Course number: CSC.T341

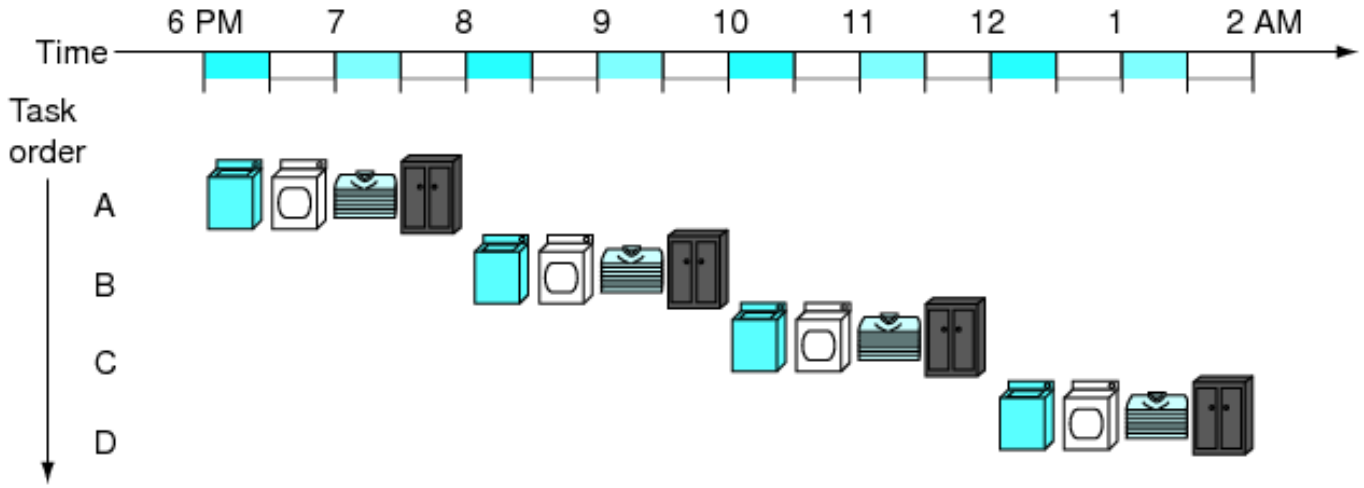
コンピュータ論理設計
Computer Logic Design

14. パイプライン処理とデータハザード
Pipelining and Data Hazard

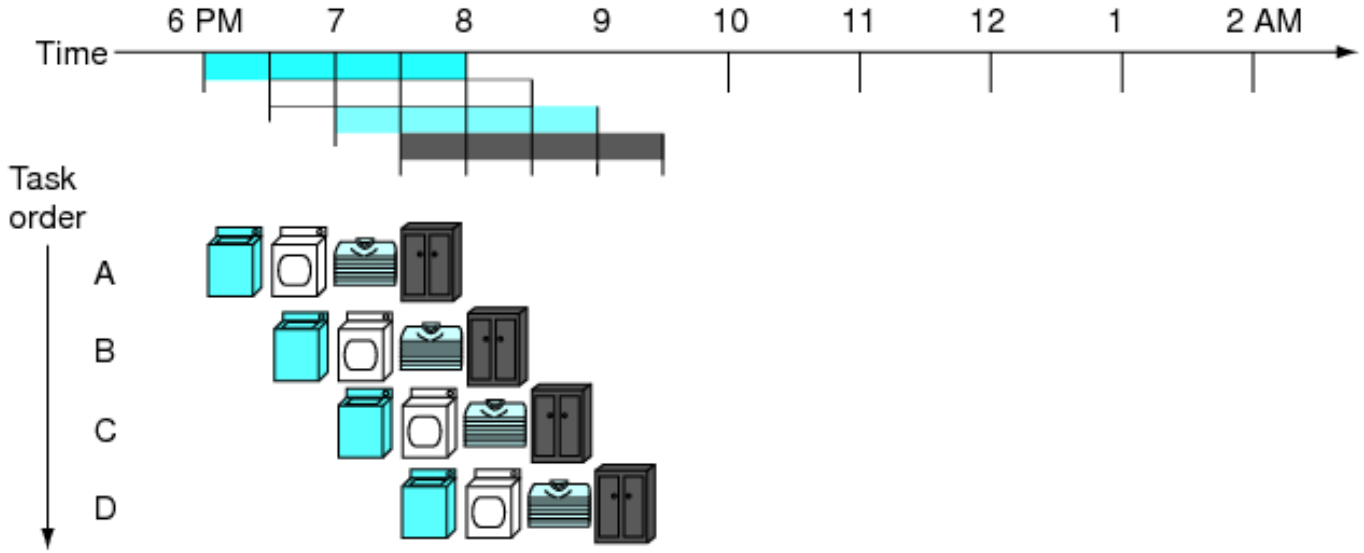
吉瀬 謙二　情報工学系
Kenji Kise, Department of Computer Science
kise _at_ c.titech.ac.jp　www.arch.cs.titech.ac.jp/lecture/CLD/
W621 講義室　月 10:45-12:15, 木 9:00-12:15

