情報工学実験 IV
第 7 回： 制御ハザードと遅延分岐

柴田裕一郎・松尾堅太郎・元島晃伸

shibata@cis.nagasaki-u.ac.jp
情報工学コース

2019 年 1 月 29 日

パイプラインハザード

パイプライン → 理想的には 1 命令／ 1 クロックだが…

- 構造ハザード（structural hazard）
  異なるステージで同一の資源（たとえば演算器など）を使おうとして起こる.
- データハザード（data hazard）
  ある命令が計算する結果を後続の命令が使おうとしたときに，前の命令がまだパイプラインの中にあってまだ必要なデータが計算できていないために起こる.
- 制御ハザード（control hazard）
  分岐命令がまだパイプラインの中にあり PC（プログラムカウンタ）が飛び先番地に設定される前に，次々と後続の命令を読んでしまうために起こる.

コントロールハザードを起こすプログラムの例

先頭で無限ループするプログラム．2 番地以降は実行されないはず．

```
 0 番地:   JMP   #-2       (11000111 11111110)
 2 番地:   LLI   r1, #1    (00001001 00000001)
 4 番地:   LLI   r2, #2    (00001010 00000010)
 6 番地:   LLI   r3, #3    (00001011 00000011)
 8 番地:   LLI   r4, #4    (00001100 00000100)
10 番地:   LLI   r5, #5    (00001101 00000101)
```

