Department of Computer Science
Course number: CSC.T341

コンピュータ論理設計
Computer Logic Design

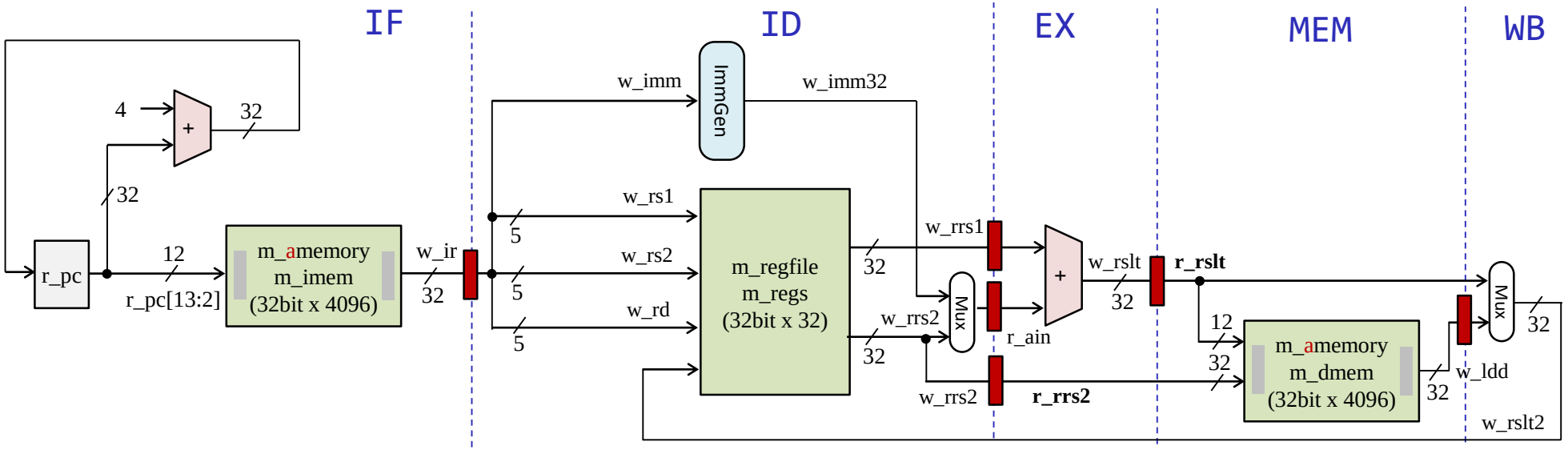
11. パイプラインプロセッサとハザード処理 (1)
Pipelining Processor and Hazards (1)

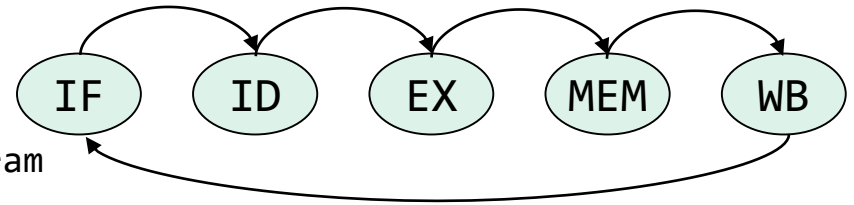
吉瀬 謙二　情報工学系
Kenji Kise, Department of Computer Science
kise _at_ c.titech.ac.jp　www.arch.cs.titech.ac.jp/lecture/CLD/
講義：月曜日 10:45-12:25, 木曜日 10:45-12:25

Ver. 2023-05-18b

m_proc09 (multi-cycle processor) マルチサイクル

- 5サイクルで1命令を処理する IPC = 0.2 のマルチサイクルのプロセッサ, add, addi, lw, sw をサポート
- 最も遅延の長い(長い時間を要する)ステップがプロセッサの動作周波数を決める.

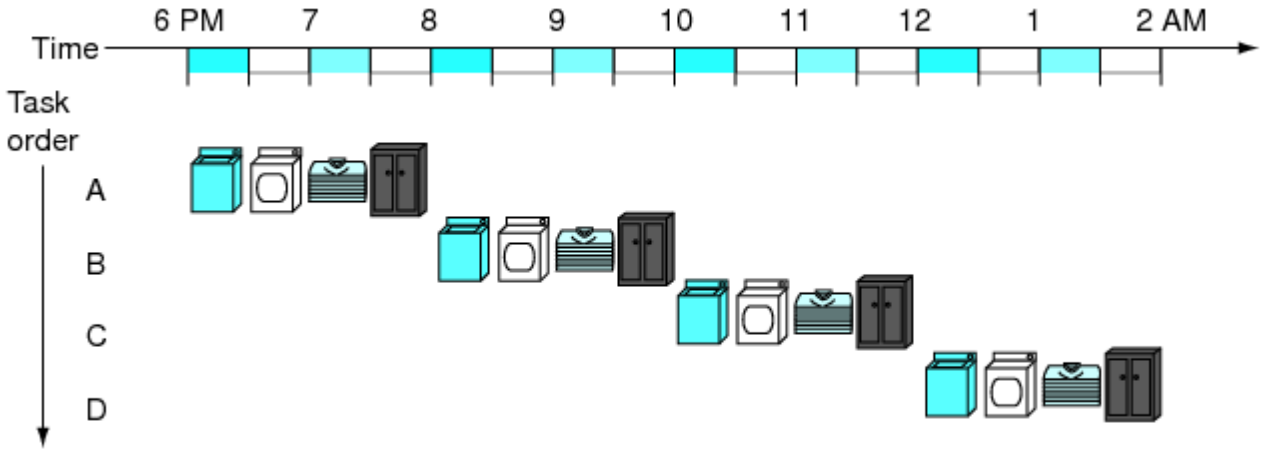




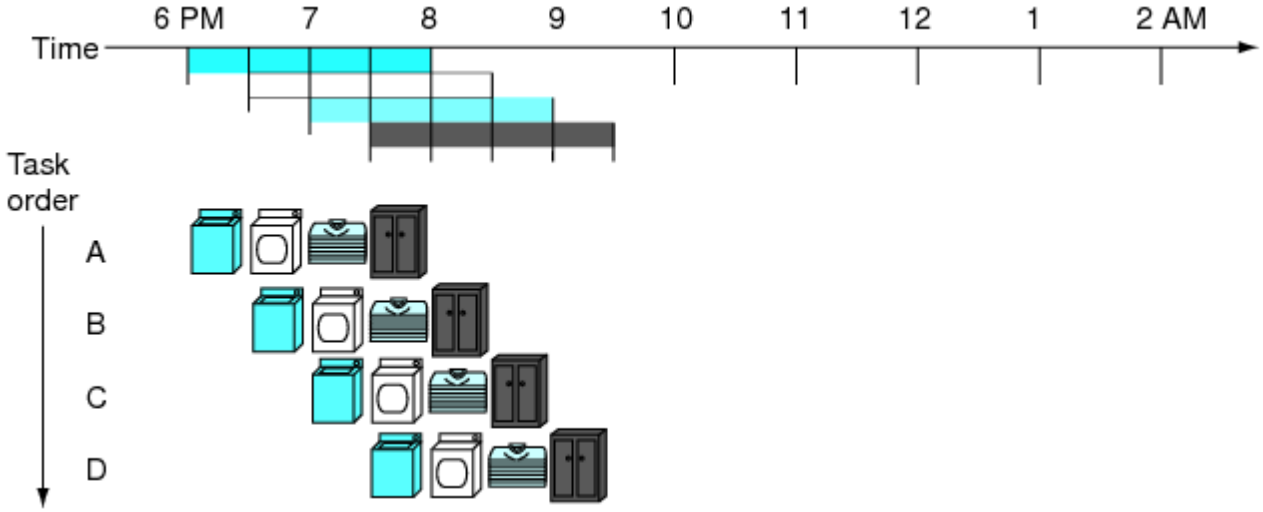
Simple State Machine Diagram

An Overview of Pipelining

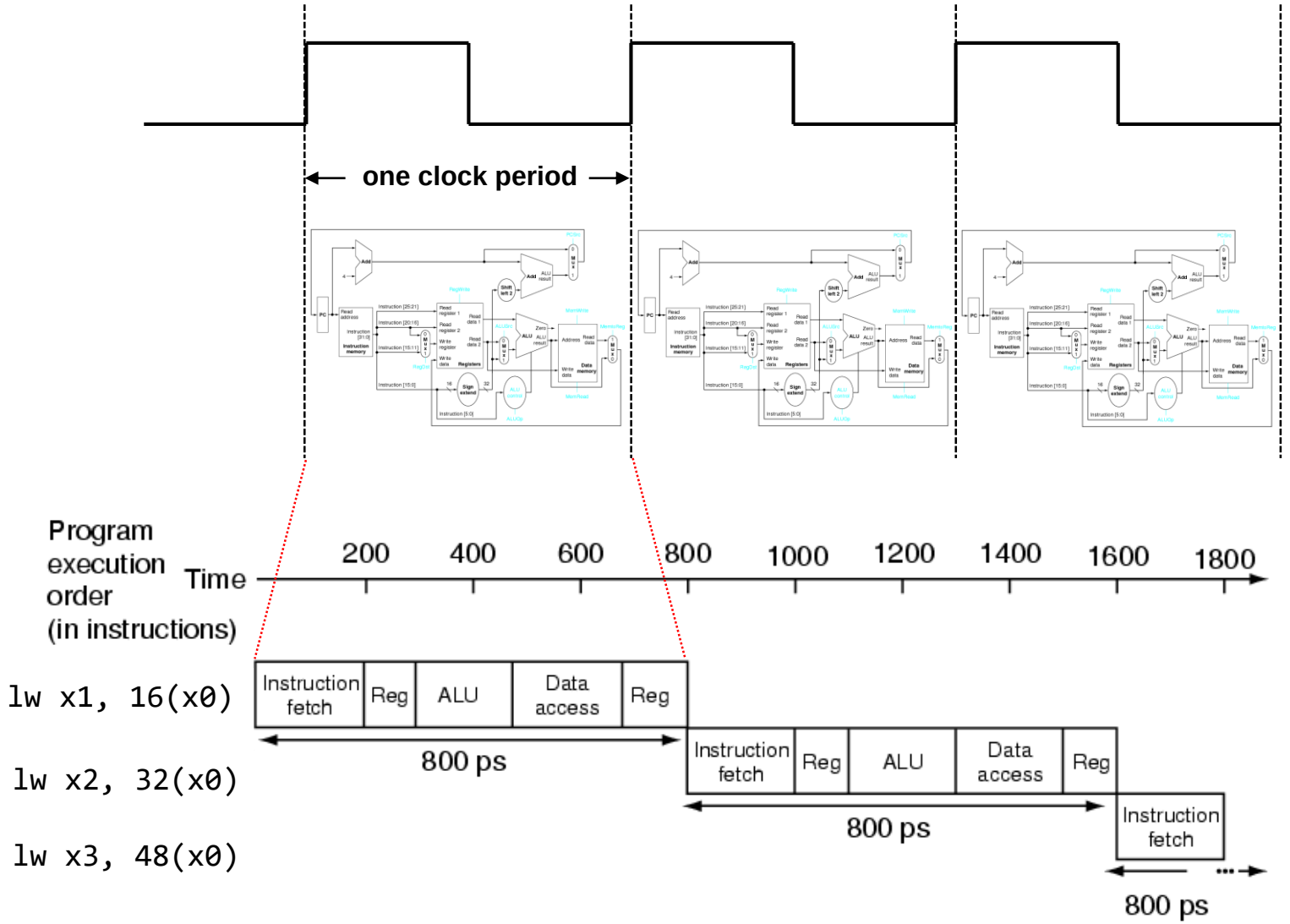
- Non pipelining (Multi-cycle)