Pipelined Processor

- Non pipelining (Multi-cycle)

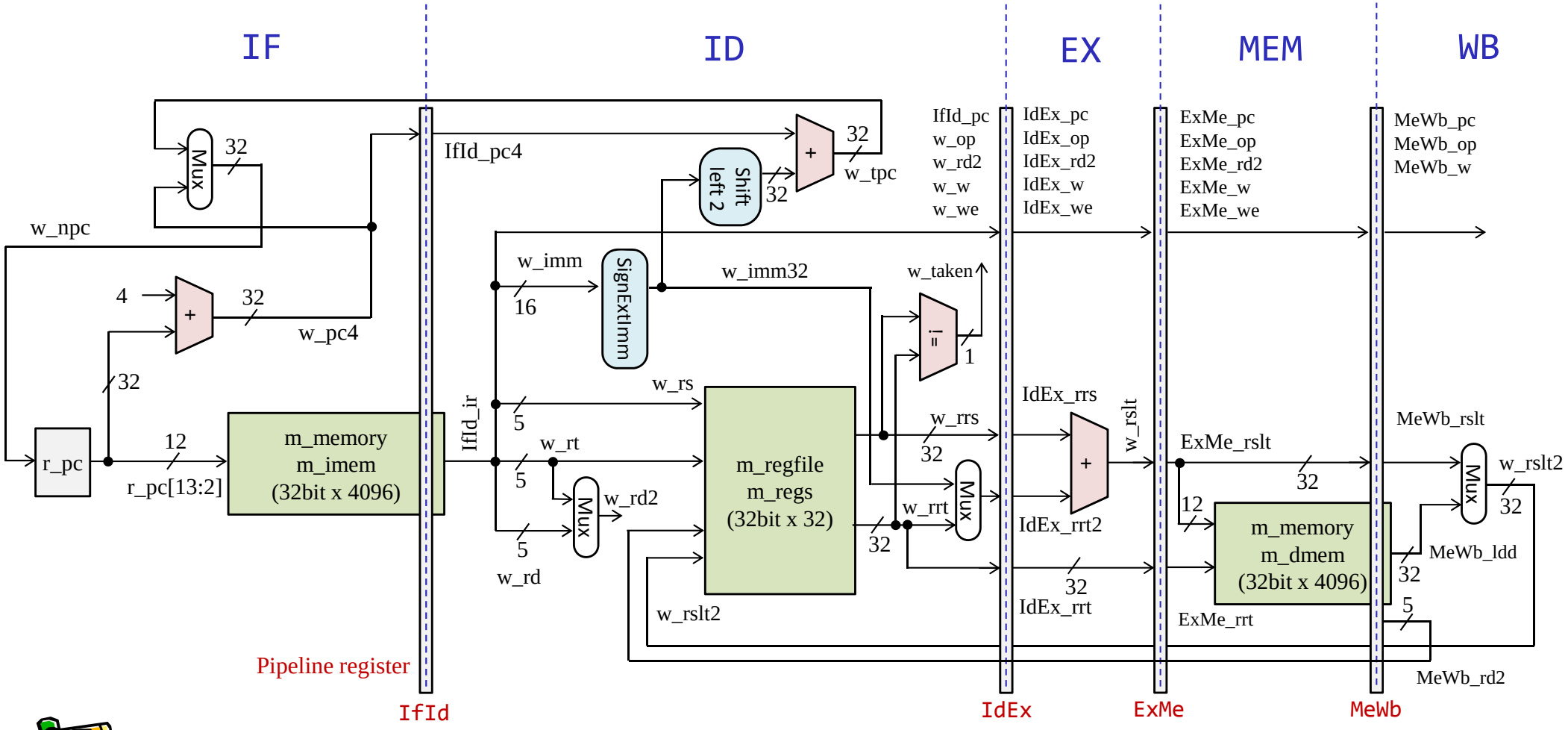


- Pipelining



Inside module m_proc11 (pipelined about 100MHz)

- add, addi, lw, sw, bne, halt命令に対応したデータハザードを考慮しないパイプライン版
- 最も遅延の長い（長い時間を要する）ステージがプロセッサの動作周波数を決める.



