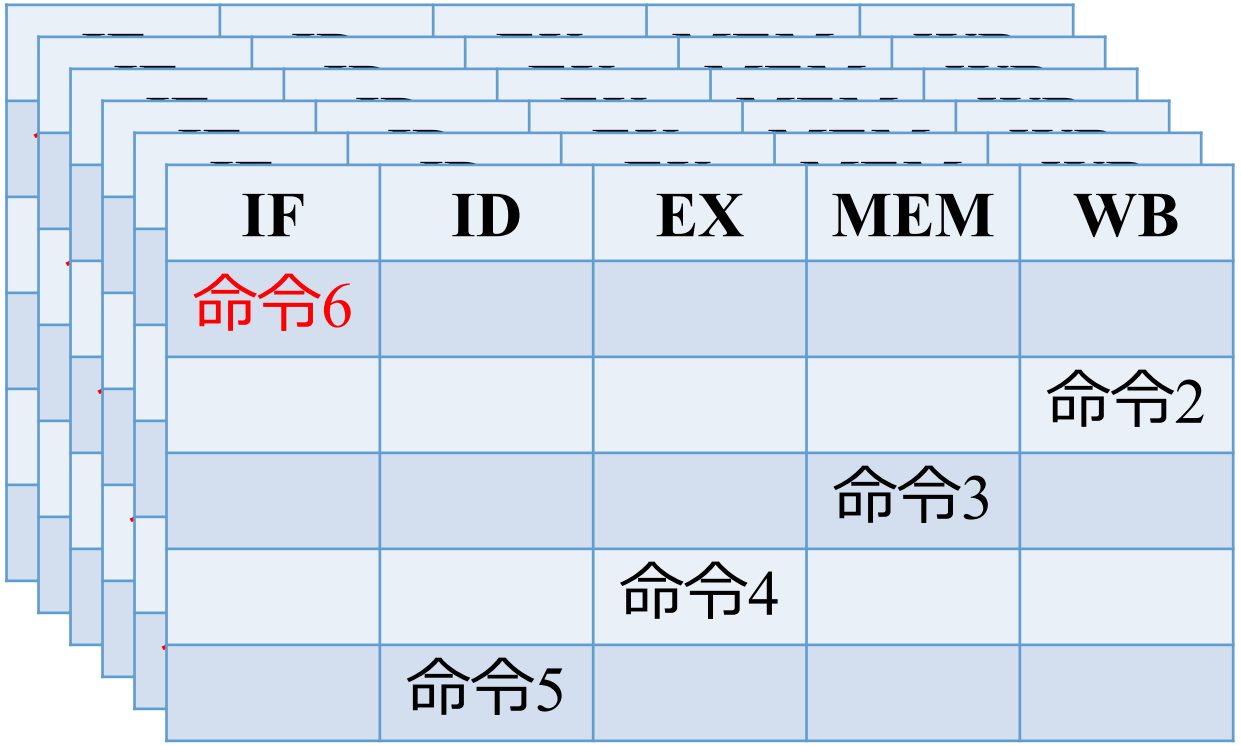
パイプライン処理

- 複数の命令を少しずつずらし並行的に実行
- スループットを向上



パイプライン処理

■命令の流れ



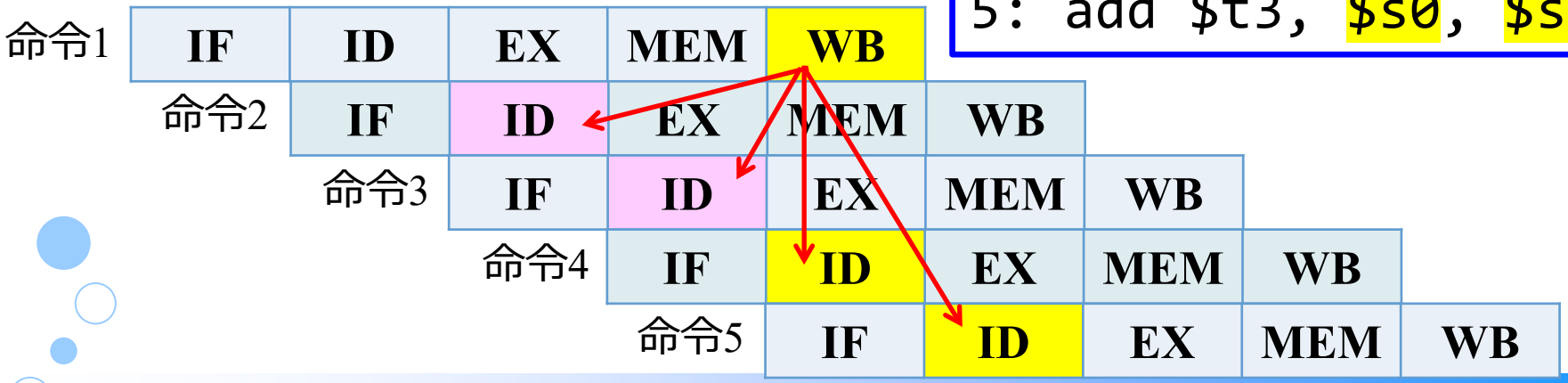
命令が読み出されるたびに常に各ステージの回路が使用される

データハザード

- 先行命令の結果を後続命令が使用
- 最新の結果を使用できない場合がある

命令1：WBで結果を書き込み
命令2：**間に合わない**
命令3：**間に合わない**
命令4：間に合う
命令5：間に合う

```
1: add $s0, $s1, $s2
2: add $t0, $s0, $s3
3: add $t1, $s4, $s0
4: add $t2, $s0, $s0
5: add $t3, $s0, $s0
```

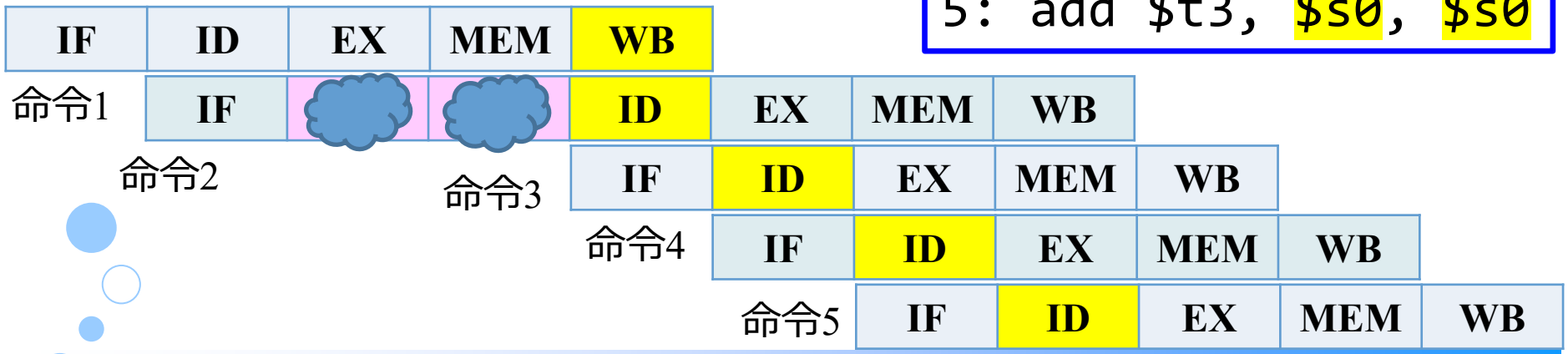


対処方法（ストール）

- ハザードが解消されるまで後続命令を停止
- 後続命令は最新の結果を使用可能
- 無駄なサイクルが発生

```
1: add $s0, $s1, $s2
2: add $t0, $s0, $s3
3: add $t1, $s4, $s0
4: add $t2, $s0, $s0
5: add $t3, $s0, $s0
```