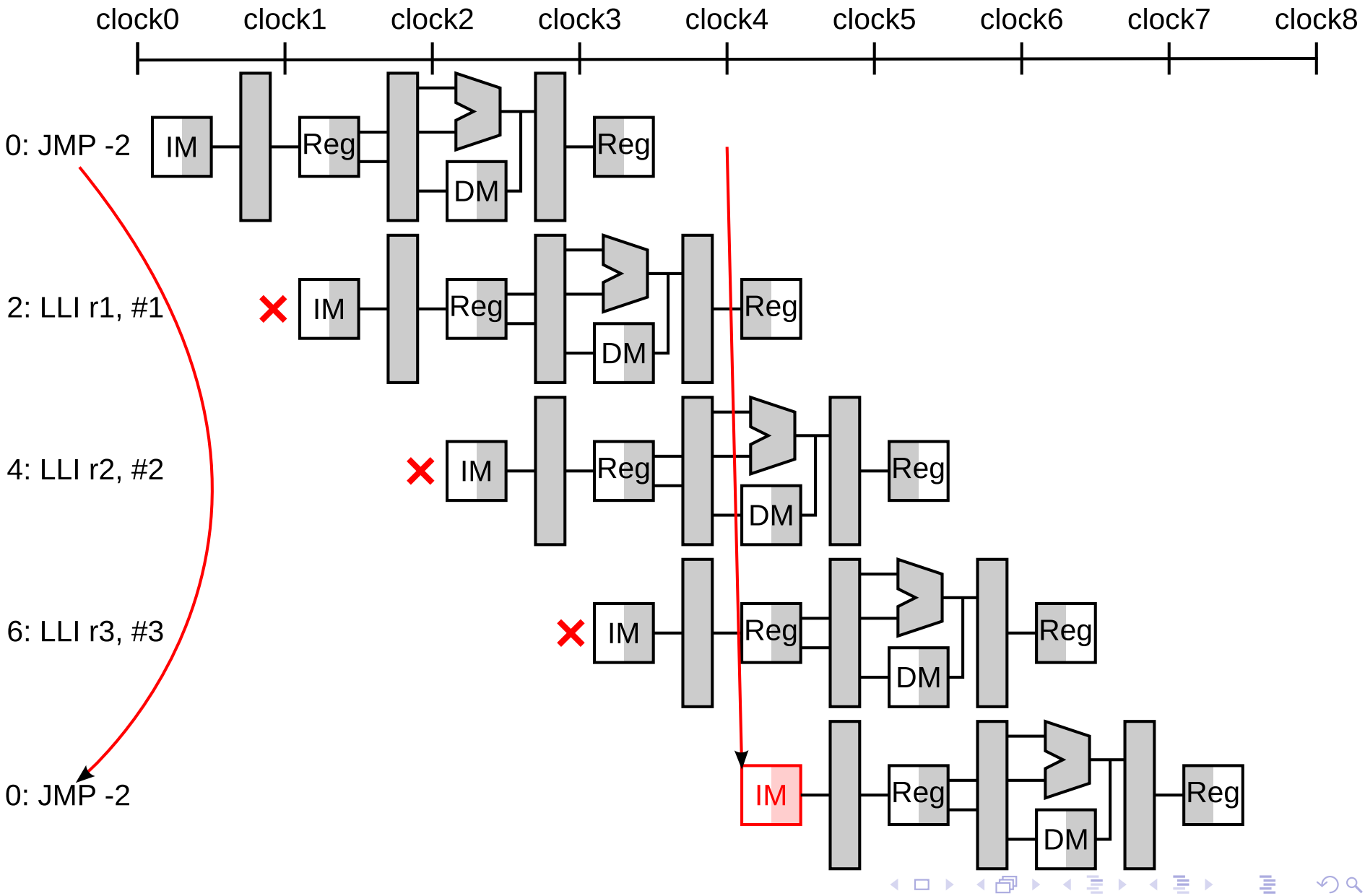
実行結果

```
==== clock: 0 ====
 if_pc:0000 if_ir:0000000000000000
 rf_pc:0000 rf_ir:0000000000000000 rf_treg1:0000 rf_treg2:0000 rf_immediate:0000
 ex_ir:0000000000000000 ex_result:0000
 daddr:0000 ddin:xxxx ddout:xxxx doe:0 dwe:0 iaddr:0000 idin:c7fe ioe:1
 alu_ain:0000 alu_bin:0000 alu_op:0000 reg_file_we:0 if_pc_we:0 led:000000
 regs: 0000 0000 0000 0000 0000 0000 0000 0000

==== clock: 1 ====
 if_pc:0002 if_ir:1100011111111110
 rf_pc:0000 rf_ir:0000000000000000 rf_treg1:0000 rf_treg2:0000 rf_immediate:0000
 ex_ir:0000000000000000 ex_result:0000
 daddr:0000 ddin:xxxx ddout:xxxx doe:0 dwe:0 iaddr:0002 idin:0901 ioe:1
 alu_ain:0000 alu_bin:0000 alu_op:0000 reg_file_we:0 if_pc_we:0 led:000000
 regs: 0000 0000 0000 0000 0000 0000 0000 0000

==== clock: 2 ====
 if_pc:0004 if_ir:0000100100000001
 rf_pc:0002 rf_ir:1100011111111110 rf_treg1:0000 rf_treg2:0000 rf_immediate:fffe
 ex_ir:0000000000000000 ex_result:0000
 daddr:0000 ddin:xxxx ddout:xxxx doe:0 dwe:0 iaddr:0004 idin:0a02 ioe:1
 alu_ain:0002 alu_bin:fffe alu_op:0100 reg_file_we:0 if_pc_we:0 led:000000
 regs: 0000 0000 0000 0000 0000 0000 0000 0000

==== clock: 3 ====
 if_pc:0006 if_ir:0000101000000010
 rf_pc:0004 rf_ir:0000100100000001 rf_treg1:0000 rf_treg2:0000 rf_immediate:0001
 ex_ir:1100011111111110 ex_result:0000
 daddr:0000 ddin:xxxx ddout:xxxx doe:0 dwe:0 iaddr:0006 idin:0b03 ioe:1
 alu_ain:0000 alu_bin:0001 alu_op:0001 reg_file_we:0 if_pc_we:1 led:000000
 regs: 0000 0000 0000 0000 0000 0000 0000 0000
```