Inside module m_proc11 (pipelined about 100MHz)

code140.v

```
module m_proc11 (w_clk, w_rst, r_rout, r_halt);
  input  wire w_clk, w_rst;
  output reg [31:0] r_rout;
  output reg        r_halt;

  reg  [31:0] IfId_pc4=0;                                    // pipe regs
  reg  [31:0] IdEx_rrs=0, IdEx_rrt=0, IdEx_rrt2=0;           //
  reg  [31:0] ExMe_rslt=0, ExMe_rrt=0;                       //
  reg  [31:0] MeWb_rslt=0;                                   //
  reg   [5:0]             IdEx_op=0,  ExMe_op=0,  MeWb_op=0; //
  reg  [31:0] IfId_pc=0,  IdEx_pc=0,  ExMe_pc=0,  MeWb_pc=0; //
  reg   [4:0] IfId_rd2=0, IdEx_rd2=0, ExMe_rd2=0, MeWb_rd2=0;//
  reg         IfId_w=0,   IdEx_w=0,   ExMe_w=0,   MeWb_w=0;  //
  reg         IfId_we=0,  IdEx_we=0,  ExMe_we=0;             //
  wire [31:0] IfId_ir, MeWb_ldd;                             // note
  /*************************** IF stage *********************************/
  wire w_taken;
  wire [31:0] w_tpc;
  reg  [31:0] r_pc  = 0;
  wire [31:0] w_pc4 = r_pc + 4;
  m_memory m_imem (w_clk, r_pc[13:2], 0, 0, IfId_ir);
  always @(posedge w_clk) begin
    r_pc     <= #3 (w_rst | r_halt) ? 0 : (w_taken) ? w_tpc : w_pc4;
    IfId_pc  <= #3 r_pc;
    IfId_pc4 <= #3 w_pc4;
  end
  /*************************** ID stage *********************************/
  wire [31:0] w_rrs, w_rrt, w_rslt2;
  wire  [5:0] w_op    = IfId_ir[31:26];
  wire  [4:0] w_rs    = IfId_ir[25:21];
  wire  [4:0] w_rt    = IfId_ir[20:16];
  wire  [4:0] w_rd    = IfId_ir[15:11];
  wire  [4:0] w_rd2   = (w_op!=0) ? w_rt : w_rd;
  wire [15:0] w_imm   = IfId_ir[15:0];
  wire [31:0] w_imm32 = {{16{w_imm[15]}}, w_imm};
  wire [31:0] w_rrt2  = (w_op>6'h5) ? w_imm32 : w_rrt;
  assign      w_tpc   = IfId_pc4 + {w_imm32[29:0], 2'h0};
  assign      w_taken = (w_op==`BNE && w_rrs!=w_rrt);
  m_regfile m_regs (w_clk, w_rs, w_rt, MeWb_rd2, MeWb_w, w_rslt2, w_rrs, w_rrt);
  always @(posedge w_clk) begin
    IdEx_pc   <= #3 IfId_pc;
    IdEx_op   <= #3 w_op;
    IdEx_rd2  <= #3 w_rd2;
    IdEx_w    <= #3 (w_op==0 || (w_op>6'h5 && w_op<6'h28));
    IdEx_we   <= #3 (w_op>6'h27);
    IdEx_rrs  <= #3 w_rrs;
    IdEx_rrt  <= #3 w_rrt;
    IdEx_rrt2 <= #3 w_rrt2;
  end
  /*************************** EX stage *********************************/
  wire [31:0] #10 w_rslt = IdEx_rrs + IdEx_rrt2; // ALU
  always @(posedge w_clk) begin
    ExMe_pc   <= #3 IdEx_pc;
    ExMe_op   <= #3 IdEx_op;
    ExMe_rd2  <= #3 IdEx_rd2;
    ExMe_w    <= #3 IdEx_w;
    ExMe_we   <= #3 IdEx_we;
    ExMe_rslt <= #3 w_rslt;
    ExMe_rrt  <= #3 IdEx_rrt;
  end
  /*************************** MEM stage ********************************/
  m_memory m_dmem (w_clk, ExMe_rslt[13:2], ExMe_we, ExMe_rrt, MeWb_ldd);
  always @(posedge w_clk) begin
    MeWb_pc   <= #3 ExMe_pc;
    MeWb_rslt <= #3 ExMe_rslt;
    MeWb_op   <= #3 ExMe_op;
    MeWb_rd2  <= #3 ExMe_rd2;
    MeWb_w    <= #3 ExMe_w;
  end
  /*************************** WB stage *********************************/
  assign w_rslt2 = (MeWb_op>6'h19 && MeWb_op<6'h28) ? MeWb_ldd : MeWb_rslt;
  /**********************************************************************/
  initial r_halt = 0;
  always @(posedge w_clk) if (MeWb_op==`HALT) r_halt <= 1;
  initial r_rout = 0;
  reg [31:0] r_tmp=0;
  always @(posedge w_clk) r_tmp <= (w_rst) ? 0 : (w_rs==30) ? w_rrs : r_tmp;
  always @(posedge w_clk) r_rout <= r_tmp;
endmodule
```