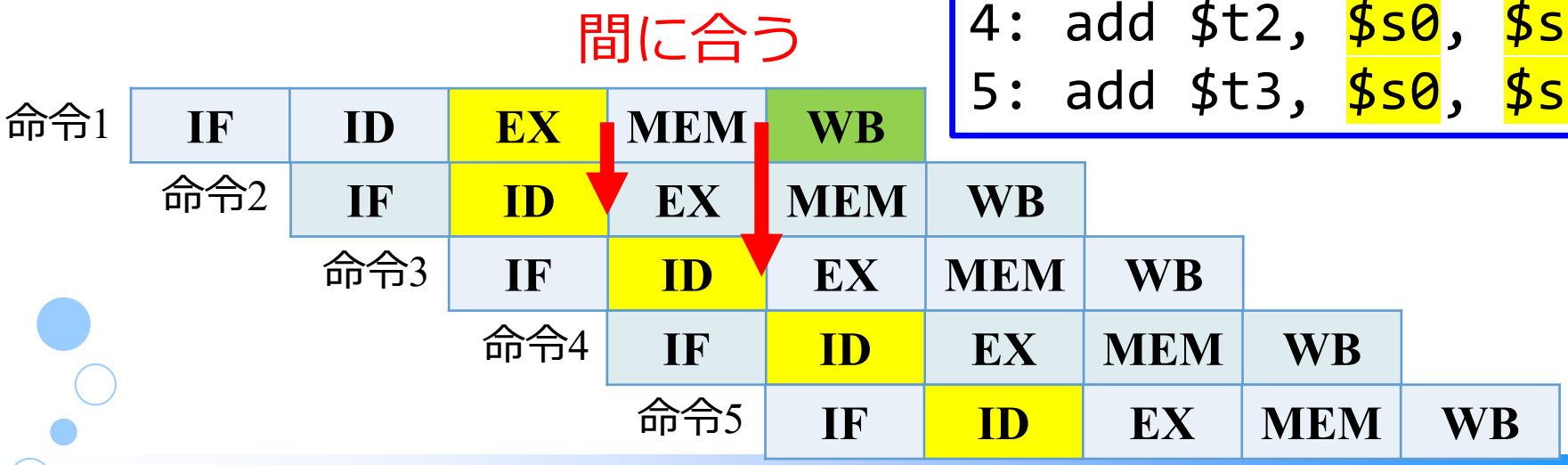
ストールはプロセッサが
自動的に挿入する



対処方法（フォワーディング）

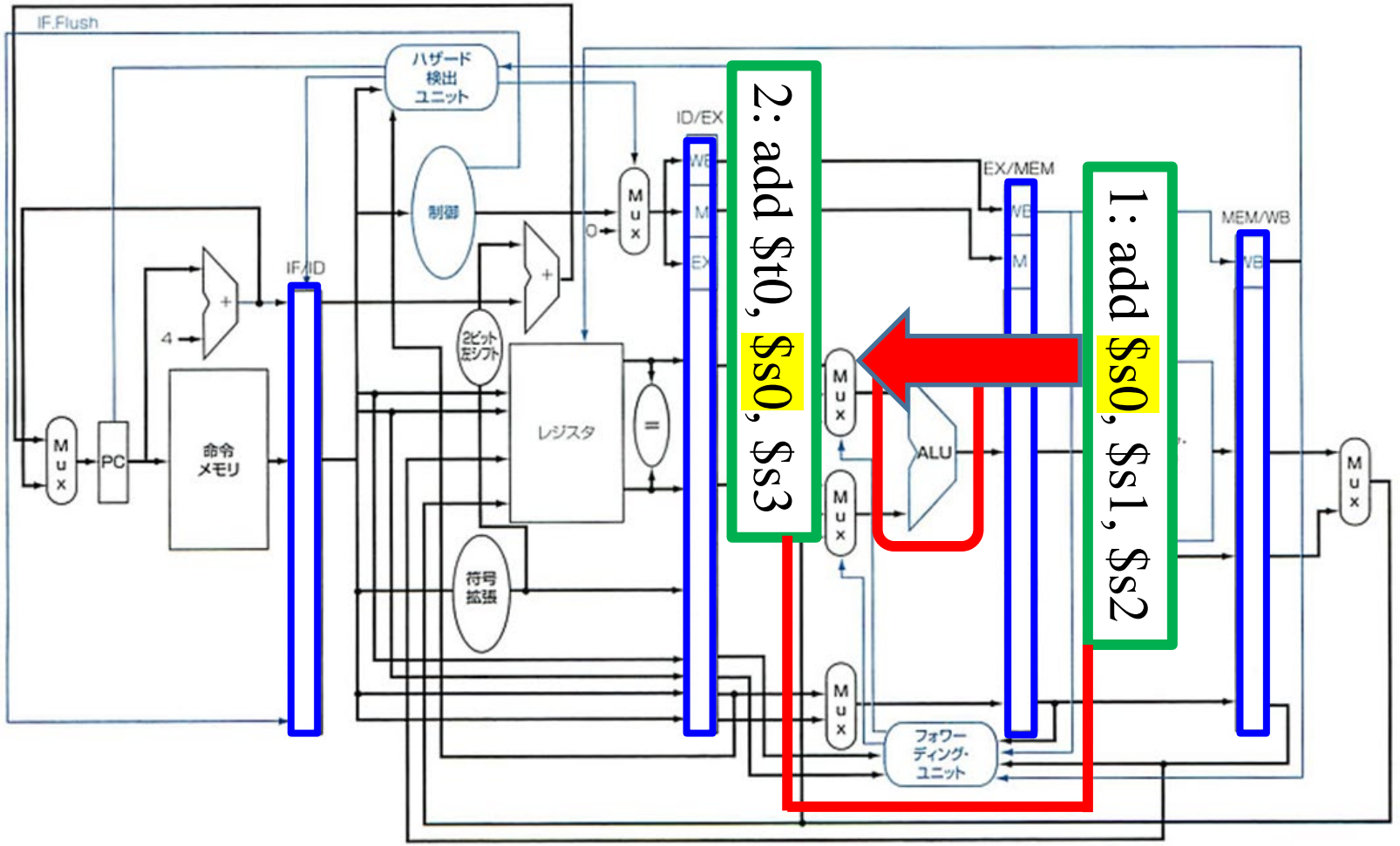
- 演算の結果はEX終了後に決定
- その結果を後続命令にバイパス
- ストールは発生しない

```
1: add $s0, $s1, $s2
2: add $t0, $s0, $s3
3: add $t1, $s4, $s0
4: add $t2, $s0, $s0
5: add $t3, $s0, $s0
```