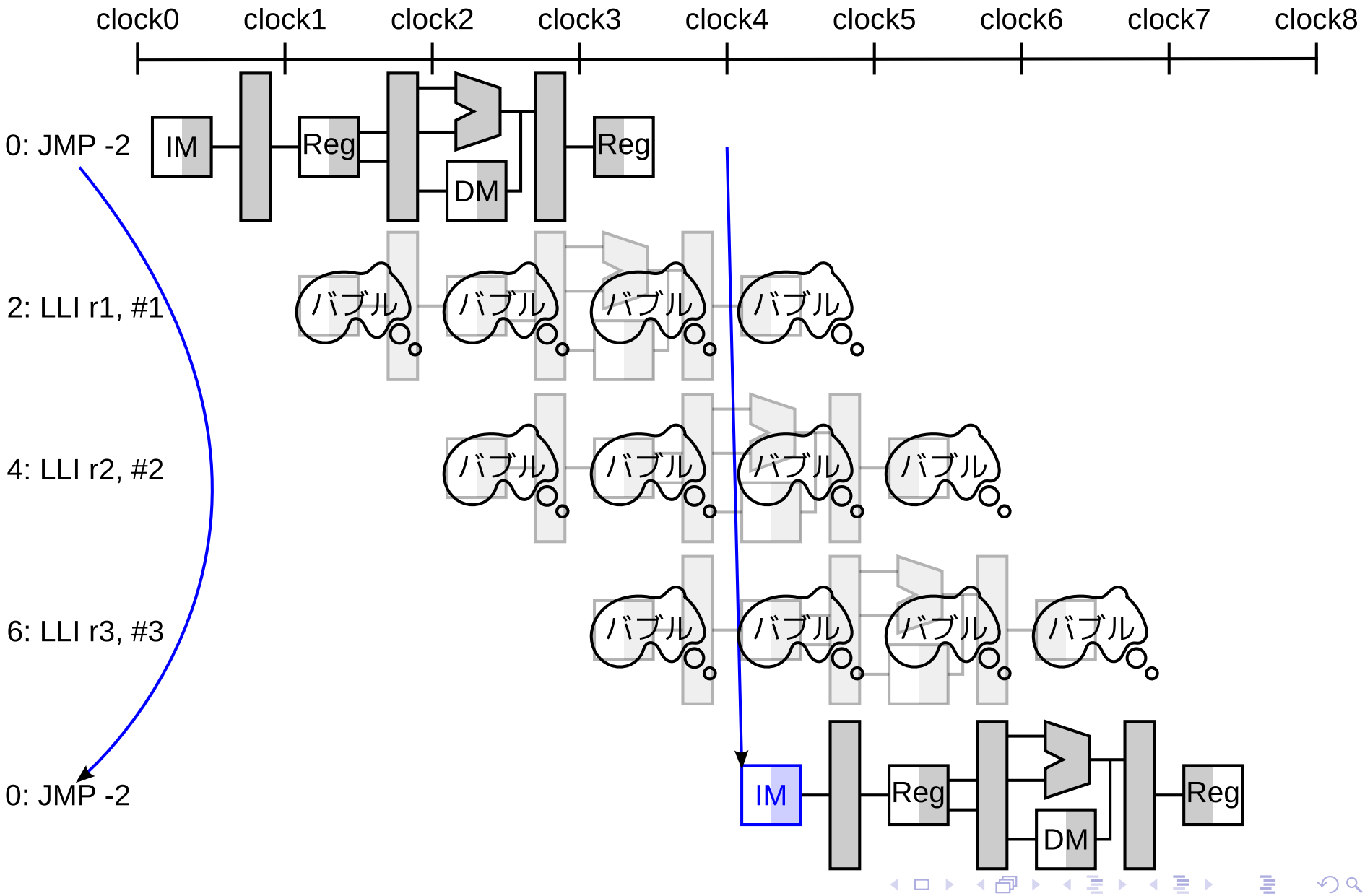
実行結果

```
==== clock: 4 ====
 if_pc:0000 if_ir:0000101100000011
 rf_pc:0006 rf_ir:0000101000000010 rf_treg1:0000 rf_treg2:0000 rf_immediate:0002
 ex_ir:0000100100000001 ex_result:0001
 daddr:0000 ddin:xxxx ddout:xxxx doe:0 dwe:0 iaddr:0000 idin:c7fe ioe:1
 alu_ain:0000 alu_bin:0002 alu_op:0001 reg_file_we:1 if_pc_we:0 led:000000
 regs: 0000 0000 0000 0000 0000 0000 0000 0000

==== clock: 5 ====
 if_pc:0002 if_ir:1100011111111110
 rf_pc:0000 rf_ir:0000101100000011 rf_treg1:0000 rf_treg2:0000 rf_immediate:0003
 ex_ir:0000101000000010 ex_result:0002
 daddr:0000 ddin:xxxx ddout:xxxx doe:0 dwe:0 iaddr:0002 idin:0901 ioe:1
 alu_ain:0000 alu_bin:0003 alu_op:0001 reg_file_we:1 if_pc_we:0 led:000000
 regs: 0000 0001 0000 0000 0000 0000 0000 0000

==== clock: 6 ====
 if_pc:0004 if_ir:0000100100000001
 rf_pc:0002 rf_ir:1100011111111110 rf_treg1:0000 rf_treg2:0000 rf_immediate:fffe
 ex_ir:0000101100000011 ex_result:0003
 daddr:0000 ddin:xxxx ddout:xxxx doe:0 dwe:0 iaddr:0004 idin:0a02 ioe:1
 alu_ain:0002 alu_bin:fffe alu_op:0100 reg_file_we:1 if_pc_we:0 led:000000
 regs: 0000 0001 0002 0000 0000 0000 0000 0000

==== clock: 7 ====
 if_pc:0006 if_ir:0000101000000010
 rf_pc:0004 rf_ir:0000100100000001 rf_treg1:0001 rf_treg2:0000 rf_immediate:0001
 ex_ir:1100011111111110 ex_result:0000
 daddr:0000 ddin:xxxx ddout:xxxx doe:0 dwe:0 iaddr:0006 idin:0b03 ioe:1
 alu_ain:0001 alu_bin:0001 alu_op:0001 reg_file_we:0 if_pc_we:1 led:000000
 regs: 0000 0001 0002 0003 0000 0000 0000 0000
```