- Pipelining

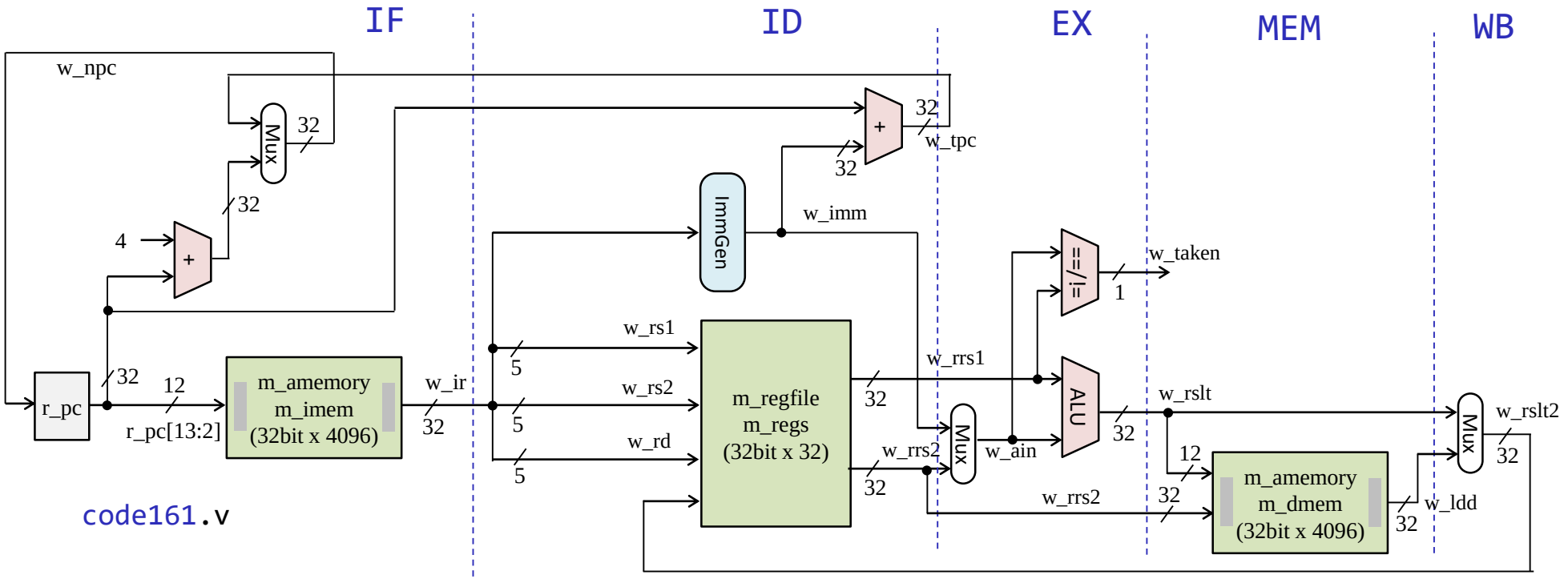


Single Cycle Processor



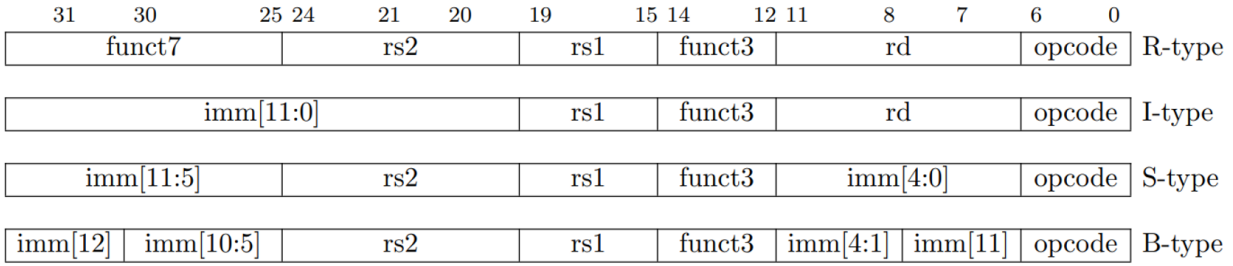
m_proc07 ベースラインのプロセッサ（シングルサイクル）

- 命令フェッチ(IF), デコードとオペランドフェッチ(ID), 実行(EX), メモリアクセス(MEM), ライトバック(WB) の処理をおこなうadd, addi, sll, srl, lw, sw, beq, bne命令に対応したプロセッサ

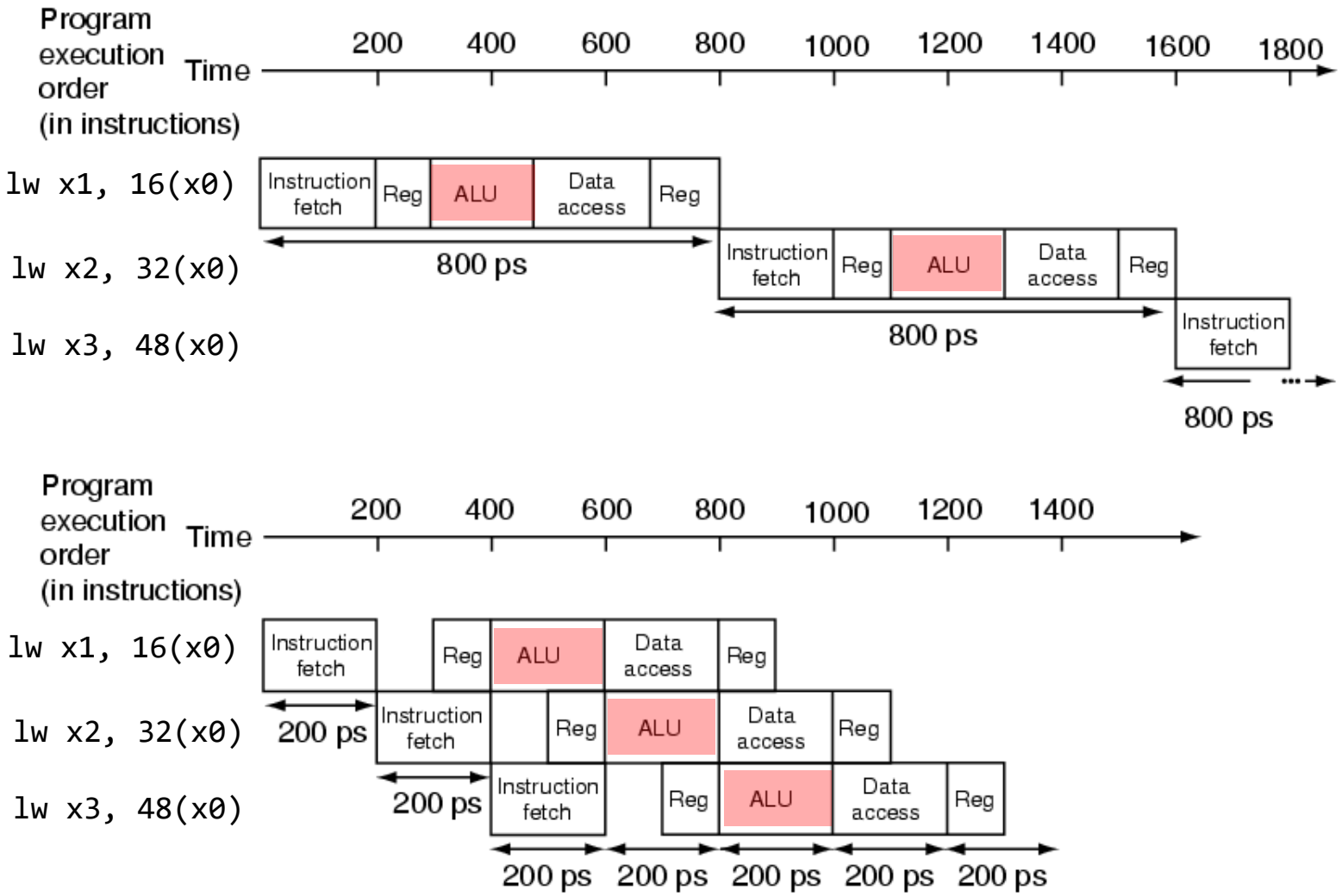


code161.v

```
f    = 60MHz
IPC  = 1.000
Perf = 60.00
```



Single-cycle versus pipelined execution



Hazards make pipelining hard

- 命令を適切なサイクルで実行できないような状況が存在する．これをハザード(hazard)と呼ぶ．
  - 構造ハザード (structural hazard)
    - オーバラップ実行する命令の組み合わせをハードウェアがサポートしていない場合．資源不足により生じる．
  - データ・ハザード(data hazard)
    - データの受け渡しの制約によって生じるハザード

```
cm_ram[0]={7'd0, 5'd0, 5'd0, 3'b000, 5'd0, 7'b0110011}; //    add  x0, x0, x0  // NOP
cm_ram[1]={12'd3,      5'd0, 3'b000, 5'd4, 7'b0010011}; //    addi x4, x0, 3   // x4 = 3
cm_ram[2]={12'd4,      5'd0, 3'b000, 5'd5, 7'b0010011}; //    addi x5, x0, 4   // x5 = 4
cm_ram[3]={7'd0, 5'd5, 5'd4, 3'b000, 5'd30,7'b0110011}; //    add  x30,x4, x5  // led = x4 + x5
```

  - 制御ハザード(control hazard) 今回は，この対処方法を考える．
    - 分岐命令，ジャンプ命令によって生じるハザード

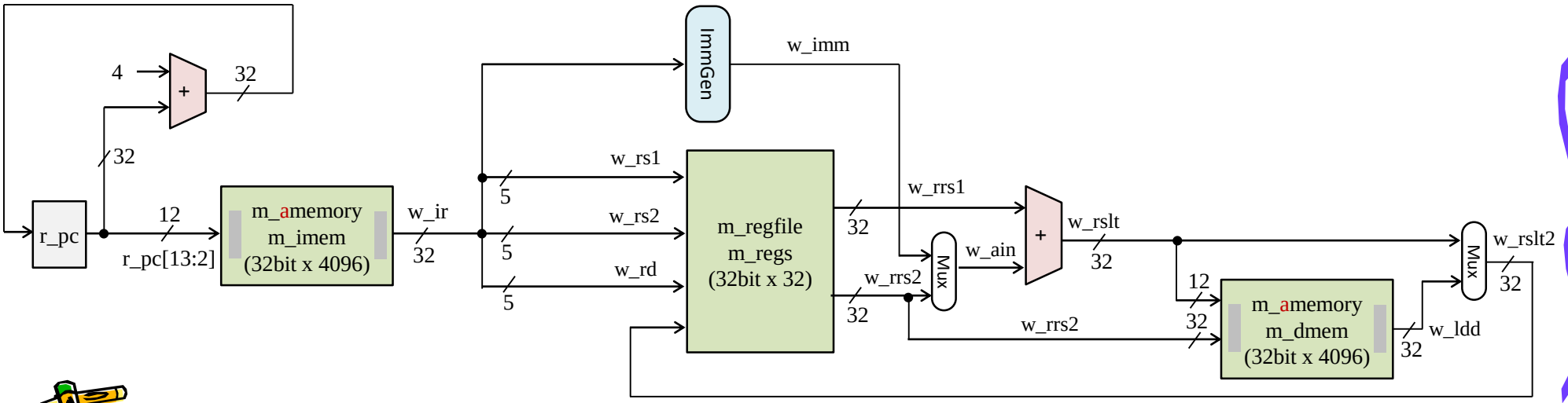
```
cm_ram[1]={12'd5,      5'd0, 3'b000, 5'd4, 7'b0010011};     //    addi x4, x0, 5   // x4 = 5
cm_ram[2]={12'd1,      5'd0, 3'b000, 5'd5, 7'b0010011};     //    addi x5, x0, 1   // x5 = 1
cm_ram[3]={12'd1,      5'd5, 3'b000, 5'd5, 7'b0010011};     // L1:addi x5, x5, 1   // x5 = x5 + 1
cm_ram[4]={7'h7f,5'd4, 5'd5, 3'b001, 5'b11101, 7'b1100011}; //    bne  x5, x4, L1  // goto L1 if x5!=x4
cm_ram[5]={7'd0, 5'd0, 5'd5, 3'b000, 5'd30,7'b0110011};     //    add  x30,x5, x0  // led = x5
```

m_proc05 add, addi, lw, sw を処理するシングルサイクル版

```
module m_proc05 (w_clk, w_ce, w_led);
  input  wire w_clk, w_ce;
  output wire [31:0] w_led;

  reg [31:0] r_pc = 0;
  wire [31:0] w_ir;
  wire [4:0]  w_op5 = w_ir[6:2];
  wire [4:0]  w_rs1 = w_ir[19:15];
  wire [4:0]  w_rs2 = w_ir[24:20];
  wire [4:0]  w_rd  = w_ir[11:7];
```