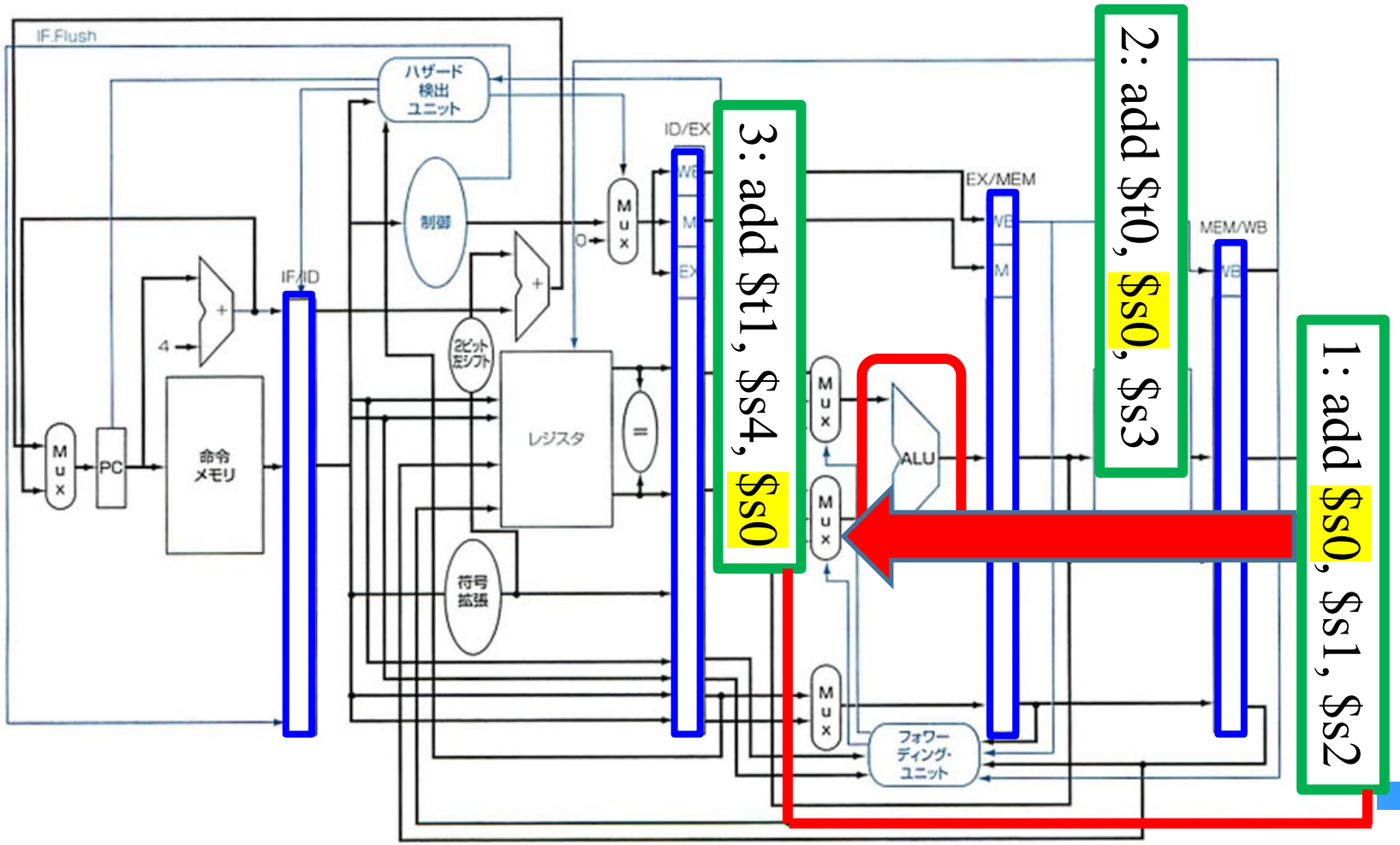
フォワーディング

- 演算結果を後続命令にバイパス



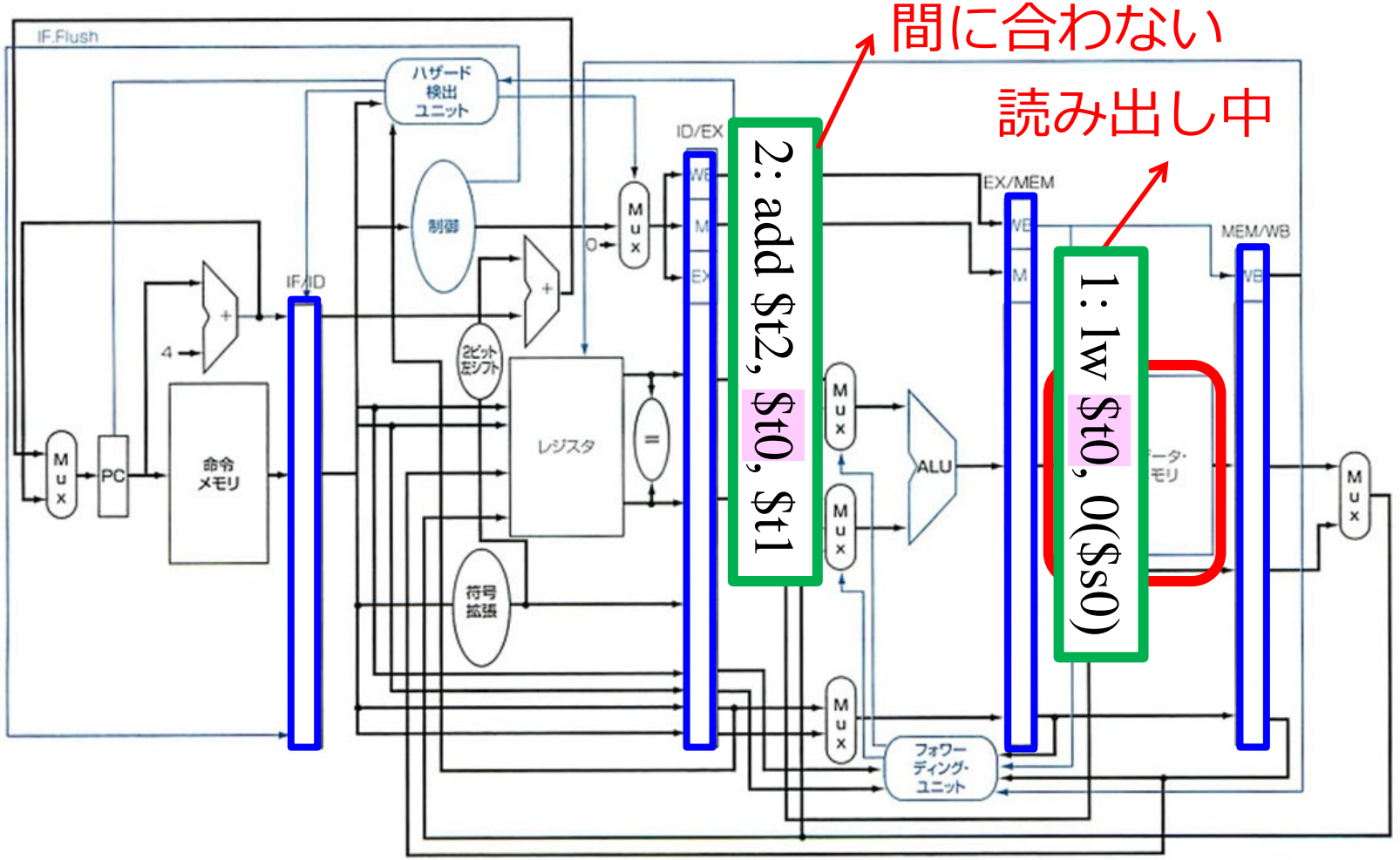
フォワーディング

- 演算結果を後続命令にバイパス



ロードハザード

■ ロードの結果を直後の命令は使用できない



対処方法（命令の挿入）

- ロードの結果を直後の命令は使用できない
  - フォワーディングでも間に合わない

```
lw  $t0,0($s0)
add $t2,$t0,$t1
```