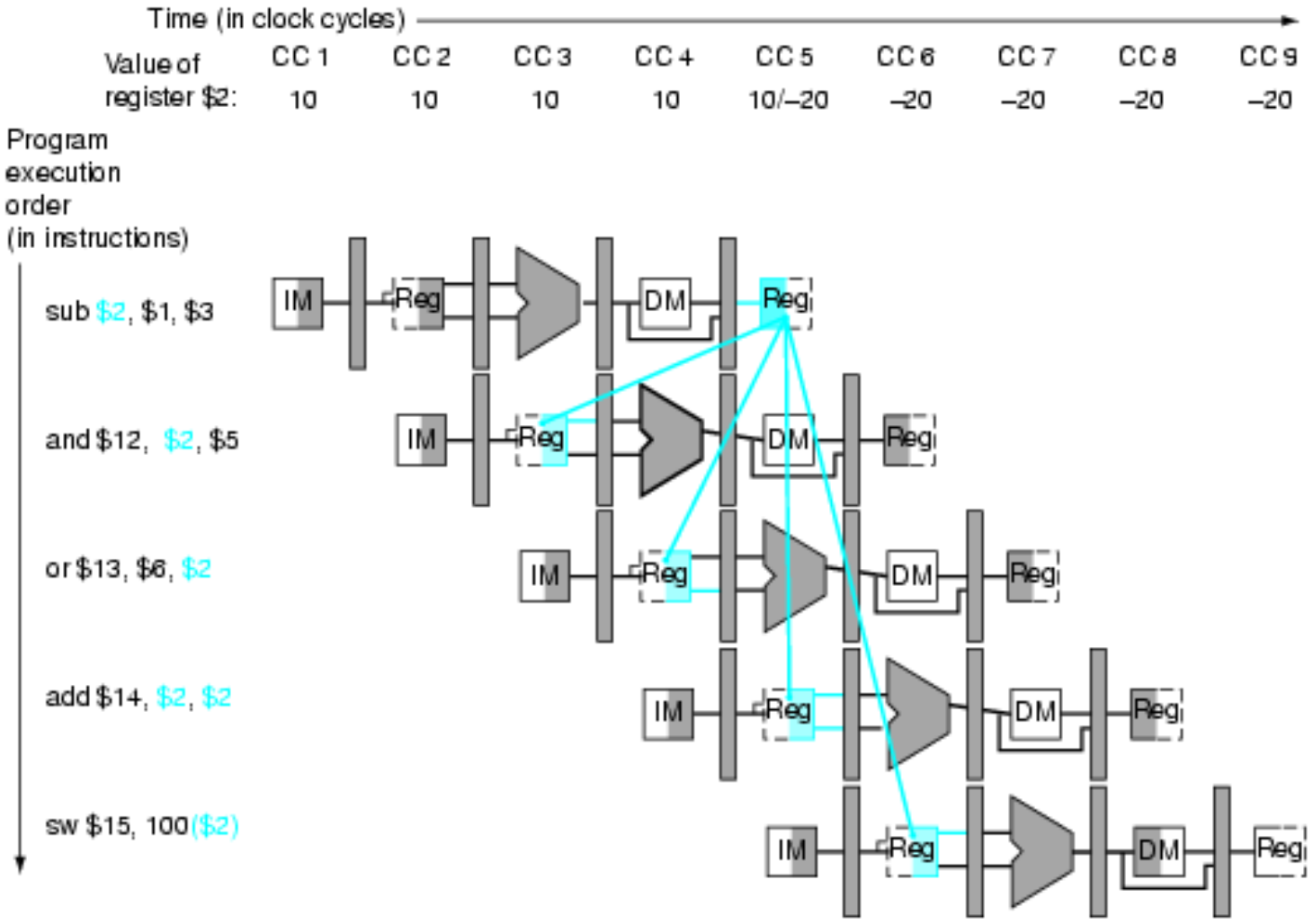
Hazards make pipelining hard

- 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