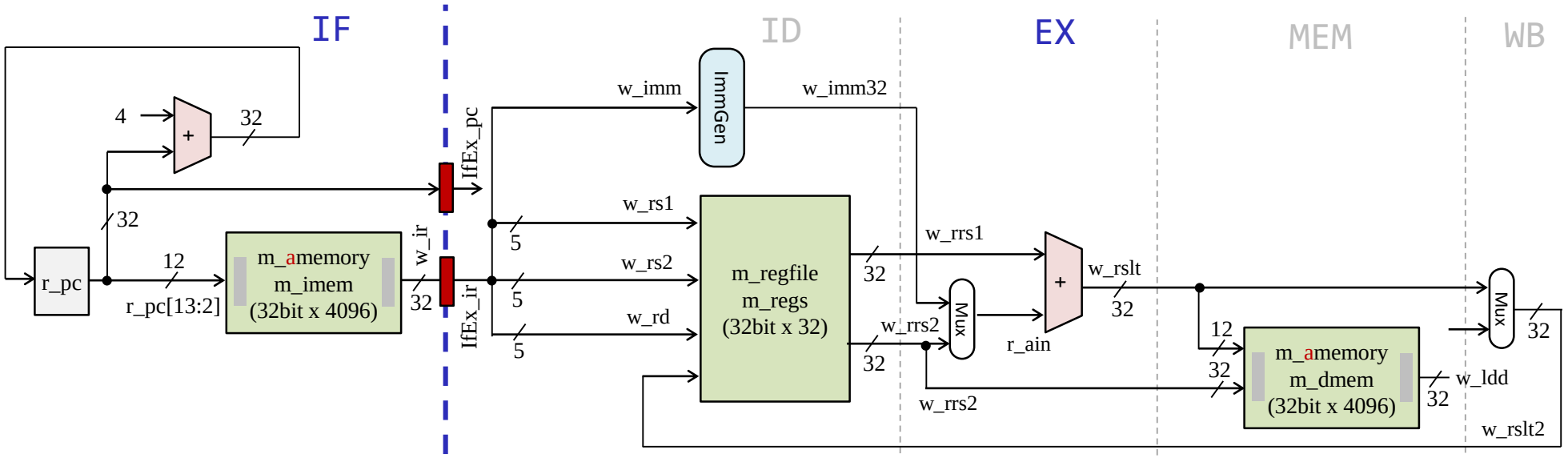
code171.v

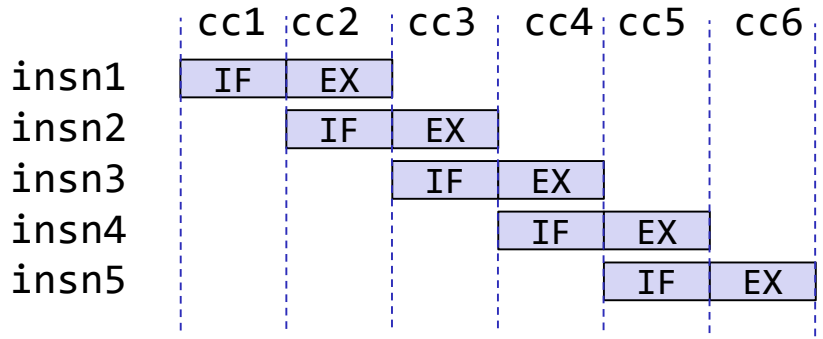
```
  wire #4 w_we = w_ce & (w_op5==5'b01100 || w_op5==5'b00100 || w_op5==5'b00000);
  wire [31:0] w_imm, w_rrs1, w_rrs2, w_ain, w_rslt, w_ldd, w_rslt2;

  m_amemory m_imem (w_clk, r_pc[13:2], 1'd0, 32'd0, w_ir);
  m_immgen m_immgen0 (w_ir, w_imm);
  m_regfile m_regs (w_clk, w_rs1, w_rs2, w_rd, w_we, w_rslt2, w_rrs1, w_rrs2);
  assign #3 w_ain = (w_op5==5'b01100) ? w_rrs2 : w_imm;
  assign #9 w_rslt = w_rrs1 + w_ain;
  m_amemory m_dmem (w_clk, w_rslt[13:2], (w_op5==5'b01000), w_rrs2, w_ldd);
  assign #3 w_rslt2 = (w_op5==5'b00000) ? w_ldd : w_rslt;
  always @(posedge w_clk) #5 if(w_ce & r_pc!=24) r_pc <= r_pc + 4;

  reg [31:0] r_led = 0;
  always @(posedge w_clk) if(w_ce & w_we & w_rd==30) r_led <= w_rslt2;
  assign w_led = r_led;
endmodule
```



m_proc11 2ステージのパイプラインプロセッサ

- IF を1ステージ，その他の ID, EX, MEM, WB を1ステージとするパイプラインプロセッサ
- add, addi, lw, sw を処理するプロセッサ



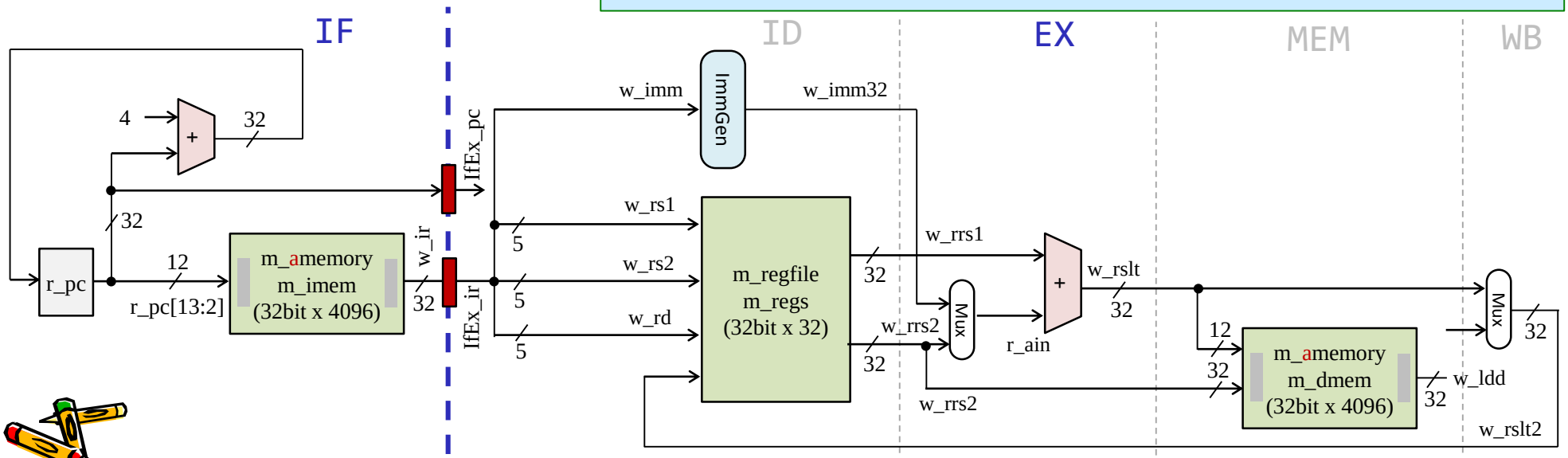


m_proc11 add, addi, lw, sw を処理する2段のパイプライン版

```
module m_proc11 (w_clk, w_ce, w_led);
  input  wire w_clk, w_ce;
  output wire [31:0] w_led;

  reg [31:0] r_pc = 0;
  reg [31:0] IfEx_pc=0;
  reg [31:0] IfEx_ir=0;

  wire [31:0] w_ir;
  wire [4:0]  w_op5 = IfEx_ir[6:2];
  wire [4:0]  w_rs1 = IfEx_ir[19:15];
  wire [4:0]  w_rs2 = IfEx_ir[24:20];
  wire [4:0]  w_rd  = IfEx_ir[11:7];
```

code181.v

```
  wire [31:0] w_imm, w_rrs1, w_rrs2, w_ain, w_rslt, w_ldd, w_rslt2;
  wire #4 w_we = w_ce & (w_op5==5'b01100 || w_op5==5'b00100 || w_op5==5'b00000);
  always @(posedge w_clk) #5 IfEx_pc <= r_pc;
  m_amemory m_imem (w_clk, r_pc[13:2], 1'd0, 32'd0, w_ir);
  always @(posedge w_clk) #5 IfEx_ir <= w_ir;

  m_immgen m_immgen0 (IfEx_ir, w_imm);
  m_regfile m_regs (w_clk, w_rs1, w_rs2, w_rd, w_we, w_rslt2, w_rrs1, w_rrs2);
  assign #3 w_ain = (w_op5==5'b01100) ? w_rrs2 : w_imm;
  assign #9 w_rslt = w_rrs1 + w_ain;
  m_amemory m_dmem (w_clk, w_rslt[13:2], (w_op5==5'b01000), w_rrs2, w_ldd);
  assign #3 w_rslt2 = (w_op5==5'b00000) ? w_ldd : w_rslt;
  always @(posedge w_clk) #5 if(w_ce && IfEx_ir!=32'h000f0033) r_pc <= r_pc + 4;

  reg [31:0] r_led = 0;
  always @(posedge w_clk) if(w_ce & w_we & w_rd==30) r_led <= w_rslt;
  assign w_led = r_led;
endmodule
```



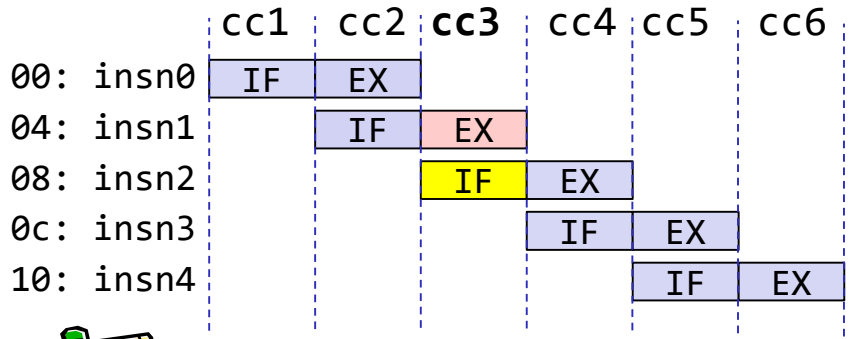
m_proc11 add, addi, lw, sw を処理する2段のパイプライン版

```
module m_proc11 (w_clk, w_ce, w_led);
  input  wire w_clk, w_ce;
  output wire [31:0] w_led;

  reg [31:0] r_pc = 0;
  reg [31:0] IfEx_pc=0;
  reg [31:0] IfEx_ir=0;

  wire [31:0] w_ir;
  wire [4:0]  w_op5 = IfEx_ir[6:2];
  wire [4:0]  w_rs1 = IfEx_ir[19:15];
  wire [4:0]  w_rs2 = IfEx_ir[24:20];
  wire [4:0]  w_rd  = IfEx_ir[11:7];
```

code181.v

```
  wire [31:0] w_imm, w_rrs1, w_rrs2, w_ain, w_rslt, w_ldd, w_rslt2;
  wire #4 w_we = w_ce & (w_op5==5'b01100 || w_op5==5'b00100 || w_op5==5'b00000);
  always @(posedge w_clk) #5 IfEx_pc <= r_pc;
  m_amemory m_imem (w_clk, r_pc[13:2], 1'd0, 32'd0, w_ir);
  always @(posedge w_clk) #5 IfEx_ir <= w_ir;

  m_immgen m_immgen0 (IfEx_ir, w_imm);
  m_regfile m_regs (w_clk, w_rs1, w_rs2, w_rd, w_we, w_rslt2, w_rrs1, w_rrs2);
  assign #3 w_ain = (w_op5==5'b01100) ? w_rrs2 : w_imm;
  assign #9 w_rslt = w_rrs1 + w_ain;
  m_amemory m_dmem (w_clk, w_rslt[13:2], (w_op5==5'b01000), w_rrs2, w_ldd);
  assign #3 w_rslt2 = (w_op5==5'b00000) ? w_ldd : w_rslt;
  always @(posedge w_clk) #5 if(w_ce && IfEx_ir!=32'h000f0033) r_pc <= r_pc + 4;

  reg [31:0] r_led = 0;
  always @(posedge w_clk) if(w_ce & w_we & w_rd==30) r_led <= w_rslt;
  assign w_led = r_led;
endmodule
```

```
cm_ram[1]={12'd3,      5'd0, 3'b000, 5'd4, 7'b0010011}; // 04   addi x4, x0, 3   // x4 = 3
cm_ram[2]={12'd4,      5'd0, 3'b000, 5'd5, 7'b0010011}; // 08   addi x5, x0, 4   // x5 = 4
cm_ram[3]={7'd0, 5'd5, 5'd4, 3'b000, 5'd30,7'b0110011}; // 0c   add  x30,x4, x5  // led = x4 + x5
```




```
clock: r_pc     IfEx_pc  IfEx_ir  w_rrs1   w_ain    r_rslt2  r_led
    1: 00000000 00000000 00000000 00000000 00000000 00000033 00000000
    2: 00000004 00000000 00000033 00000000 00000000 00000000 00000000
    3: 00000008 00000004 00300213 00000000 00000003 00000003 00000000
    4: 0000000c 00000008 00400293 00000000 00000004 00000004 00000000
    5: 00000010 0000000c 00520f33 00000003 00000004 00000007 00000000
    6: 00000014 00000010 00000033 00000000 00000000 00000000 00000007
```