後続命令は 1 サイクルのストール
（プロセッサが自動的に挿入）

- ロード命令の後に他の命令を入れる
  - ロードの結果に依存しない命令

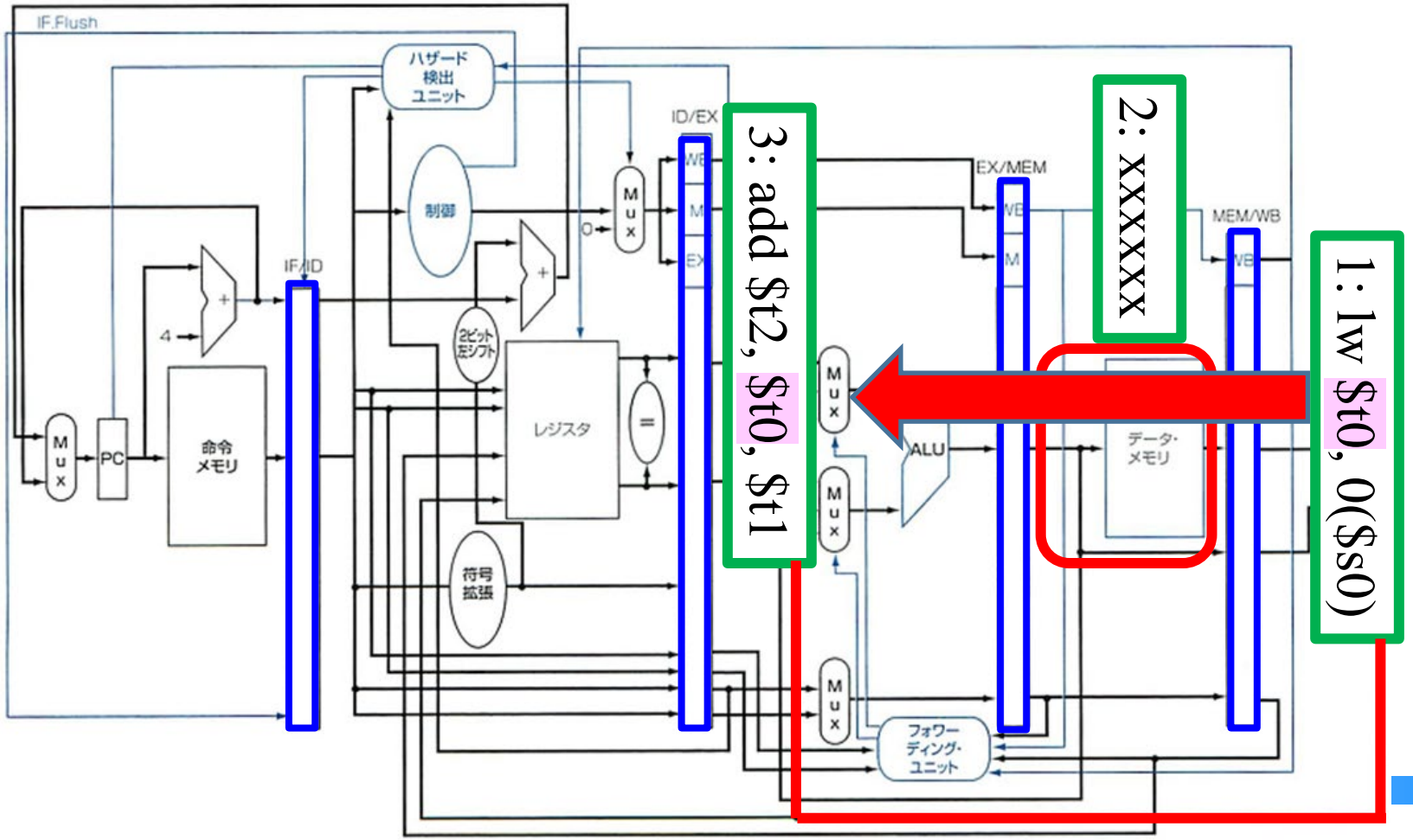
```
lw  $t0,0($s0)
add $t3,$t4,$t5
add $t2,$t0,$t1
```