  - 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

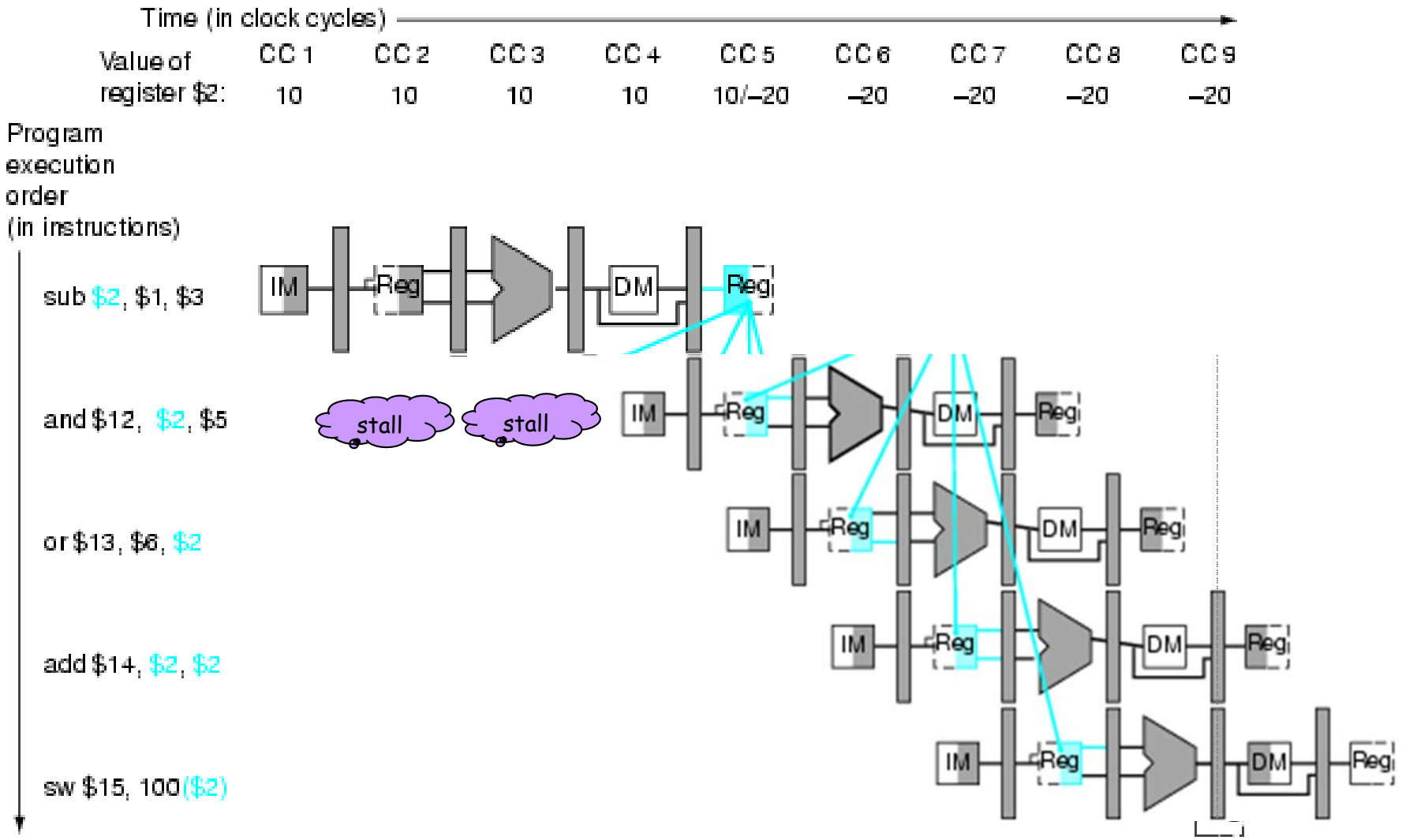
- まずは，データ・ハザードと制御ハザードが無いものとしてパイプラインプロセッサを構築する．その後，これらに対応するための修正を加える．