Hazards make pipelining hard

- 命令を適切なサイクルで実行できないような状況が存在する．これをハザード(hazard)と呼ぶ．
  - 構造ハザード (structural hazard)
    - オーバラップ実行する命令の組み合わせをハードウェアがサポートしていない場合．資源不足により生じる．
  - データ・ハザード(data hazard)
    - データの受け渡しの制約によって生じるハザード

```
cm_ram[0]={7'd0, 5'd0, 5'd0, 3'b000, 5'd0, 7'b0110011}; //    add  x0, x0, x0  // NOP
cm_ram[1]={12'd3,      5'd0, 3'b000, 5'd4, 7'b0010011}; //    addi x4, x0, 3   // x4 = 3
cm_ram[2]={12'd4,      5'd0, 3'b000, 5'd5, 7'b0010011}; //    addi x5, x0, 4   // x5 = 4
cm_ram[3]={7'd0, 5'd5, 5'd4, 3'b000, 5'd30,7'b0110011}; //    add  x30,x4, x5  // led = x4 + x5
```

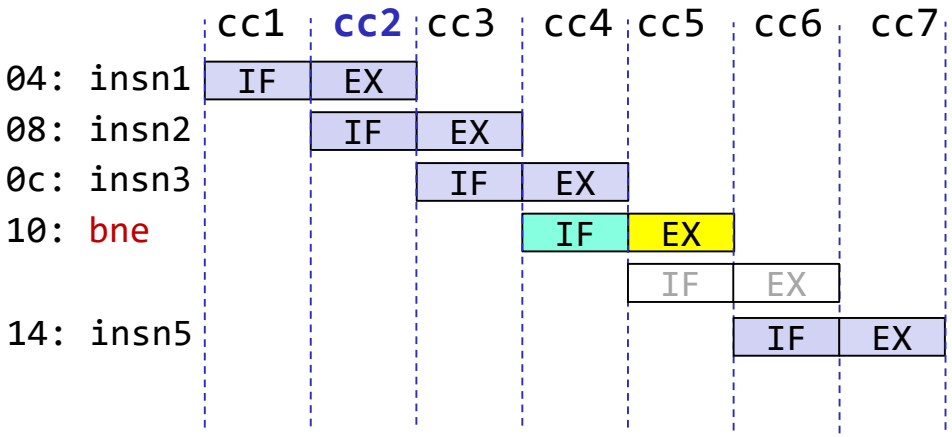
  - 制御ハザード(control hazard)
    - 分岐命令，ジャンプ命令によって生じるハザード

```
cm_ram[1]={12'd5,      5'd0, 3'b000, 5'd4, 7'b0010011};     //    addi x4, x0, 5   // x4 = 5
cm_ram[2]={12'd1,      5'd0, 3'b000, 5'd5, 7'b0010011};     //    addi x5, x0, 1   // x5 = 1
cm_ram[3]={12'd1,      5'd5, 3'b000, 5'd5, 7'b0010011};     // L1:addi x5, x5, 1   // x5 = x5 + 1
cm_ram[4]={7'h7f,5'd4, 5'd5, 3'b001, 5'b11101, 7'b1100011}; //    bne  x5, x4, L1  // goto L1 if x5!=x4
cm_ram[5]={7'd0, 5'd0, 5'd5, 3'b000, 5'd30,7'b0110011};     //    add  x30,x5, x0  // led = x5
```

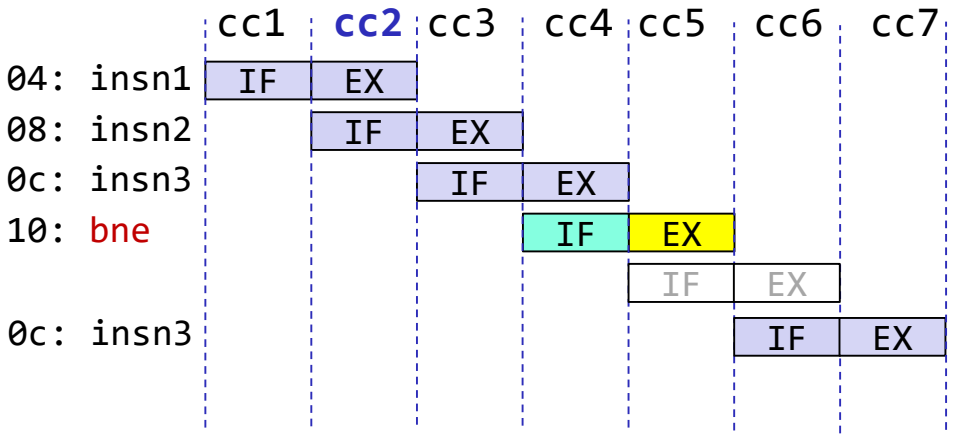
制御ハザードの対処：戦略1

- 分岐方向が判明するまで，分岐命令の後続命令のフェッチを止める（ストールさせる）.
  - このプロセッサ構成では，分岐命令の出現毎に1サイクルの無駄が生じる.

```
cm_ram[1]={12'd5,      5'd0, 3'b000, 5'd4, 7'b0010011};     // 04    addi x4, x0, 5   // x4 = 5
cm_ram[2]={12'd1,      5'd0, 3'b000, 5'd5, 7'b0010011};     // 08    addi x5, x0, 1   // x5 = 1
cm_ram[3]={12'd1,      5'd5, 3'b000, 5'd5, 7'b0010011};     // 0c L1:addi x5, x5, 1   // x5 = x5 + 1
cm_ram[4]={7'h7f,5'd4, 5'd5, 3'b001, 5'b11101, 7'b1100011}; // 10    bne  x5, x4, L1  // goto L1 if x5!=x4
cm_ram[5]={7'd0, 5'd0, 5'd5, 3'b000, 5'd30,7'b0110011};     // 14    add  x30,x5, x0  // led = x5
```



分岐が不成立の場合

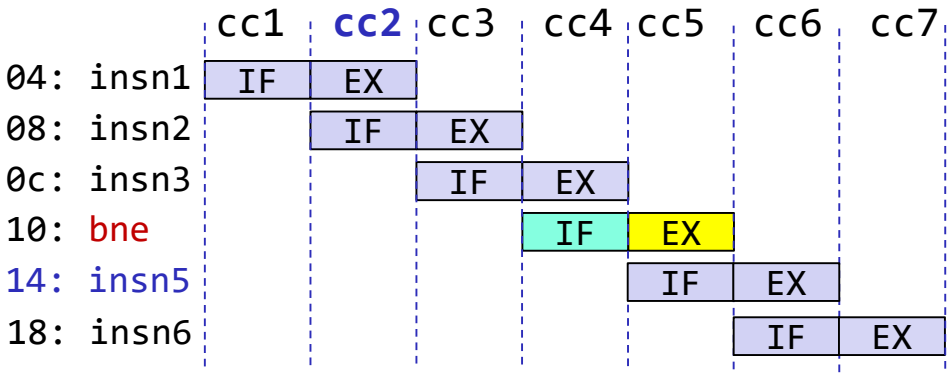


分岐が成立の場合

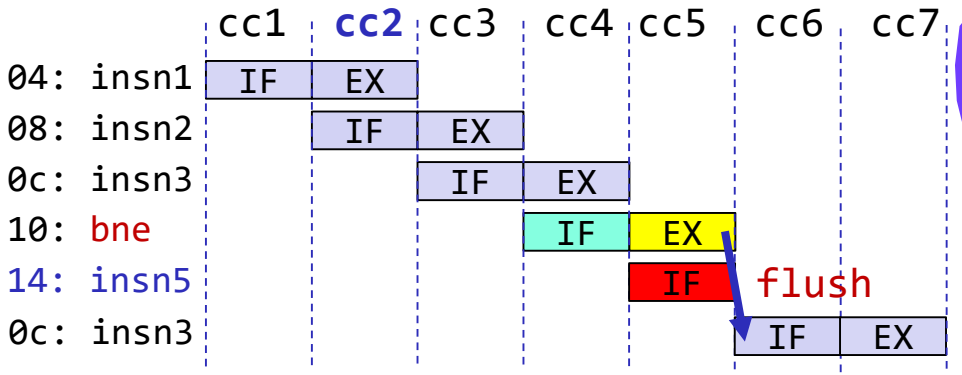
制御ハザードの対処：戦略2

- 分岐が不成立と仮定して分岐命令の後続命令の処理を進める.
  - その分岐が不成立の場合には，仮定が正しいので無駄は生じない.
  - この構成では，その分岐が成立の場合，間違ってフェッチした次の命令を削除(flush)して，正しい pc の命令のフェッチを再開する．1サイクルの無駄が発生する.

```
cm_ram[1]={12'd5,      5'd0, 3'b000, 5'd4, 7'b0010011};     // 04    addi x4, x0, 5   // x4 = 5
cm_ram[2]={12'd1,      5'd0, 3'b000, 5'd5, 7'b0010011};     // 08    addi x5, x0, 1   // x5 = 1
cm_ram[3]={12'd1,      5'd5, 3'b000, 5'd5, 7'b0010011};     // 0c L1:addi x5, x5, 1   // x5 = x5 + 1
cm_ram[4]={7'h7f,5'd4, 5'd5, 3'b001, 5'b11101, 7'b1100011}; // 10    bne  x5, x4, L1  // goto L1 if x5!=x4
cm_ram[5]={7'd0, 5'd0, 5'd5, 3'b000, 5'd30,7'b0110011};     // 14    add  x30,x5, x0  // led = x5
```



分岐が不成立の場合

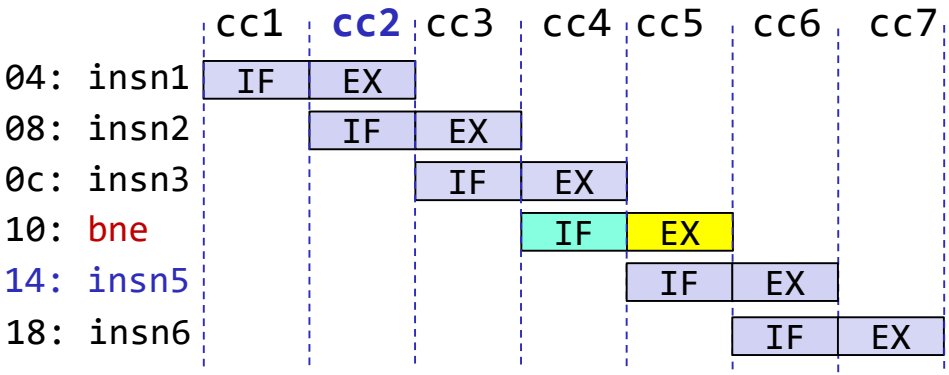


分岐が成立の場合

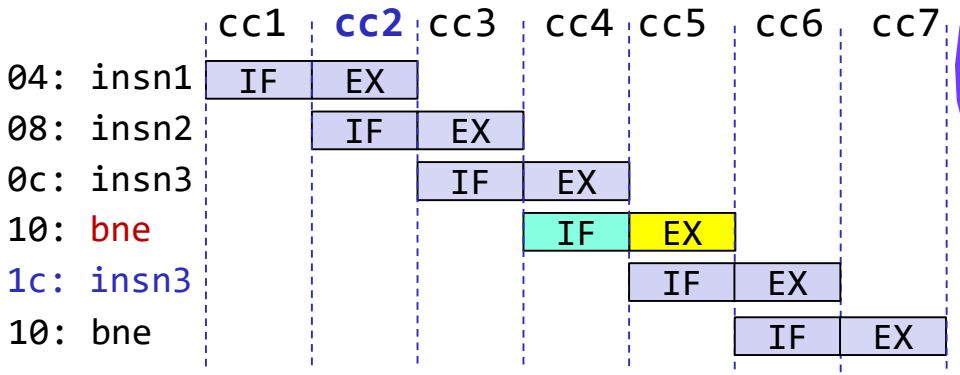
制御ハザードの対処：戦略3

- 分岐の成立／不成立を予測して，その予測が正しいと仮定して分岐命令の後続命令の処理を進める．
  - その分岐の結果が予測と同じ場合には，予測が成功で無駄は生じない．
  - この構成では，予測が失敗の場合，間違ってフェッチした次の命令を削除(flush)して，正しい pc の命令のフェッチを再開する．1サイクルの無駄が発生．

```
cm_ram[1]={12'd5,      5'd0, 3'b000, 5'd4, 7'b0010011};     // 04    addi x4, x0, 5   // x4 = 5
cm_ram[2]={12'd1,      5'd0, 3'b000, 5'd5, 7'b0010011};     // 08    addi x5, x0, 1   // x5 = 1
cm_ram[3]={12'd1,      5'd5, 3'b000, 5'd5, 7'b0010011};     // 0c L1:addi x5, x5, 1   // x5 = x5 + 1
cm_ram[4]={7'h7f,5'd4, 5'd5, 3'b001, 5'b11101, 7'b1100011}; // 10    bne  x5, x4, L1  // goto L1 if x5!=x4
cm_ram[5]={7'd0, 5'd0, 5'd5, 3'b000, 5'd30,7'b0110011};     // 14    add  x30,x5, x0  // led = x5
```