ロードの結果は間に合う

ストールが起きずに実行可能

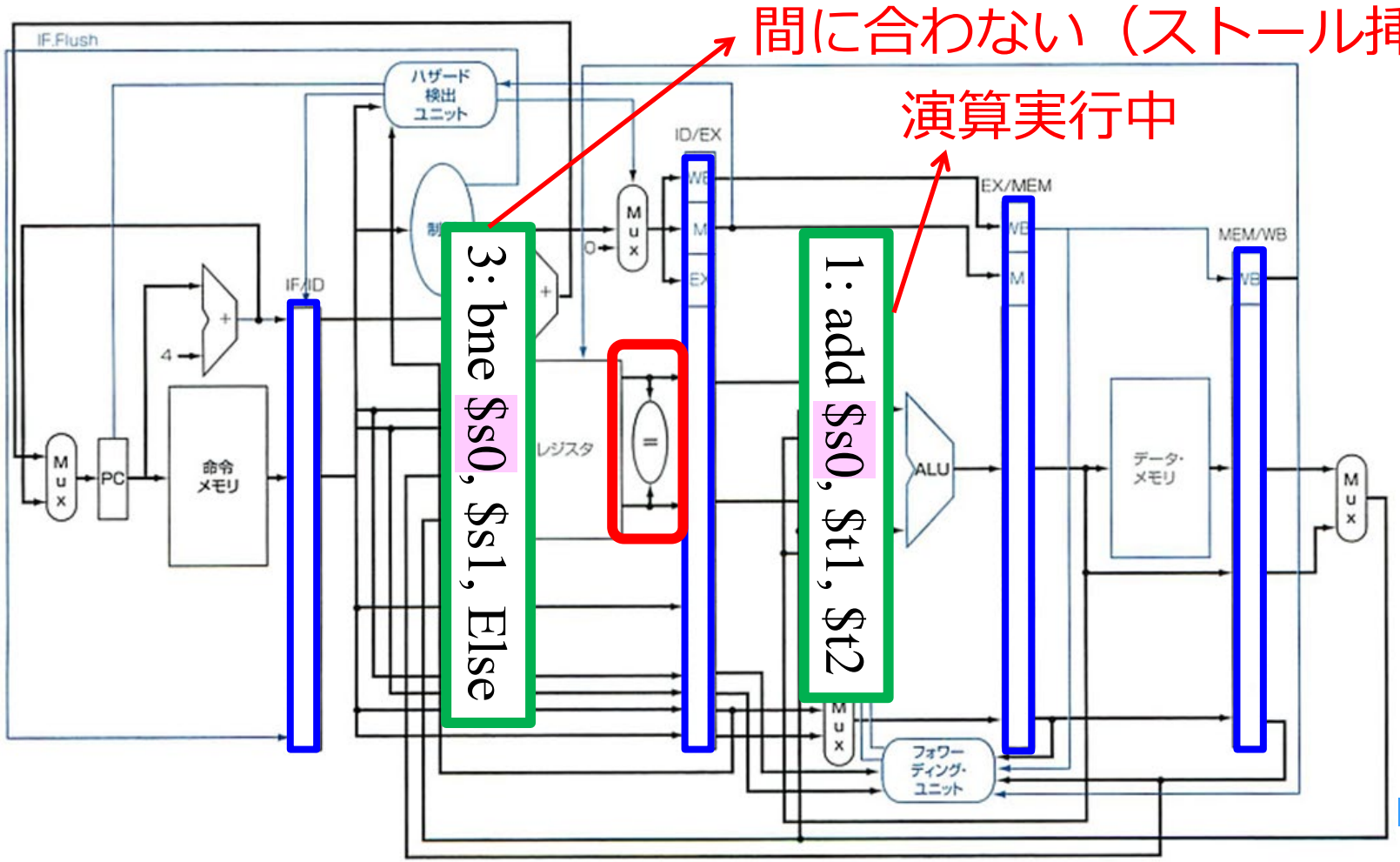
命令の挿入

- ロード命令の後に他の命令を入れる



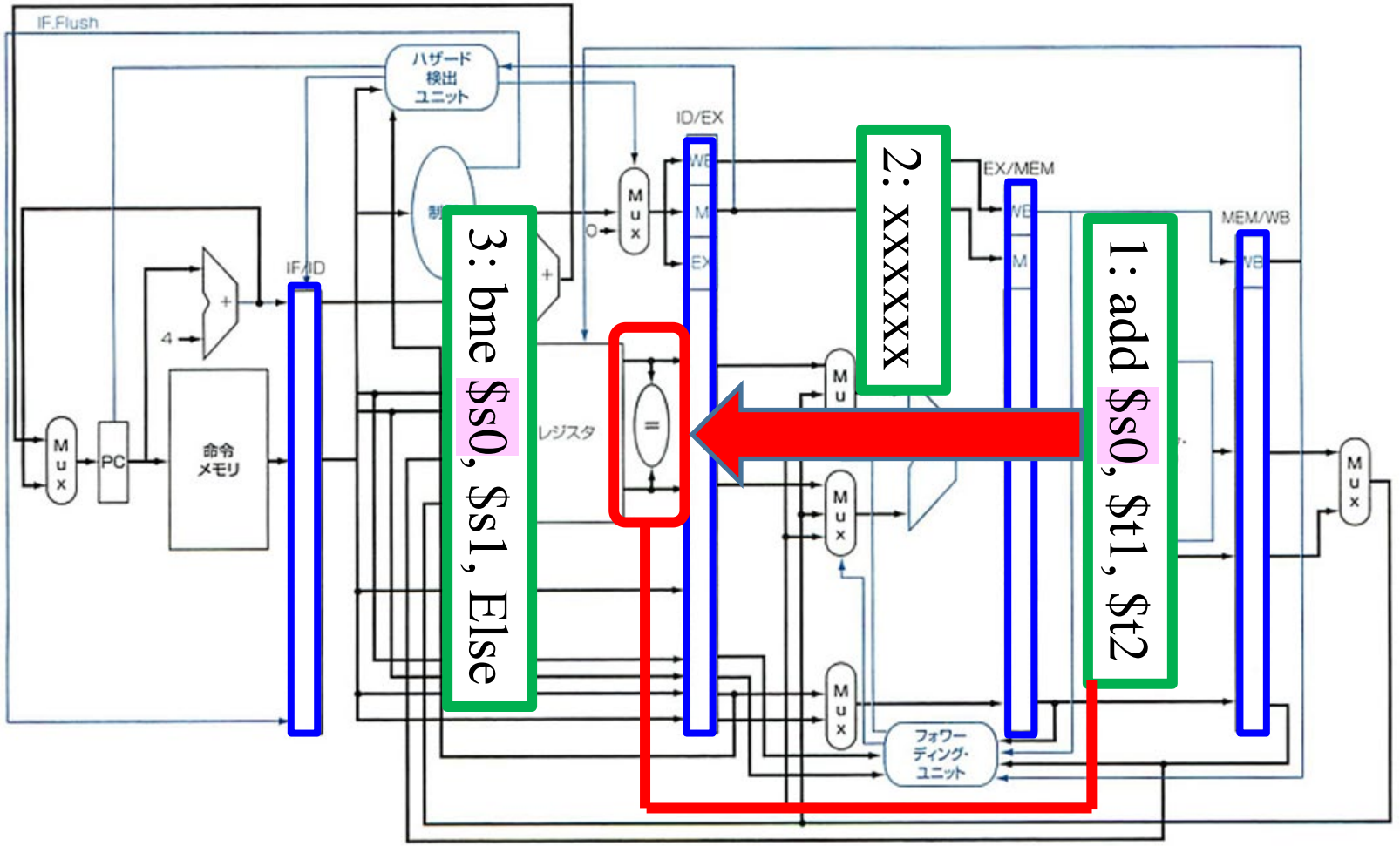
分岐命令のデータハザード

■直前の命令の結果を分岐判定で使えない



対処方法（命令の挿入）

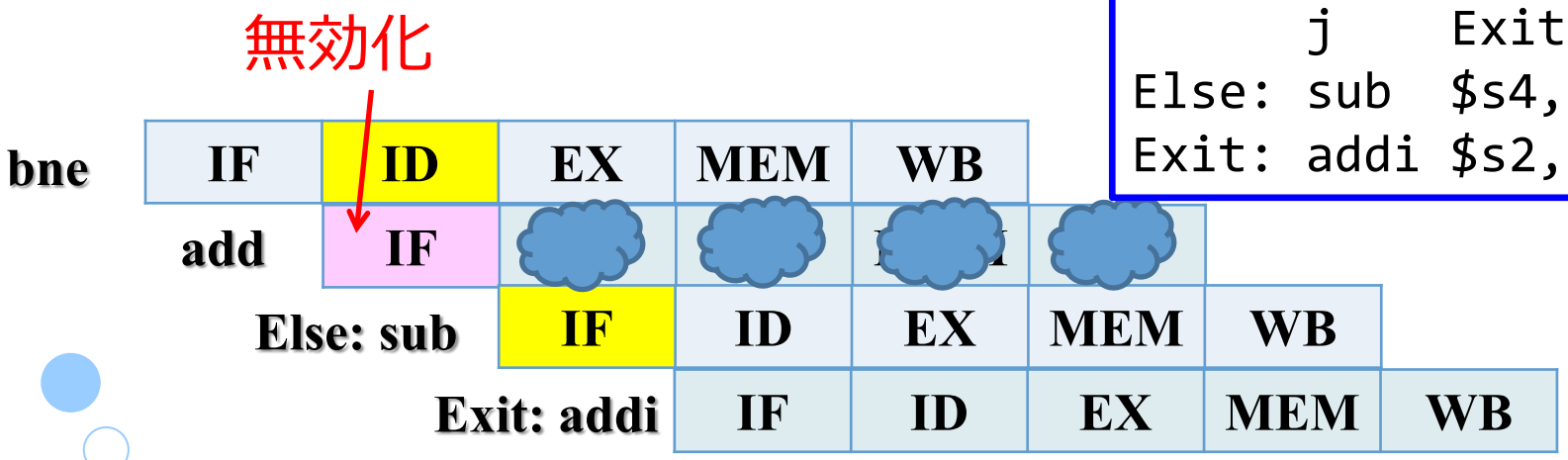
■分岐命令の直前に他の命令を入れる



制御ハザード

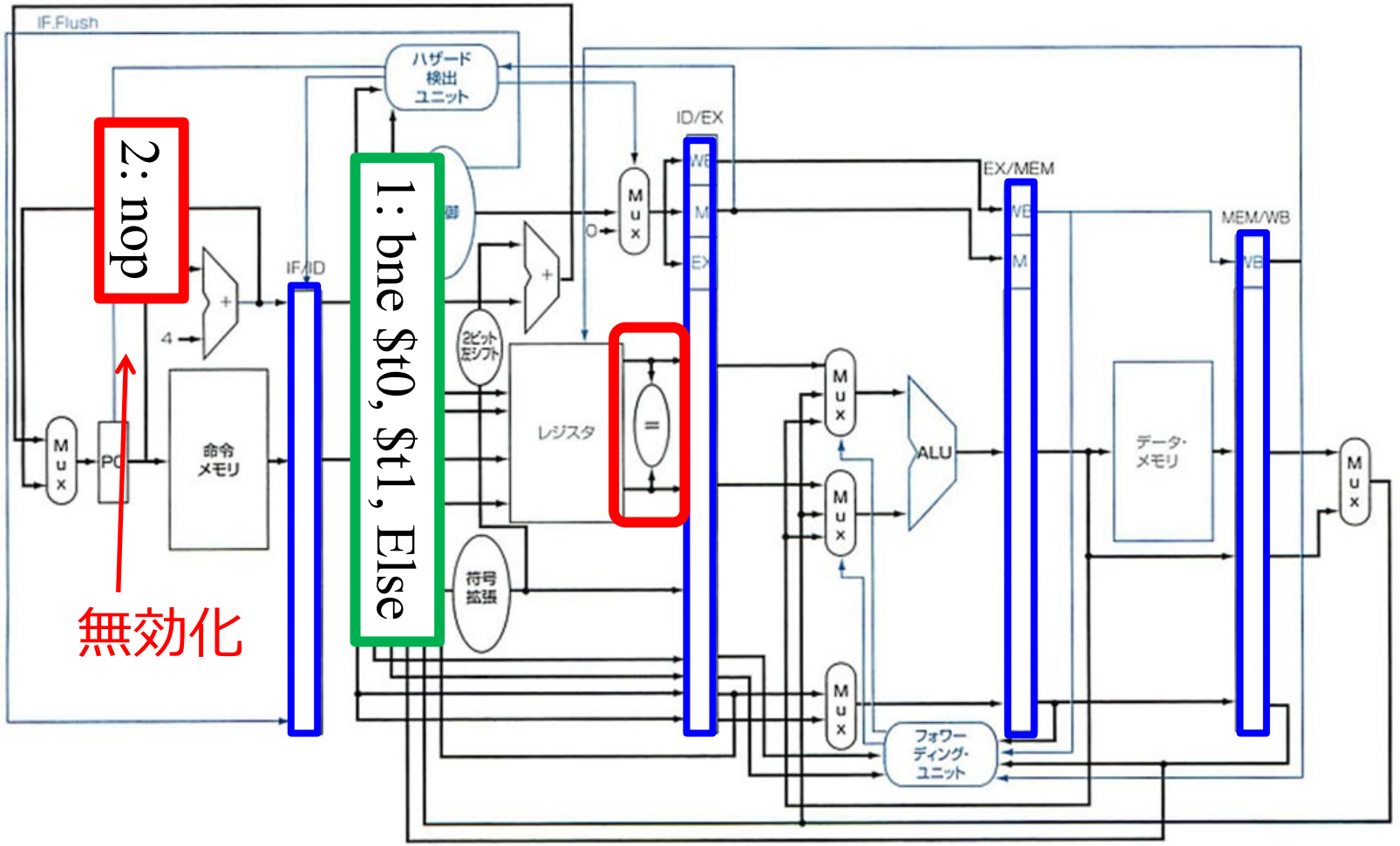
- 分岐が確定するまで後続命令を読み出し
- 分岐不成立の場合は問題なし
- 分岐成立の場合は無効化が必要
- ジャンプ命令も同様

```
        bne  $s1,$s2,Else
        add  $s4,$s1,$s3
        j    Exit
Else:   sub  $s4,$s1,$s3
Exit:   addi $s2,$s2,1
```