コントロールハザード発生時のパイプライン



NOP を挿入して制御ハザードを回避



NOP 挿入法の問題点

- 最悪の場合，4 クロックに 1 個の命令しか処理できない
- 分岐命令が連続することはそれほどないが，それでも 1 命令に 4 クロックではパイプライン化の意味がない

⇓

もう少し分岐命令のペナルティを減らしたい

⇓

より早いステージで分岐命令を検出し対処する

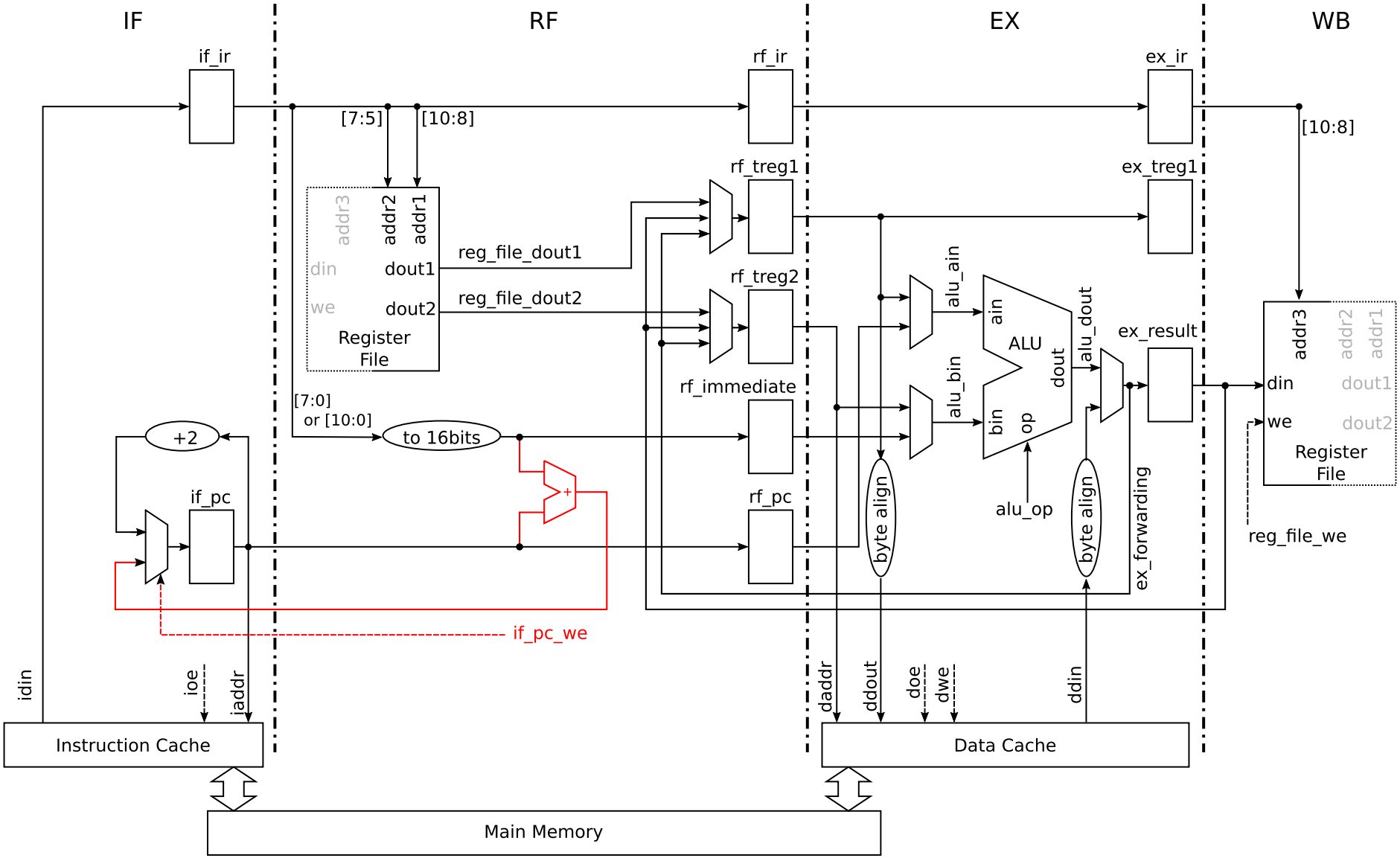
分岐ペナルティの削減

フェッチした命令が分岐命令かどうかは，RF ステージで調べることができる．

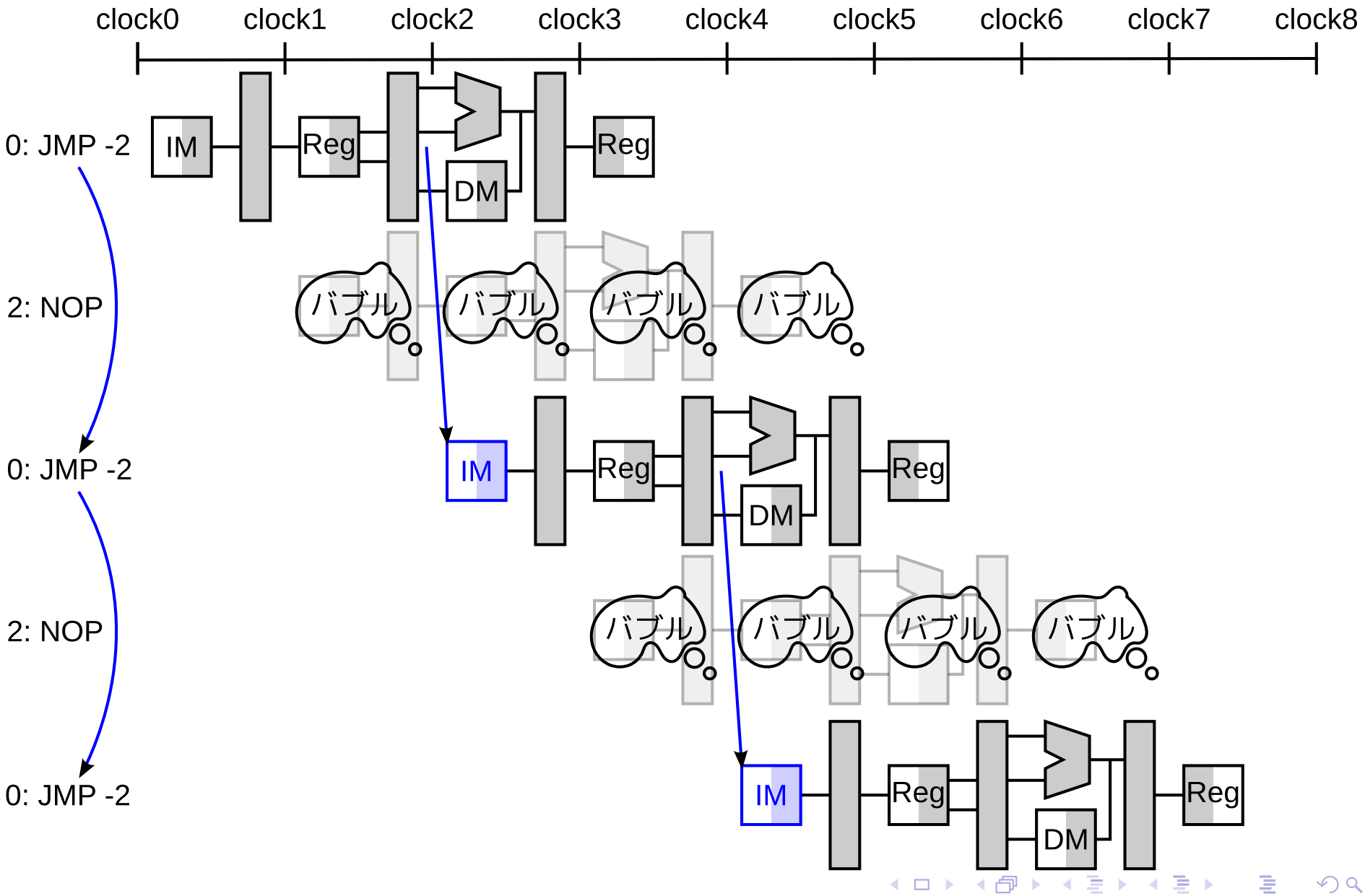
1. RF ステージでフェッチした命令が分岐命令かどうか，また，分岐の条件を満たしているかを判断
2. これと並列に，飛び先のアドレスを計算

- 飛び先アドレスの計算のために，専用の加算器が必要になる
- 分岐条件の判定用レジスタの値が，他のステージからフォワードされてくる可能性もあるので注意

分岐ペナルティを削減した RISC16b の構成



分岐ペナルティを削減したパイプラインの動作



遅延分岐

- 依然として分岐命令の次には 1 個の NOP 命令を挿入する必要がある
- しかし，分岐命令のペナルティをこれ以上減らすのは難しい

⇓

発想の転換

⇓

プロセッサの分岐は 1 命令遅れて起きるものと考える

遅延分岐（delayed branch）

遅延スロット（ディレイスロット）

- 遅延分岐の後にあるのに分岐の前に実行される命令をディレイスロットと呼ぶ
- ディレイスロットに入れられた命令は，分岐が飛ぶ場合も飛ばない場合にも必ず実行される
- 改良前の RISC16p のディレイスロット数は 3
- 改良後の RISC16p のディレイスロット数は 1
- うまく命令をスケジューリング（並び替える）ことによって，NOP 以外の有効な命令をディレイスロットに割り当てれば無駄にならない