分岐が不成立と予測して成功の場合



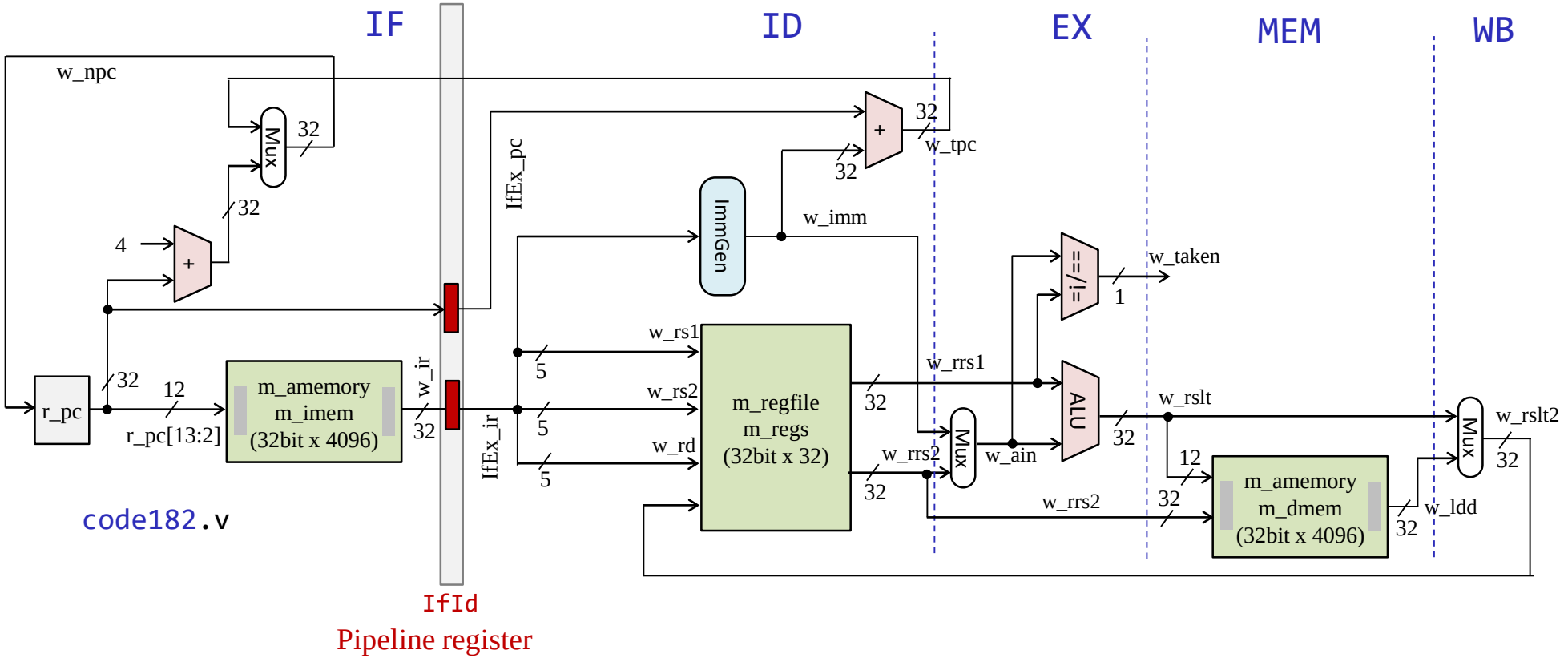
分岐が成立と予測して成功の場合

制御ハザードへの対処

- 戦略3の分岐予測を用いる方法は，コンピュータアーキテクチャの講義で扱う.
- この講義では，戦略2を採用する.
  - 分岐が不成立と仮定して分岐命令の後続命令の処理を進める.

m_proc13 2段のパイプライン版

- add, addi, sll, srl, lw, sw, beq, bne命令に対応したプロセッサ
- IF を1ステージ，その他の ID, EX, MEM, WB を1ステージとするパイプラインプロセッサ



m_proc13 2段のパイプライン版

- 分岐が成立 (w_tkn==1) の場合，間違ってフェッチした次の命令を削除 (flush) して，正しい pc の命令のフェッチを再開する．1サイクルの無駄が発生する．
- 具体的には，w_tkn==1 の時に，フェッチすべき命令を 0 (NOP) で上書きする．
  - always @(posedge w_clk) IfEx_ir <= (w_tkn) ? 0 : w_ir;
- 命令に有効ビット (valid bit) を付けてもよい．

```
module m_proc13 (w_clk, w_ce, w_led);
  input  wire w_clk, w_ce;
  output wire [31:0] w_led;

  reg [31:0] r_pc = 0;
  reg [31:0] IfEx_pc=0;
  reg [31:0] IfEx_ir=0;

  wire [31:0] w_ir;
  wire [4:0]  w_op5 = IfEx_ir[6:2];
  wire [4:0]  w_rs1 = IfEx_ir[19:15];
  wire [4:0]  w_rs2 = IfEx_ir[24:20];
  wire [4:0]  w_rd  = IfEx_ir[11:7];
  wire [2:0]  w_f3  = IfEx_ir[14:12];
```

code182.v

```
  wire [31:0] w_imm, w_rrs1, w_rrs2, w_ain, w_rslt, w_ldd, w_rslt2;
  wire #4 w_we = w_ce & (w_op5==5'b01100 || w_op5==5'b00100 || w_op5==5'b00000);
  always @(posedge w_clk) #5 IfEx_pc <= r_pc;
  m_amemory m_imem (w_clk, r_pc[13:2], 1'd0, 32'd0, w_ir);
  always @(posedge w_clk) #5 IfEx_ir <= (w_tkn) ? 0 : w_ir;

  m_immgen m_immgen0 (IfEx_ir, w_imm);

  m_regfile m_regs (w_clk, w_rs1, w_rs2, w_rd, w_we, w_rslt2, w_rrs1, w_rrs2);
  assign w_ain = (w_op5==5'b01100) ? w_rrs2 : w_imm;
  assign w_rslt = (w_f3==3'b001) ? w_rrs1 << w_ain[4:0] :
                  (w_f3==3'b101) ? w_rrs1 >> w_ain[4:0] : w_rrs1 + w_ain;

  m_amemory m_dmem (w_clk, w_rslt[13:2], (w_op5==5'b01000), w_rrs2, w_ldd);
  assign w_rslt2 = (w_op5==5'b00000) ? w_ldd : w_rslt;

  wire w_tkn = ({IfEx_ir[12],w_op5}==6'b011000 & w_rrs1==w_rrs2) ||  // BEQ
               ({IfEx_ir[12],w_op5}==6'b111000 & w_rrs1!=w_rrs2);    // BNE
  always @(posedge w_clk) #5
    if(w_ce && IfEx_ir!=32'h000f0033) r_pc <= (w_tkn) ? IfEx_pc + w_imm : r_pc+4;

  reg [31:0] r_led = 0;
  always @(posedge w_clk) if(w_ce & w_we & w_rd==30) r_led <= w_rslt;
  assign w_led = r_led;
endmodule
```

m_proc13 2段のパイプライン版

/home/tu_kise/cld/2023/baseline/program5.txt

```
initial begin
  cm_ram[0]={7'd0, 5'd0, 5'd0, 3'b000, 5'd0, 7'b0110011};     // 00    add  x0, x0, x0  // NOP
  cm_ram[1]={12'd5,      5'd0, 3'b000, 5'd4, 7'b0010011};     // 04    addi x4, x0, 5   // x4 = 5
  cm_ram[2]={12'd1,      5'd0, 3'b000, 5'd5, 7'b0010011};     // 08    addi x5, x0, 1   // x5 = 1
  cm_ram[3]={12'd1,      5'd5, 3'b000, 5'd5, 7'b0010011};     // 0c L1:addi x5, x5, 1   // x5 = x5 + 1
  cm_ram[4]={7'h7f,5'd4, 5'd5, 3'b001, 5'b11101, 7'b1100011}; // 10    bne  x5, x4, L1  // goto L1 if x5!=x4
  cm_ram[5]={7'd0, 5'd0, 5'd5, 3'b000, 5'd30,7'b0110011};     // 14    add  x30,x5, x0  // led = x5
  cm_ram[6]={7'd0, 5'd0, 5'd0, 3'b000, 5'd0, 7'b0110011};     // 18    add  x0, x0, x0  // NOP
  cm_ram[7]={7'd0, 5'd0, 5'd0, 3'b000, 5'd0, 7'b0110011};     // 1c    add  x0, x0, x0  // NOP
end
```

```
clock: r_pc     w_ir     w_rrs1   w_ain    r_rslt2  r_led
    1: 00000000 00000033 00000000 00000000 00000000 00000000
    2: 00000004 00500213 00000000 00000005 00000005 00000000
    3: 00000008 00100293 00000000 00000001 00000001 00000000
    4: 0000000c 00128293 00000001 00000001 00000002 00000000
    5: 00000010 fe429ee3 00000002 fffffffc 20000000 00000000
    6: 0000000c 00128293 00000002 00000001 00000003 00000000
    7: 00000010 fe429ee3 00000003 fffffffc 30000000 00000000
    8: 0000000c 00128293 00000003 00000001 00000004 00000000
    9: 00000010 fe429ee3 00000004 fffffffc 40000000 00000000
   10: 0000000c 00128293 00000004 00000001 00000005 00000000
   11: 00000010 fe429ee3 00000005 fffffffc 50000000 00000000
   12: 00000014 00028f33 00000005 00000000 00000005 00000000
   13: 00000018 00000033 00000000 00000000 00000000 00000005
```

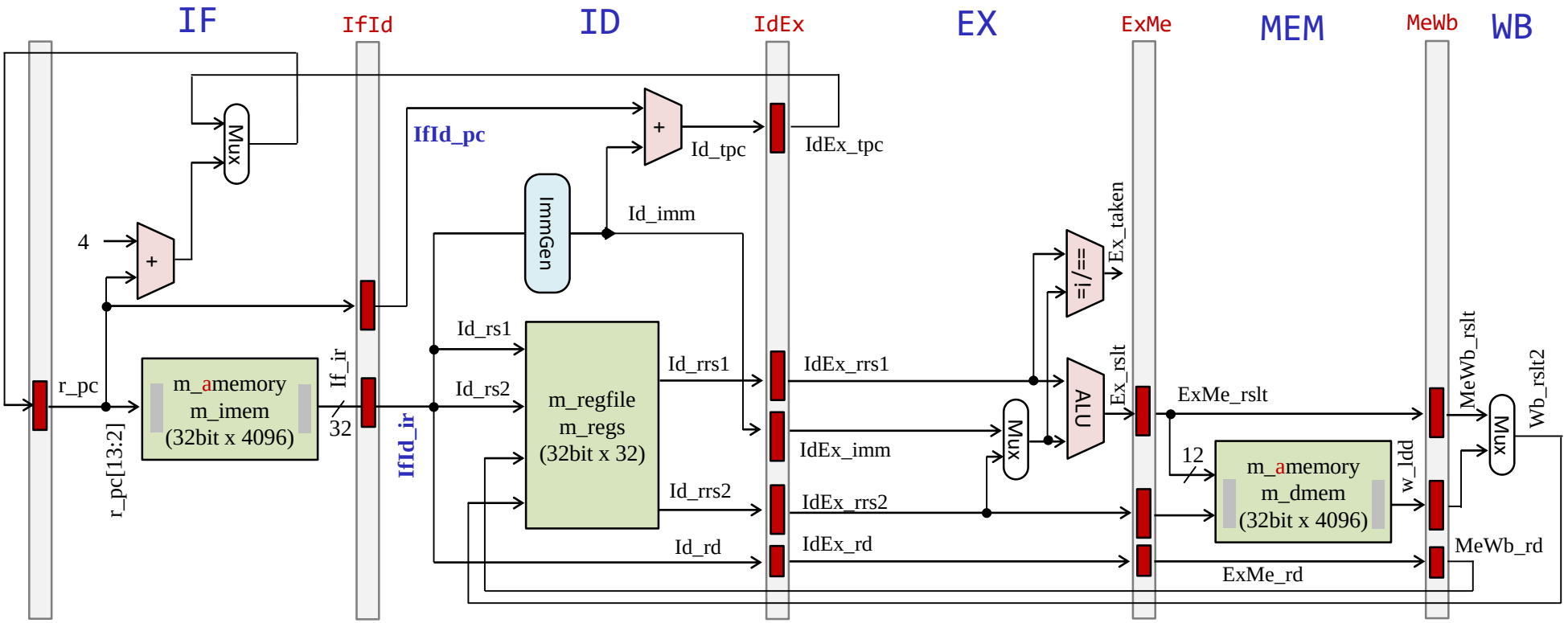
m_proc07 ベースラインのプロセッサの結果

```
clock: r_pc     IfEx_pc  IfEx_ir  w_rrs1   w_ain    r_rslt2  r_led
    1: 00000000 00000000 00000000 00000000 00000000 00000033 00000000
    2: 00000004 00000000 00000033 00000000 00000000 00000000 00000000
    3: 00000008 00000004 00500213 00000000 00000005 00000005 00000000
    4: 0000000c 00000008 00100293 00000000 00000001 00000001 00000000
    5: 00000010 0000000c 00128293 00000001 00000001 00000002 00000000
    6: 00000014 00000010 fe429ee3 00000002 fffffffc 20000000 00000000
    7: 0000000c 00000014 00000000 00000000 00000000 00000033 00000000
    8: 00000010 0000000c 00128293 00000002 00000001 00000003 00000000
    9: 00000014 00000010 fe429ee3 00000003 fffffffc 30000000 00000000
   10: 0000000c 00000014 00000000 00000000 00000000 00000033 00000000
   11: 00000010 0000000c 00128293 00000003 00000001 00000004 00000000
   12: 00000014 00000010 fe429ee3 00000004 fffffffc 40000000 00000000
   13: 0000000c 00000014 00000000 00000000 00000000 00000033 00000000
   14: 00000010 0000000c 00128293 00000004 00000001 00000005 00000000
   15: 00000014 00000010 fe429ee3 00000005 fffffffc 50000000 00000000
   16: 00000018 00000014 00028f33 00000005 00000000 00000005 00000000
   17: 0000001c 00000018 00000033 00000000 00000000 00000000 00000005
```