対処方法（後続命令の無効化）

- 後続命令を何もしない命令(nop命令)に変換

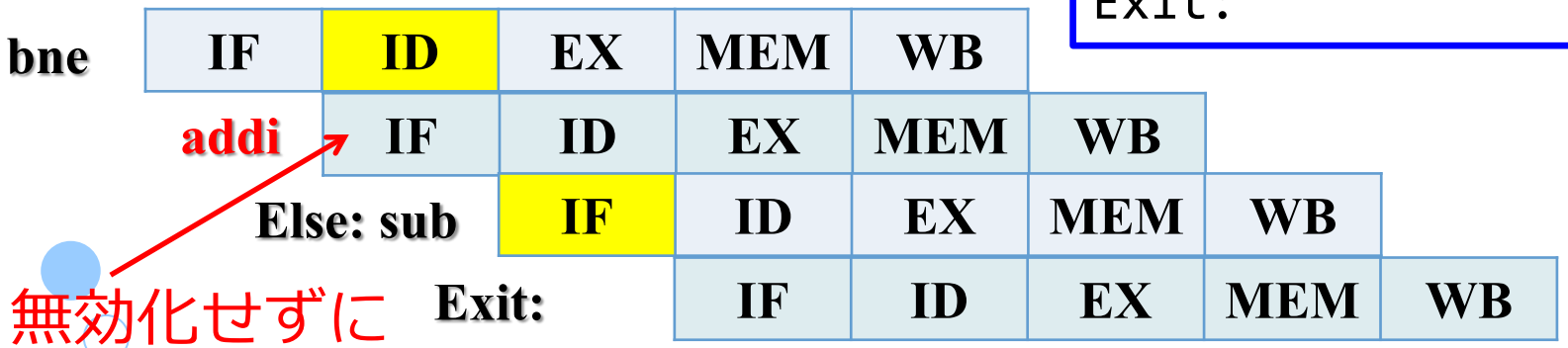


遅延分岐

- 分岐命令の直後の命令（遅延スロット）
- 遅延スロットは常に実行：他の命令を挿入

```
1  1          bne   $s1,$s2,Else
2             add   $s4,$s1,$s3
3             j     Exit
   2  Else:   sub   $s4,$s1,$s3
4  3  Exit:   addi  $s2,$s2,1
```

```
1  1          bne   $s1,$s2,Else
2  2          addi  $s2,$s2,1
3             add   $s4,$s1,$s3
4             j     Exit
   3  Else:   sub   $s4,$s1,$s3
        Exit:
```



無効化せずに
実行可能

遅延分岐

- ジャンプ命令も同様
- 遅延スロットに他の命令を挿入

```
1  1          bne  $s1,$s2,Else
2  2          addi $s2,$s2,1
3             add  $s4,$s1,$s3
4             j    Exit
5  3  Else:   sub  $s4,$s1,$s3
   Exit:
```

実行されてしまう

```
1  1          bne  $s1,$s2,Else
2  2          addi $s2,$s2,1
3             j    Exit
4             add  $s4,$s1,$s3
   3  Else:   sub  $s4,$s1,$s3
5  4  Exit:
```

遅延分岐

■ 設定なしの場合：
後続命令をストール
（自動的に挿入）

```
        bne   $s1,$s2,Else
              
        add   $s4,$s1,$s3
        j     Exit
              
Else:   sub   $s4,$s1,$s3
Exit:   addi  $s2,$s2,1
```

- 遅延スロットに nop 命令を入れると
実行結果は元のコードと同じ
- 遅延スロットを有効に使えば高速

■ 設定ありの場合
後続命令を必ず実行

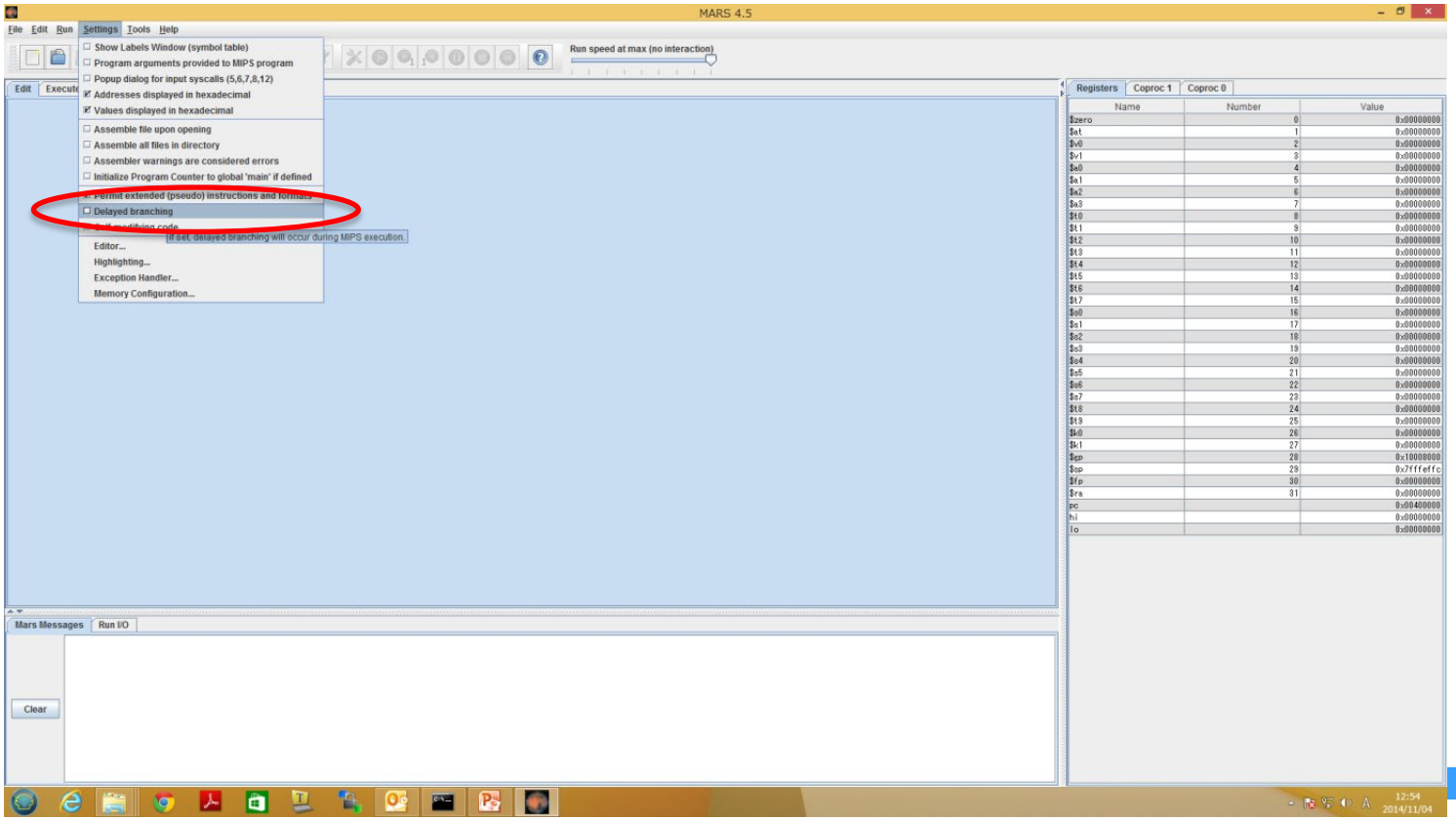
```
        bne   $s1,$s2,Else
        nop
        add   $s4,$s1,$s3
        j     Exit
        nop
Else:   sub   $s4,$s1,$s3
Exit:   addi  $s2,$s2,1
```

```
        bne   $s1,$s2,Else
        addi  $s2,$s2,1
        j     Exit
        add   $s4,$s1,$s3
Else:   sub   $s4,$s1,$s3
Exit:
```