Data Hazard

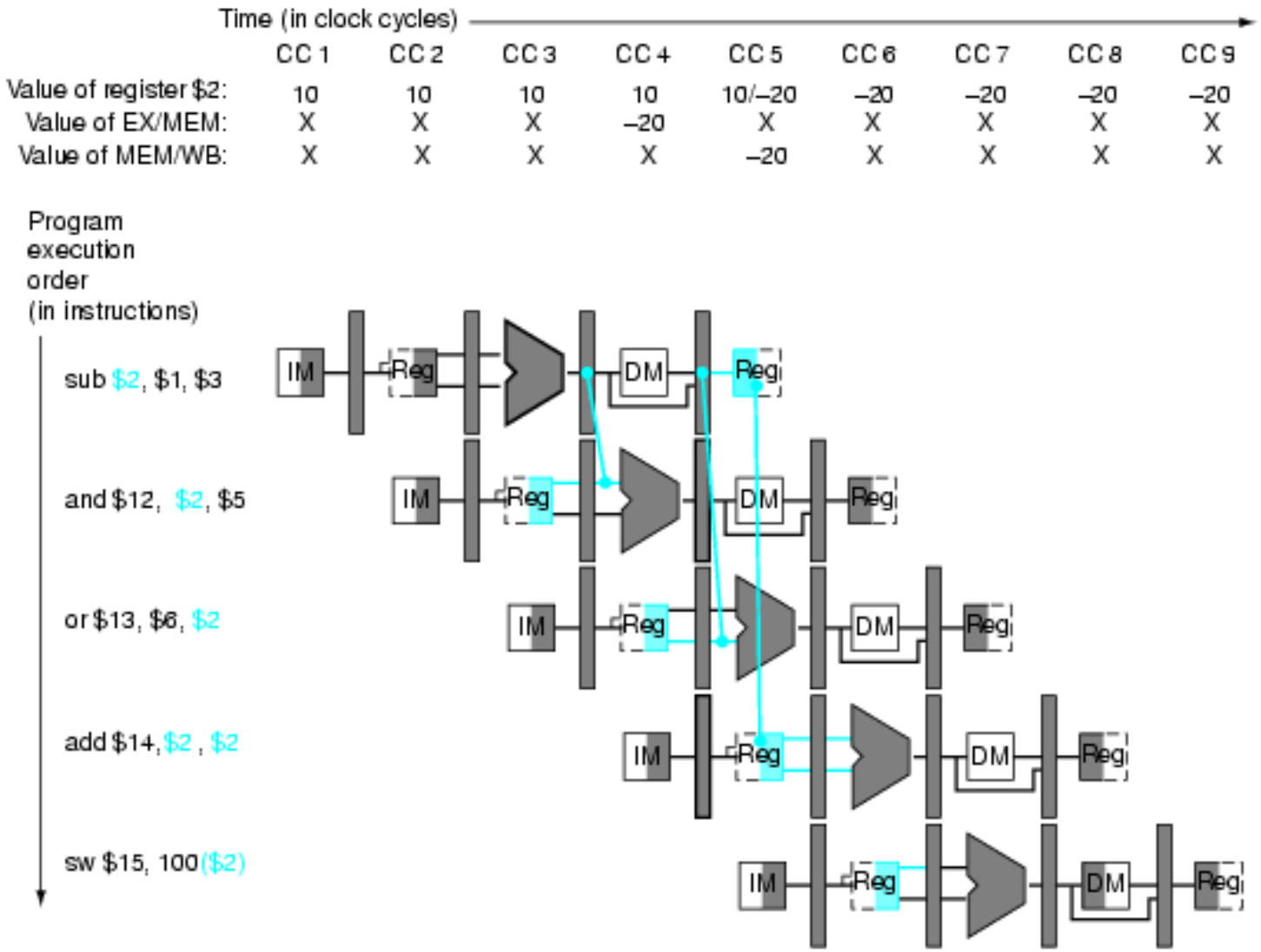
- sub命令が生成した$2を後続の命令が利用する場合にデータの受け渡しの制約が生じる.