m_proc13 2段のパイプライン版の結果

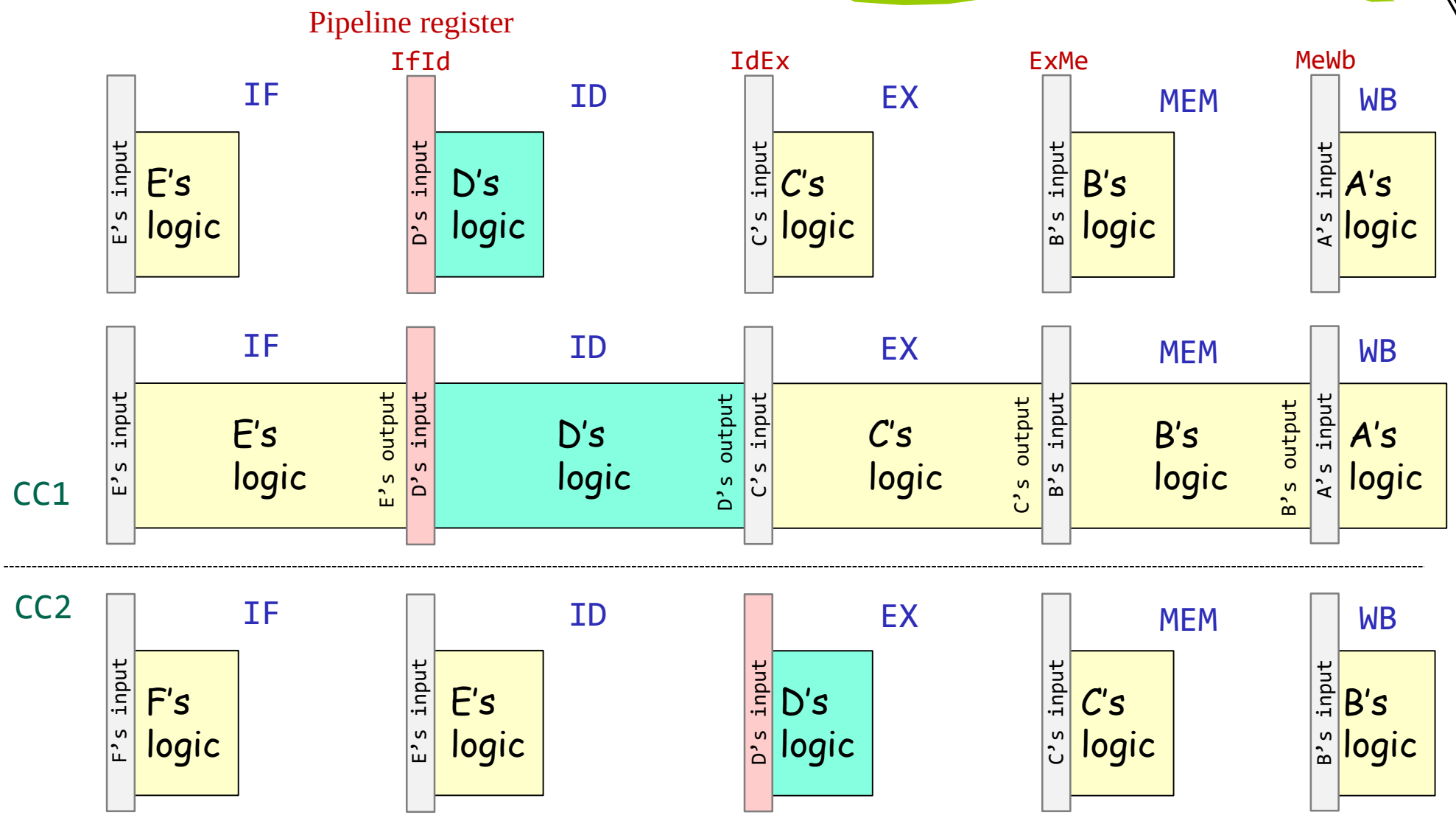
m_proc14 5段のパイプライン版

- add, addi, sll, srl, lw, sw, beq, bne命令に対応したプロセッサ（データフォワーディング無し）
- IF, ID, EX, MEM, WB のそれぞれをステージとする5段のパイプラインプロセッサ
- ステージ IF と ID の間のパイプラインレジスタには IfId_ から始まる名前を利用する.
- ステージ ID で生成される配線には Id_ から始まる名前を利用する.



code185.v

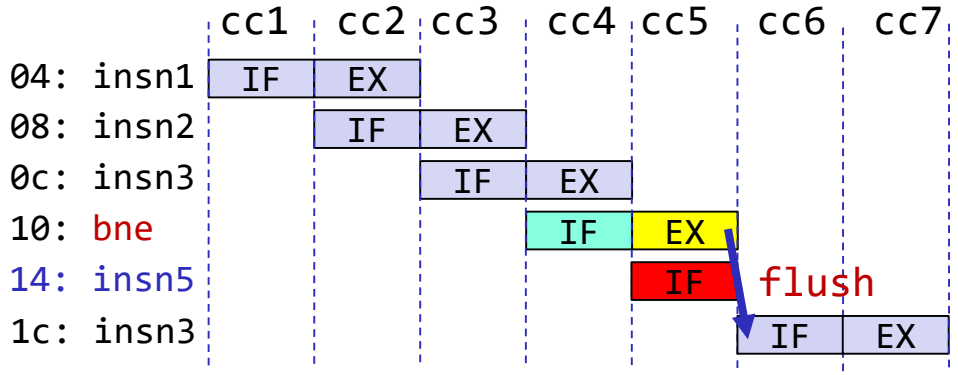
5段のパイプラインプロセッサとパイプラインレジスタ



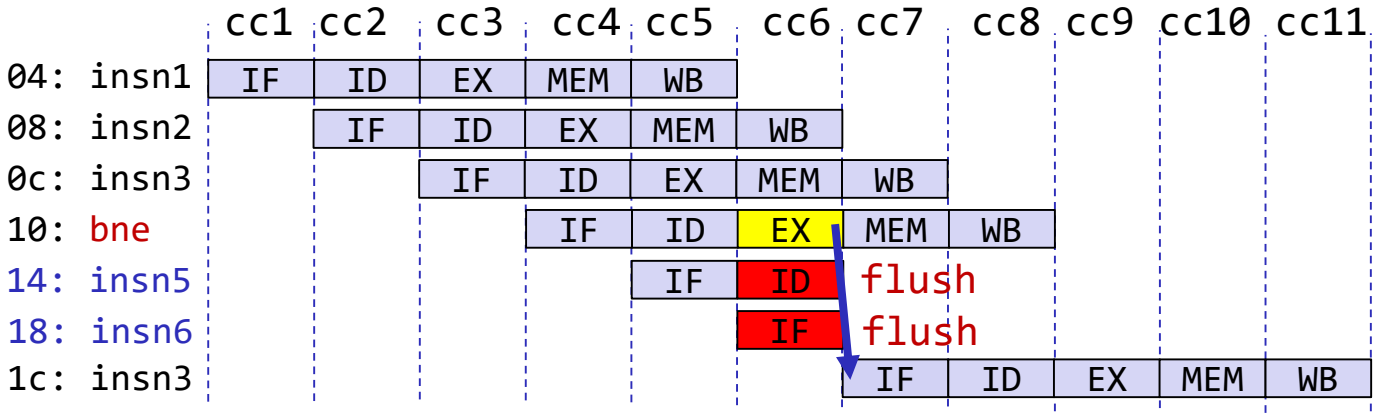
m_proc14 5段のパイプライン版の制御ハザードの対処

- 5段のパイプラインプロセッサでは，EXステージ（図ではcc6の黄色）で分岐の結果がわかる．
- この時，ID と IF で処理している命令を NOP に変更（flush，赤色）することで，無効な命令（正しくない制御の命令）を実行しない．

2段のパイプライン
分岐が成立の場合



5段のパイプライン
分岐が成立の場合



m_proc14 5段のパイプライン版のテストプログラム

- データ・ハザードを無くすために NOP を挿入する.

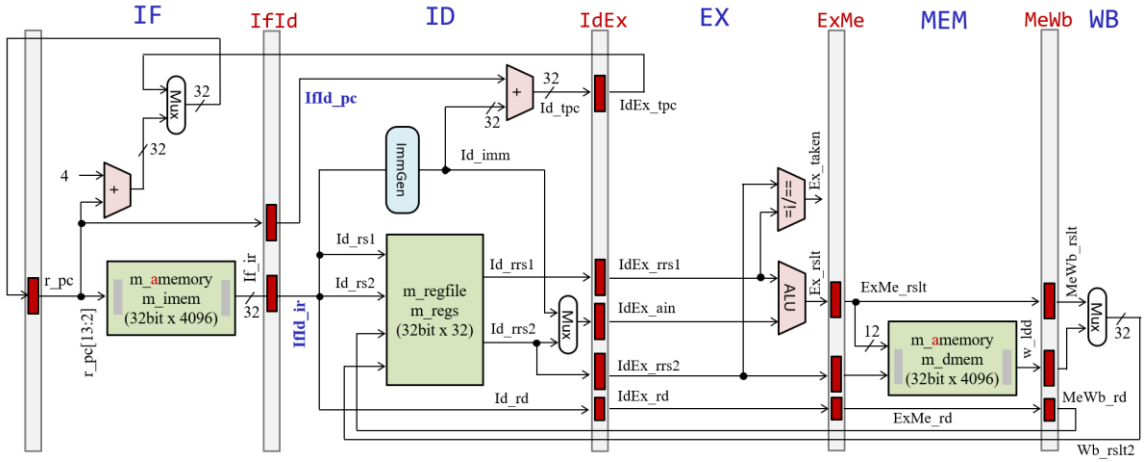
/home/tu_kise/cld/2023/baseline/program6.txt

```
initial begin
  cm_ram[ 0]={7'd0, 5'd0, 5'd0, 3'b000, 5'd0, 7'b0110011};     //    add  x0, x0, x0  // NOP
  cm_ram[ 1]={12'd5,      5'd0, 3'b000, 5'd4, 7'b0010011};     //    addi x4, x0, 5   // x4 = 5
  cm_ram[ 2]={12'd1,      5'd0, 3'b000, 5'd5, 7'b0010011};     //    addi x5, x0, 1   // x5 = 1
  cm_ram[ 3]={7'd0, 5'd0, 5'd0, 3'b000, 5'd0, 7'b0110011};     //    add  x0, x0, x0  // NOP
  cm_ram[ 4]={7'd0, 5'd0, 5'd0, 3'b000, 5'd0, 7'b0110011};     //    add  x0, x0, x0  // NOP
  cm_ram[ 5]={7'd0, 5'd0, 5'd0, 3'b000, 5'd0, 7'b0110011};     //    add  x0, x0, x0  // NOP
  cm_ram[ 6]={7'd0, 5'd0, 5'd0, 3'b000, 5'd0, 7'b0110011};     //    add  x0, x0, x0  // NOP
  cm_ram[ 7]={12'd1,      5'd5, 3'b000, 5'd5, 7'b0010011};     // L1:addi x5, x5, 1   // x5 = x5 + 1
  cm_ram[ 8]={7'd0, 5'd0, 5'd0, 3'b000, 5'd0, 7'b0110011};     //    add  x0, x0, x0  // NOP
  cm_ram[ 9]={7'd0, 5'd0, 5'd0, 3'b000, 5'd0, 7'b0110011};     //    add  x0, x0, x0  // NOP
  cm_ram[10]={7'd0, 5'd0, 5'd0, 3'b000, 5'd0, 7'b0110011};     //    add  x0, x0, x0  // NOP
  cm_ram[11]={7'd0, 5'd0, 5'd0, 3'b000, 5'd0, 7'b0110011};     //    add  x0, x0, x0  // NOP
  cm_ram[12]={7'h7f,5'd4, 5'd5, 3'b001, 5'b0110_1, 7'b1100011};//    bne  x5, x4, L1  // goto L1 if x5!=x4
  cm_ram[13]={7'd0, 5'd0, 5'd0, 3'b000, 5'd0, 7'b0110011};     //    add  x0, x0, x0  // NOP
  cm_ram[14]={7'd0, 5'd0, 5'd0, 3'b000, 5'd0, 7'b0110011};     //    add  x0, x0, x0  // NOP
  cm_ram[15]={7'd0, 5'd0, 5'd0, 3'b000, 5'd0, 7'b0110011};     //    add  x0, x0, x0  // NOP
  cm_ram[16]={7'd0, 5'd0, 5'd0, 3'b000, 5'd0, 7'b0110011};     //    add  x0, x0, x0  // NOP
  cm_ram[17]={7'd0, 5'd0, 5'd5, 3'b000, 5'd30,7'b0110011};     //    add  x30,x5, x0  // led = x5
  cm_ram[18]={7'd0, 5'd0, 5'd0, 3'b000, 5'd0, 7'b0110011};     //    add  x0, x0, x0  // NOP
  cm_ram[19]={7'd0, 5'd0, 5'd0, 3'b000, 5'd0, 7'b0110011};     //    add  x0, x0, x0  // NOP
end
```