MARSの使い方

■MARSの設定(遅延分岐を設定する場合)

- Settings > Delayed branchingにチェック



アセンブリプログラムの変換

■作成

- File > Dump Memory
- Dump Format > Text/Data Segment Window
- 「rom.txt」(任意のファイル名)で保存

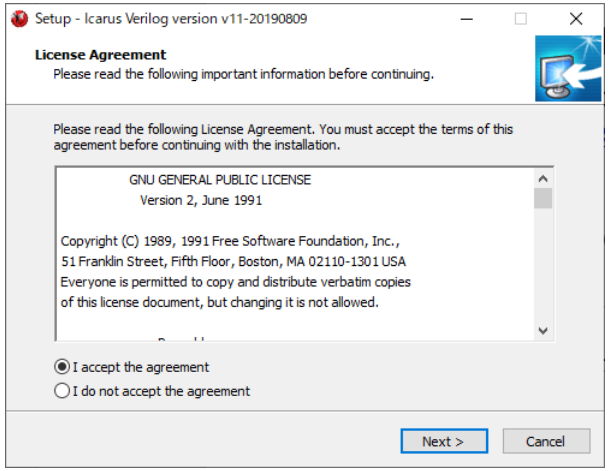
■変換

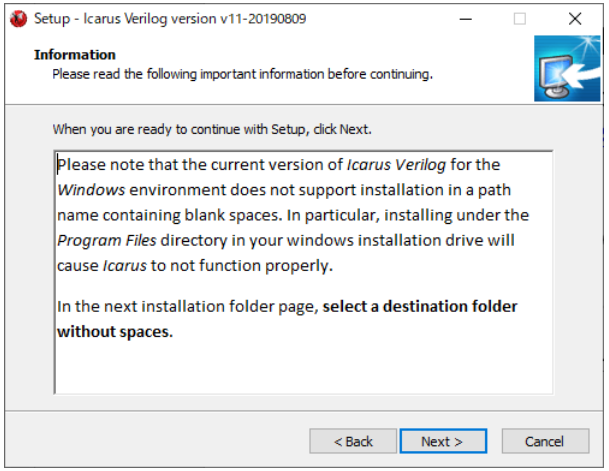
- perl rom.pl rom.txt ➔ rom.v が出力される
- 変換された rom.v をHDLソースがあるディレクトリにコピー

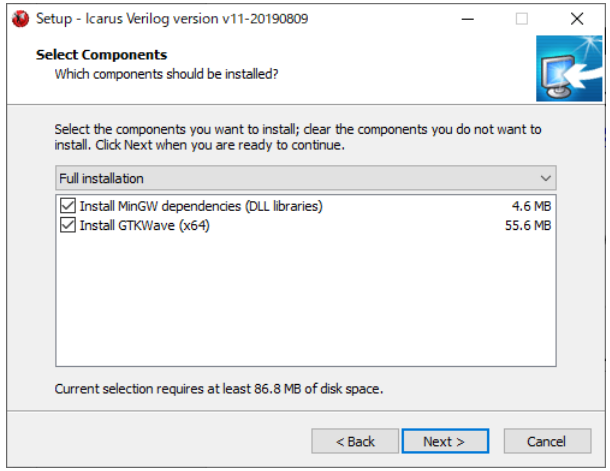
ツールのインストール

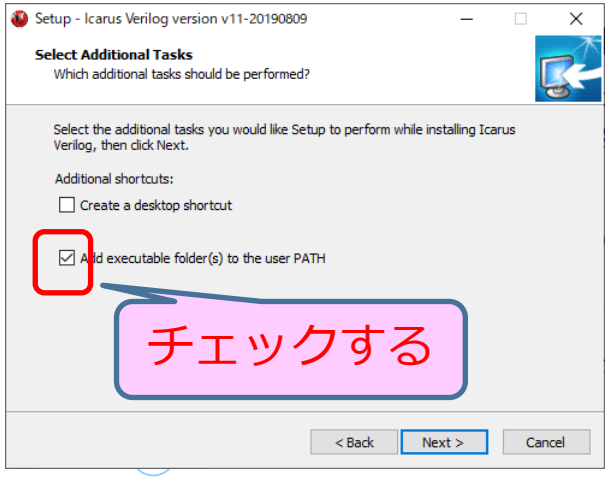
- iverilog と gtkwave
  - Ubuntu:「sudo apt install iverilog gtkwave」を実行
  - Windows: http://bleyer.org/icarus/
    - 上記URLから以下をダウンロードしてインストール
      - iverilog-v11-20201123-x64_setup.exe
    - インストール時に以下をチェックする
      - 「Add executable folder(s) to the user PATH」

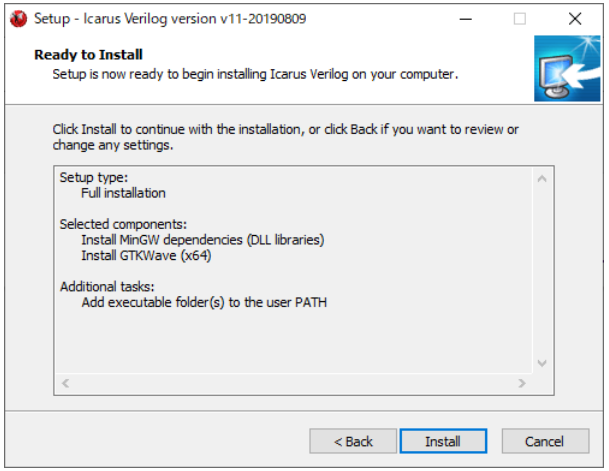
Windows版インストール画面

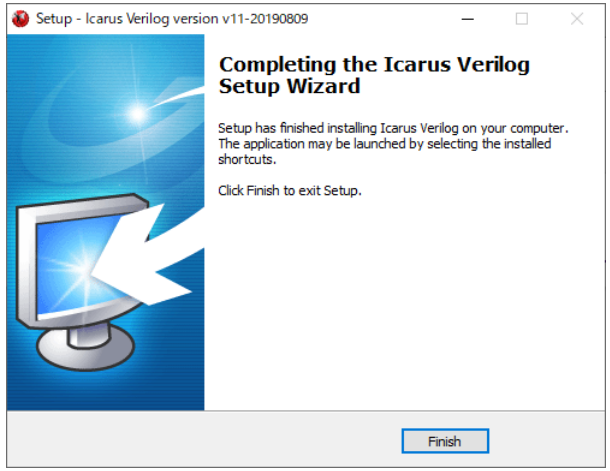












Windows版ActivePerlのインストール

- rom.plからrom.vを出力するときに必要
- 以下のサイトからインストール
  - https://www.activestate.com/products/perl/