m_proc14 5段のパイプライン版

- ステージ IF と ID の間のパイプラインレジスタには IfId_ から始まる名前を利用する.
- ステージ ID で生成される配線には Id_ から始まる名前を利用する.

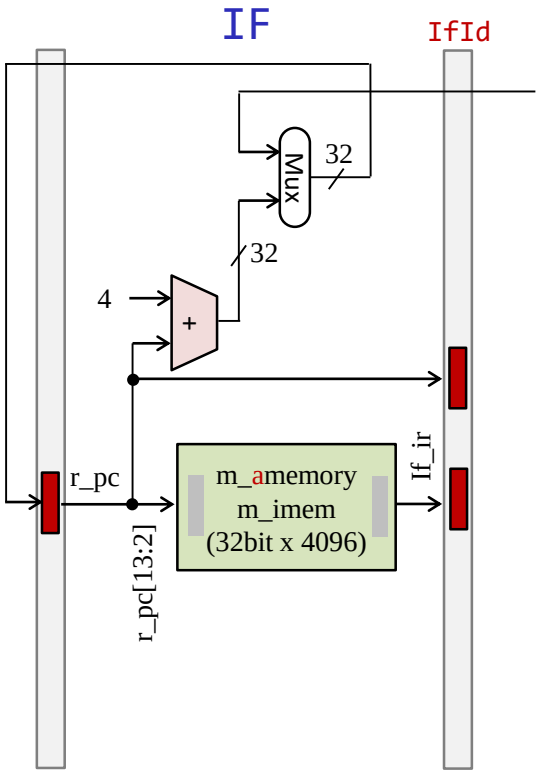
```
module m_proc14 (w_clk, w_ce, w_led);
  input  wire w_clk, w_ce;
  output wire [31:0] w_led;

  reg [31:0] r_pc      = 0; // program counter
  reg [31:0] IfId_pc   = 0; // IfId pipeline registers
  reg [31:0] IfId_ir   = 0;

  reg [31:0] IdEx_pc   = 0; // IdEx pipeline registers
  reg [31:0] IdEx_ir   = 0;
  reg [31:0] IdEx_tpc  = 0;
  reg [31:0] IdEx_rrs1 = 0;
  reg [31:0] IdEx_rrs2 = 0;
  reg [31:0] IdEx_imm  = 0;
  reg [4:0]  IdEx_rd   = 0;

  reg [31:0] ExMe_pc   = 0; // ExMe pipeline registers
  reg [31:0] ExMe_ir   = 0;
  reg [31:0] ExMe_rslt = 0;
  reg [31:0] ExMe_rrs2 = 0;
  reg [4:0]  ExMe_rd   = 0;

  reg [31:0] MeWb_pc   = 0; // MeWb pipeline registers
  reg [31:0] MeWb_ir   = 0;
  reg [31:0] MeWb_rslt = 0;
  reg [31:0] MeWb_ldd  = 0;
  reg [4:0]  MeWb_rd   = 0;
```



/home/tu_kise/cld/2023/code185.v

m_proc14 5段のパイプライン版

- ステージ IF と ID の間のパイプラインレジスタには IfId_ から始まる名前を利用する.
- ステージ ID で生成される配線には Id_ から始まる名前を利用する.

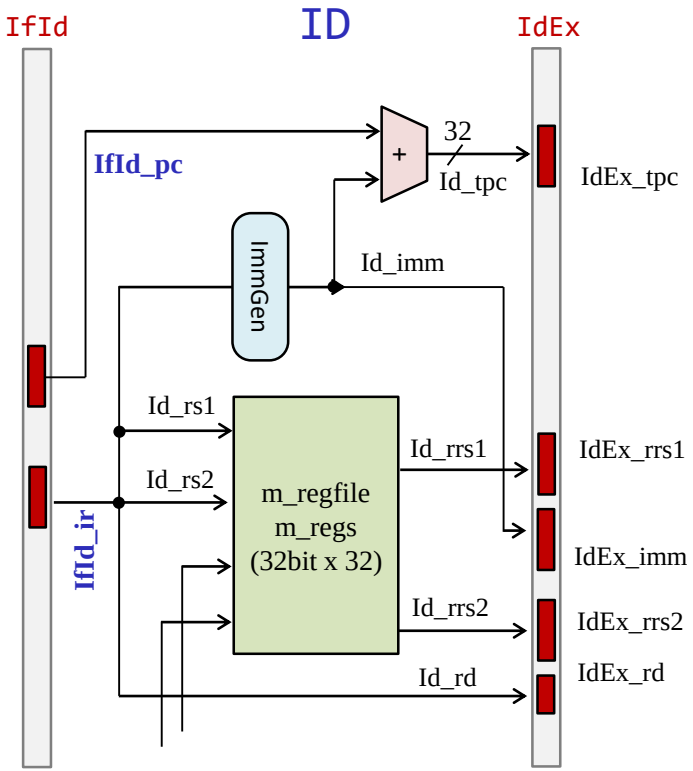



```
/********************** IF stage *******************************/
 wire [31:0] If_ir;
 m_amemory m_imem (w_clk, r_pc[13:2], 1'd0, 32'd0, If_ir);
 always @(posedge w_clk) #5 if(w_ce) begin
   IfId_pc <= (Ex_taken) ? 0 : r_pc;
   IfId_ir <= (Ex_taken) ? {25'd0, 7'b0010011} : If_ir;
 end
 /********************** ID stage *******************************/
 wire [4:0] Id_op5 = IfId_ir[6:2];
 wire [4:0] Id_rs1 = IfId_ir[19:15];
 wire [4:0] Id_rs2 = IfId_ir[24:20];
 wire [4:0] Id_rd  = IfId_ir[11:7];
 wire Id_we = (Id_op5==5'b01100 || Id_op5==5'b00100 || Id_op5==5'b00000);
 wire [31:0] Id_imm, Id_rrs1, Id_rrs2;
 m_immgen m_immgen0 (IfId_ir, Id_imm);
 m_regfile m_regs (w_clk, Id_rs1, Id_rs2, MeWb_rd, 1'b1, Wb_rslt2,
    Id_rrs1, Id_rrs2);
 always @(posedge w_clk) #5 if(w_ce) begin
   IdEx_pc   <= (Ex_taken) ? 0 : IfId_pc;
   IdEx_ir   <= (Ex_taken) ? {25'd0, 7'b0010011} : IfId_ir;
   IdEx_tpc  <= IfId_pc + Id_imm;
   IdEx_imm  <= Id_imm;
   IdEx_rrs1 <= Id_rrs1;
   IdEx_rrs2 <= Id_rrs2;
   IdEx_rd   <= (Id_we==0 || Ex_taken) ? 0 : Id_rd; // Note
 end
```

/home/tu_kise/cld/2023/code185.v

m_proc14 5段のパイプライン版

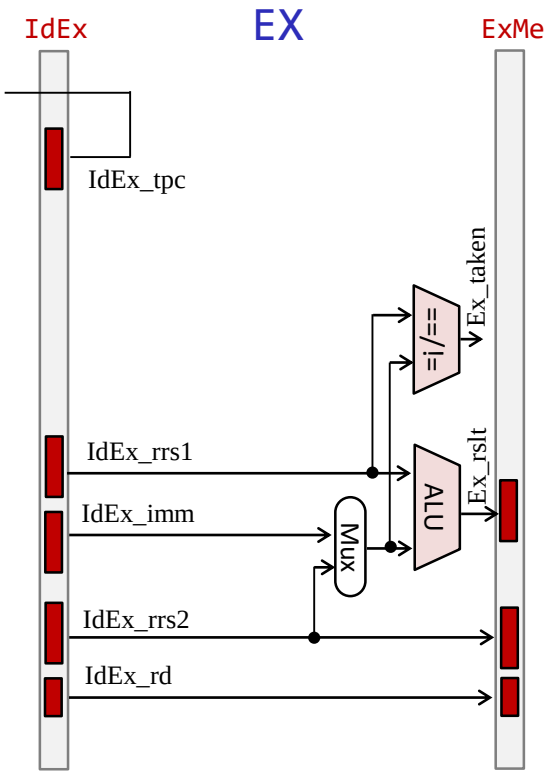
- ステージ IF と ID の間のパイプラインレジスタには IfId_ から始まる名前を利用する.
- ステージ ID で生成される配線には Id_ から始まる名前を利用する.




```
/********************** IF stage *******************************/
 wire [31:0] If_ir;
 m_amemory m_imem (w_clk, r_pc[13:2], 1'd0, 32'd0, If_ir);
 always @(posedge w_clk) #5 if(w_ce) begin
   IfId_pc <= (Ex_taken) ? 0 : r_pc;
   IfId_ir <= (Ex_taken) ? {25'd0, 7'b0010011} : If_ir;
 end
 /********************** ID stage *******************************/
 wire [4:0] Id_op5 = IfId_ir[6:2];
 wire [4:0] Id_rs1 = IfId_ir[19:15];
 wire [4:0] Id_rs2 = IfId_ir[24:20];
 wire [4:0] Id_rd  = IfId_ir[11:7];
 wire Id_we = (Id_op5==5'b01100 || Id_op5==5'b00100 || Id_op5==5'b00000);
 wire [31:0] Id_imm, Id_rrs1, Id_rrs2;
 m_immgen m_immgen0 (IfId_ir, Id_imm);
 m_regfile m_regs (w_clk, Id_rs1, Id_rs2, MeWb_rd, 1'b1, Wb_rslt2,
    Id_rrs1, Id_rrs2);
 always @(posedge w_clk) #5 if(w_ce) begin
   IdEx_pc   <= (Ex_taken) ? 0 : IfId_pc;
   IdEx_ir   <= (Ex_taken) ? {25'd0, 7'b0010011} : IfId_ir;
   IdEx_tpc  <= IfId_pc + Id_imm;
   IdEx_imm  <= Id_imm;
   IdEx_rrs1 <= Id_rrs1;
   IdEx_rrs2 <= Id_rrs2;
   IdEx_rd   <= (Id_we==0 || Ex_taken) ? 0 : Id_rd; // Note
 end
```

/home/tu_kise/cld/2023/code185.v

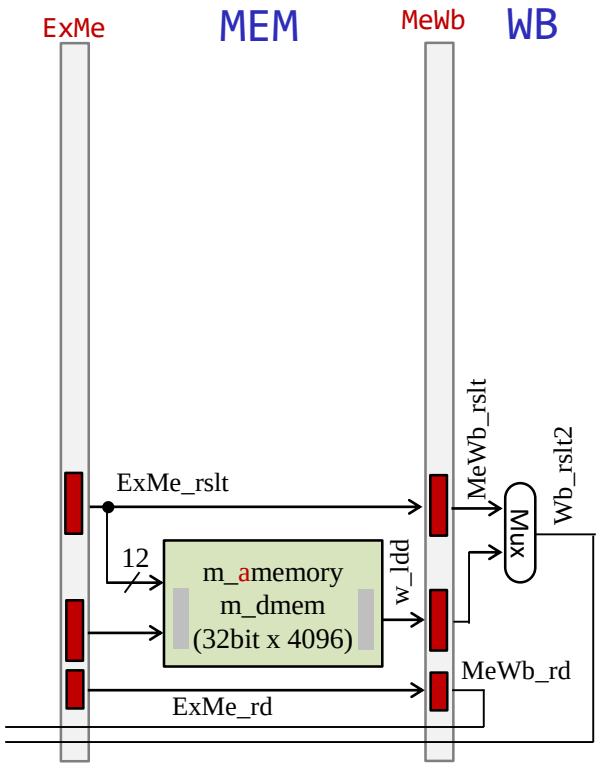
m_proc14 5段のパイプライン版




```
/********************** Ex stage ********************************/
 wire [4:0] Ex_op5 = IdEx_ir[6:2];
 wire Ex_SLL = (IdEx_ir[14:12]==3'b001);
 wire Ex_SRL = (IdEx_ir[14:12]==3'b101);
 wire Ex_BEQ = ({IdEx_ir[12], Ex_op5}==6'b011000);
 wire Ex_BNE = ({IdEx_ir[12], Ex_op5}==6'b111000);
 wire [31:0] Ex_ain = (Ex_op5==5'b01100 || Ex_op5==5'b11000) ? IdEx_rrs2 : IdEx_imm;
 wire [31:0] Ex_rslt = (Ex_SLL) ? IdEx_rrs1 << Ex_ain[4:0] :
                       (Ex_SRL) ? IdEx_rrs1 >> Ex_ain[4:0] : IdEx_rrs1 + Ex_ain;
 wire Ex_taken = (Ex_BEQ & IdEx_rrs1==Ex_ain) || (Ex_BNE & IdEx_rrs1!=Ex_ain);
 always @(posedge w_clk) #5 if(w_ce) begin
   ExMe_pc   <= IdEx_pc;
   ExMe_ir   <= IdEx_ir;
   ExMe_rslt <= Ex_rslt;
   ExMe_rrs2 <= IdEx_rrs2;
   ExMe_rd   <= IdEx_rd;
 end
```

/home/tu_kise/cld/2023/code185.v

m_proc14 5段のパイプライン版




```
/********************** Me stage ********************************/
  wire [4:0] Me_op5 = ExMe_ir[6:2];
  wire       Me_we = (Me_op5==5'b01100 || Me_op5==5'b00100 || Me_op5==5'b00000);
  wire [31:0] Me_ldd;
  m_amemory m_dmem (w_clk, ExMe_rslt[13:2], (Me_op5==5'b01000), ExMe_rrs2, Me_ldd);
  always @(posedge w_clk) #5 if(w_ce) begin
     MeWb_pc   <= ExMe_pc;
     MeWb_ir   <= ExMe_ir;
     MeWb_rslt <= ExMe_rslt;
     MeWb_ldd  <= Me_ldd;
     MeWb_rd   <= ExMe_rd;
  end
  /********************** Wb stage ********************************/
  wire Wb_LW = (MeWb_ir[6:2]==5'b00000);
  wire [31:0] Wb_rslt2 = (Wb_LW) ? MeWb_ldd : MeWb_rslt;
  always @(posedge w_clk) #5
    if(w_ce && IfId_ir!=32'h000f0033) r_pc <= (Ex_taken) ? IdEx_tpc : r_pc+4;

  reg [31:0] r_led = 0;
  always @(posedge w_clk) if(w_ce & MeWb_rd==30) r_led <= Wb_rslt2;
  assign w_led = r_led;
endmodule
```

/home/tu_kise/cld/2023/code185.v

m_proc14 5段のパイプライン版の実行結果

t の列は Ex_taken を表示

```
clock: r_pc     IfId_pc  IdEx_pc  ExMe_pc  MeWb_pc : t Id_rrs1  Id_rrs2  Ex_ain   Ex_rslt  Wb_rslt2 w_led
    1: 00000000 00000000 00000000 00000000 00000000: 0 00000000 00000000 00000000 00000000 00000000 00000000
    2: 00000004 00000000 00000000 00000000 00000000: 0 00000000 00000000 00000000 00000000 00000033 00000000
    3: 00000008 00000004 00000000 00000000 00000000: 0 00000000 xxxxxxxx 00000000 00000000 00000033 00000000
    4: 0000000c 00000008 00000004 00000000 00000000: 0 00000000 xxxxxxxx 00000005 00000005 00000033 00000000
    5: 00000010 0000000c 00000008 00000004 00000000: 0 00000000 00000000 00000001 00000001 00000000 00000000
    6: 00000014 00000010 0000000c 00000008 00000004: 0 00000000 00000000 00000000 00000000 00000005 00000000
    7: 00000018 00000014 00000010 0000000c 00000008: 0 00000000 00000000 00000000 00000000 00000001 00000000
    8: 0000001c 00000018 00000014 00000010 0000000c: 0 00000000 00000000 00000000 00000000 00000000 00000000
    9: 00000020 0000001c 00000018 00000014 00000010: 0 00000001 xxxxxxxx 00000000 00000000 00000000 00000000
   10: 00000024 00000020 0000001c 00000018 00000014: 0 00000000 00000000 00000001 00000002 00000000 00000000
   11: 00000028 00000024 00000020 0000001c 00000018: 0 00000000 00000000 00000000 00000000 00000000 00000000
   12: 0000002c 00000028 00000024 00000020 0000001c: 0 00000000 00000000 00000000 00000000 00000002 00000000
   13: 00000030 0000002c 00000028 00000024 00000020: 0 00000000 00000000 00000000 00000000 00000000 00000000
   14: 00000034 00000030 0000002c 00000028 00000024: 0 00000002 00000005 00000000 00000000 00000000 00000000
   15: 00000038 00000034 00000030 0000002c 00000028: 1 00000000 00000000 00000005 00000040 00000000 00000000
   16: 0000001c 00000000 00000000 00000030 0000002c: 0 00000000 00000000 00000000 00000000 00000000 00000000
   17: 00000020 0000001c 00000000 00000000 00000030: 0 00000002 xxxxxxxx 00000000 00000000 00000040 00000000
   18: 00000024 00000020 0000001c 00000000 00000000: 0 00000000 00000000 00000001 00000003 00000000 00000000
   19: 00000028 00000024 00000020 0000001c 00000000: 0 00000000 00000000 00000000 00000000 00000000 00000000
   20: 0000002c 00000028 00000024 00000020 0000001c: 0 00000000 00000000 00000000 00000000 00000003 00000000
   21: 00000030 0000002c 00000028 00000024 00000020: 0 00000000 00000000 00000000 00000000 00000000 00000000
   22: 00000034 00000030 0000002c 00000028 00000024: 0 00000003 00000005 00000000 00000000 00000000 00000000
   23: 00000038 00000034 00000030 0000002c 00000028: 1 00000000 00000000 00000005 00000060 00000000 00000000
   24: 0000001c 00000000 00000000 00000030 0000002c: 0 00000000 00000000 00000000 00000000 00000000 00000000
   25: 00000020 0000001c 00000000 00000000 00000030: 0 00000003 xxxxxxxx 00000000 00000000 00000060 00000000
   26: 00000024 00000020 0000001c 00000000 00000000: 0 00000000 00000000 00000001 00000004 00000000 00000000
   27: 00000028 00000024 00000020 0000001c 00000000: 0 00000000 00000000 00000000 00000000 00000000 00000000
   28: 0000002c 00000028 00000024 00000020 0000001c: 0 00000000 00000000 00000000 00000000 00000004 00000000
   29: 00000030 0000002c 00000028 00000024 00000020: 0 00000000 00000000 00000000 00000000 00000000 00000000
   30: 00000034 00000030 0000002c 00000028 00000024: 0 00000004 00000005 00000000 00000000 00000000 00000000
   31: 00000038 00000034 00000030 0000002c 00000028: 1 00000000 00000000 00000005 00000080 00000000 00000000
   32: 0000001c 00000000 00000000 00000030 0000002c: 0 00000000 00000000 00000000 00000000 00000000 00000000
   33: 00000020 0000001c 00000000 00000000 00000030: 0 00000004 xxxxxxxx 00000000 00000000 00000080 00000000
   34: 00000024 00000020 0000001c 00000000 00000000: 0 00000000 00000000 00000001 00000005 00000000 00000000
   35: 00000028 00000024 00000020 0000001c 00000000: 0 00000000 00000000 00000000 00000000 00000000 00000000
   36: 0000002c 00000028 00000024 00000020 0000001c: 0 00000000 00000000 00000000 00000000 00000005 00000000
   37: 00000030 0000002c 00000028 00000024 00000020: 0 00000000 00000000 00000000 00000000 00000000 00000000
   38: 00000034 00000030 0000002c 00000028 00000024: 0 00000005 00000005 00000000 00000000 00000000 00000000
   39: 00000038 00000034 00000030 0000002c 00000028: 0 00000000 00000000 00000005 000000a0 00000000 00000000
   40: 0000003c 00000038 00000034 00000030 0000002c: 0 00000000 00000000 00000000 00000000 00000000 00000000
   41: 00000040 0000003c 00000038 00000034 00000030: 0 00000000 00000000 00000000 00000000 000000a0 00000000
   42: 00000044 00000040 0000003c 00000038 00000034: 0 00000000 00000000 00000000 00000000 00000000 00000000
   43: 00000048 00000044 00000040 0000003c 00000038: 0 00000005 00000000 00000000 00000000 00000000 00000000
   44: 0000004c 00000048 00000044 00000040 0000003c: 0 00000000 00000000 00000000 00000005 00000000 00000000
   45: 00000050 0000004c 00000048 00000044 00000040: 0 00000000 00000000 00000000 00000000 00000000 00000000
   46: 00000054 00000050 0000004c 00000048 00000044: 0 xxxxxxxx xxxxxxxx 00000000 00000000 00000005 00000000
   47: 00000054 00000054 00000050 0000004c 00000048: x xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx 00000000 00000005
```

/home/tu_kise/
cld/2023/baseline/
program6.txt
の実行結果

Hazards make pipelining hard (topic of the next lecture)

- 命令を適切なサイクルで実行できないような状況が存在する．これをハザード(hazard)と呼ぶ．
  - 構造ハザード (structural hazard)
    - オーバラップ実行する命令の組み合わせをハードウェアがサポートしていない場合．資源不足により生じる．
  - データ・ハザード(data hazard)
    - データの受け渡しの制約によって生じるハザード

```
cm_ram[0]={7'd0, 5'd0, 5'd0, 3'b000, 5'd0, 7'b0110011}; //    add  x0, x0, x0  // NOP
cm_ram[1]={12'd3,      5'd0, 3'b000, 5'd4, 7'b0010011}; //    addi x4, x0, 3   // x4 = 3
cm_ram[2]={12'd4,      5'd0, 3'b000, 5'd5, 7'b0010011}; //    addi x5, x0, 4   // x5 = 4
cm_ram[3]={7'd0, 5'd5, 5'd4, 3'b000, 5'd30,7'b0110011}; //    add  x30,x4, x5  // led = x4 + x5
```

  - 制御ハザード(control hazard)
    - 分岐命令，ジャンプ命令によって生じるハザード

```
cm_ram[1]={12'd5,      5'd0, 3'b000, 5'd4, 7'b0010011};     //    addi x4, x0, 5   // x4 = 5
cm_ram[2]={12'd1,      5'd0, 3'b000, 5'd5, 7'b0010011};     //    addi x5, x0, 1   // x5 = 1
cm_ram[3]={12'd1,      5'd5, 3'b000, 5'd5, 7'b0010011};     // L1:addi x5, x5, 1   // x5 = x5 + 1
cm_ram[4]={7'h7f,5'd4, 5'd5, 3'b001, 5'b11101, 7'b1100011}; //    bne  x5, x4, L1  // goto L1 if x5!=x4
cm_ram[5]={7'd0, 5'd0, 5'd5, 3'b000, 5'd30,7'b0110011};     //    add  x30,x5, x0  // led = x5
```

References

- Computer Logic Design support page
  - https://www.arch.cs.titech.ac.jp/lecture/CLD/
- ACRi Room
  - https://gw.acri.c.titech.ac.jp
- ACRi Blog
  - https://www.acri.c.titech.ac.jp/wordpress/
- 情報工学系計算機室
  - http://www.csc.titech.ac.jp/
- Xilinx Vivado Design Suite
  - https://japan.xilinx.com/products/design-tools/vivado.html
- Digilent Arty A7-35T
  - https://reference.digilentinc.com/reference/programmable-logic/arty-a7/start
- Verilog HDL
  - https://ja.wikipedia.org/wiki/Verilog