Windows版ghostscriptのインストール

- gtkwaveからPDFなどを出力するときに必要
- 以下をダウンロードしてインストール
  - http://core.ring.gr.jp/pub/text/TeX/ptex-win32/gs/gs952w64full-gpl.exe
- PATH環境変数に以下の2つを登録
  - C:¥Program Files¥gs¥gs9.52¥bin
  - C:¥Program Files¥gs¥gs9.52¥lib

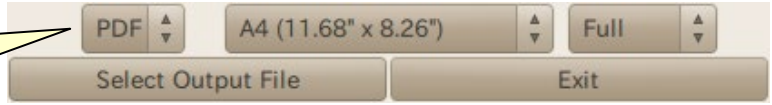
回路シミュレーションと波形の表示

- iverilogによるコンパイル
  - rom.v を作成後に端末上で「iverilog *.v」を実行
- シミュレーションの実行
  - コンパイル実行後に「vvp a.out」を実行
  - Linuxでは 「./a.out」でも実行可能
- gtkwaveによる波形の表示
  - シミュレーション実行後に「gtkwave mips.vcd mips.gtkw」を実行

画像ファイルの作成

File → Print To File

用紙サイズはA4を指定



画像形式を指定
PDFやPSなど

指定が終わったらSelect Output Fileをクリック

ファイル名を指定

ファイル名を指定したら
保存をクリック

第２週の課題

- パイプライン処理の動作確認
  - pipeline2.asm: 実行時の各ステージの動作を確認
- データハザードとフォワーディングの動作確認
  - hazard2.asm: ①ハザード検出なし・フォワーディングなし
    ②ハザード検出のみあり、③両方あり、各動作を確認
- 制御ハザードの動作確認
  - branch2.asm: ①遅延分岐なし、②遅延分岐あり
    各動作を確認、②はコードも修正
- ソートプログラムの修正
  - 実行クロック数を可能な限り短縮
    ハザードを最小限に抑えて、遅延分岐を最大限に活用

■ sim.v

`define STEP: 命令の実行結果

`define SORT: ソートの結果

見るときは定義、見ないときはコメントアウト

en_hzd : ハザード検出を行うとき 1 に設定

en_fwd: フォワーディングを行うとき 1 に設定

en_dly: 遅延スロットを使うとき 1 に設定

```
`timescale 1ns/1ns
`define STEP /* 命令の実行結果を見るとき */
//`define SORT /* ソートの結果を見るとき */
module sim();
        wire en_hzd = 1'b0; /* ハザード検出 */
        wire en_fwd = 1'b0; /* フォワーディング */
        wire en_dly = 1'b0; /* 遅延分岐 */
```

```
0x00400000   0x20100005   addi $16,$0, 5
```

pc	inst	ra	rb	rd	alu_a	alu_b	alu_q	dst	reg_din
00400000									
	20100005	0	0	16					
					00000000	00000005			
							00000005		
								16	00000005

```
0x00400010   0x02114020   add   $8, $16,$17
```

pc	inst	ra	rb	rd	alu_a	alu_b	alu_q	dst	reg_din
00400010									
	02114020	16	17	8					
					00000005	00000003			
							00000008		
								8	00000008

```
0x0040001c   0x8e130000   lw $19,0x00000000($16)
0x00400020   0x02739820   add $19,$19,$19
```

pc	inst	d	f	ra	rb	rd	alu_a	alu_q	dst	reg_din
0040001c										
00400020	8e130000	0	0	16	0	19				
00400024	02739820	1	0	19	19	19	10010000			
00400024	02739820	0	1	19	19	19	00000000	10010000		
00400028							00000003	00000000	19	00000003
								00000006	0	00000000
									19	00000006

forward

命令形式

算術演算命令
ロード／ストア命令

形式	命令	op	rs	rt	rd	shamt	funct
R	フィールド長	6	5	5	5	5	6
	add	0	入力1	入力2	出力	0	32
	addu	0	入力1	入力2	出力	0	33
	sub	0	入力1	入力2	出力	0	34
	subu	0	入力1	入力2	出力	0	35
I	addi	8	入力	出力	即値（符号付き）		
	addiu	9	入力	出力	即値（符号なし）		
	lw	35	入力	出力	アドレス		
	sw	43	入力	出力	アドレス		
	フィールド長	6	5	5	16		
形式	命令	op	rs	rt	address/immediate		

命令形式

論理演算命令

形式	命令	op	rs	rt	rd	shamt	funct
R	フィールド長	6	5	5	5	5	6
	and	0	入力1	入力2	出力	0	36
	or	0	入力1	入力2	出力	0	37
	xor	0	入力1	入力2	出力	0	38
	nor	0	入力1	入力2	出力	0	39
I	andi	12	入力	出力	即値（符号なし）		
	ori	13	入力	出力	即値（符号なし）		
	xori	14	入力	出力	即値（符号なし）		
	lui	15	0	出力	即値（符号なし）		
	フィールド長	6	5	5	16		
形式	命令	op	rs	rt	address/immediate		

命令形式

シフト演算命令
比較演算命令

形式	命令	op	rs	rt	rd	shamt	funct
R	フィールド長	6	5	5	5	5	6
	sll	0	0	入力	出力	シフト量	0
	srl	0	0	入力	出力	シフト量	2
	sra	0	0	入力	出力	シフト量	3

形式	命令	op	rs	rt	rd	shamt	funct
R	フィールド長	6	5	5	5	5	6
	slt	0	入力1	入力2	出力	0	42
	sltu	0	入力1	入力2	出力	0	43
I	slti	10	入力	出力	即値（符号付き）		
	sltiu	11	入力	出力	即値（符号なし）		
	フィールド長	6	5	5	16		
形式	命令	op	rs	rt	address/immediate		

命令形式

条件分岐／ジャンプ命令
コール／リターン命令

形式	命令	op	rs	rt	address/immediate
I	フィールド長	6	5	5	16
	beq	4	入力1	入力2	分岐先
	bne	5	入力1	入力2	分岐先
J	j	2	ジャンプ先		
	jal	3	ジャンプ先		
	フィールド長	6	26		
形式	命令	op	target address		

形式	命令	op	rs	rt	rd	shamt	funct
R	フィールド長	6	5	5	5	5	6
	jr	0	入力	0	0	0	8
J	jal	3	ジャンプ先				
	フィールド長	6	26				
形式	命令	op	target address				

命令一覧(形式別)

形式	命令	op	rs	rt	rd	shamt	funct
R	フィールド長	6	5	5	5	5	6
	sll	0	0	入力	出力	シフト量	0
	srl	0	0	入力	出力	シフト量	2
	sra	0	0	入力	出力	シフト量	3
	jr	0	入力	0	0	0	8
	add	0	入力1	入力2	出力	0	32
	addu	0	入力1	入力2	出力	0	33
	sub	0	入力1	入力2	出力	0	34
	subu	0	入力1	入力2	出力	0	35
	and	0	入力1	入力2	出力	0	36
	or	0	入力1	入力2	出力	0	37
	xor	0	入力1	入力2	出力	0	38
	nor	0	入力1	入力2	出力	0	39
	slt	0	入力1	入力2	出力	0	42
	sltu	0	入力1	入力2	出力	0	43

CSL
Computer Systems Laboratory

形式	命令	op	rs	rt	address/immediate
I	フィールド長	6	5	5	16
	beq	4	入力1	入力2	分岐先
	bne	5	入力1	入力2	分岐先
	addi	8	入力	出力	即値（符号付き）
	addiu	9	入力	出力	即値（符号なし）
	slti	10	入力	出力	即値（符号付き）
	sltiu	11	入力	出力	即値（符号なし）
	andi	12	入力	出力	即値（符号なし）
	ori	13	入力	出力	即値（符号なし）
	xori	14	入力	出力	即値（符号なし）
	lui	15	0	出力	即値（符号なし）
	lw	35	入力	出力	アドレス
	sw	43	入力	出力	アドレス

形式	命令	op	target address
J	フィールド長	6	26
	j	2	ジャンプ先
	jal	3	ジャンプ先