Data Hazard and Stall



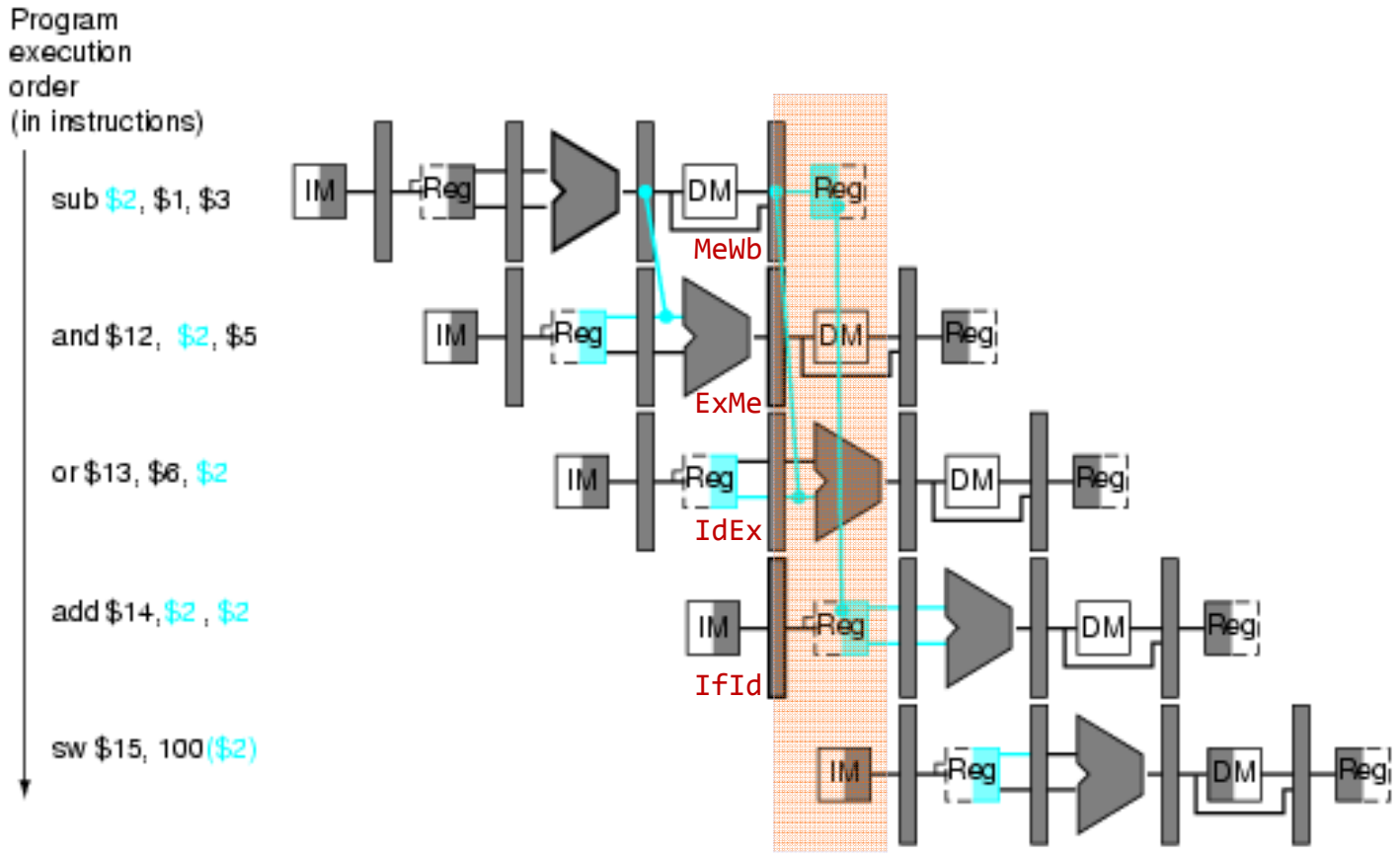
フォワーディングによるデータハザードの回避



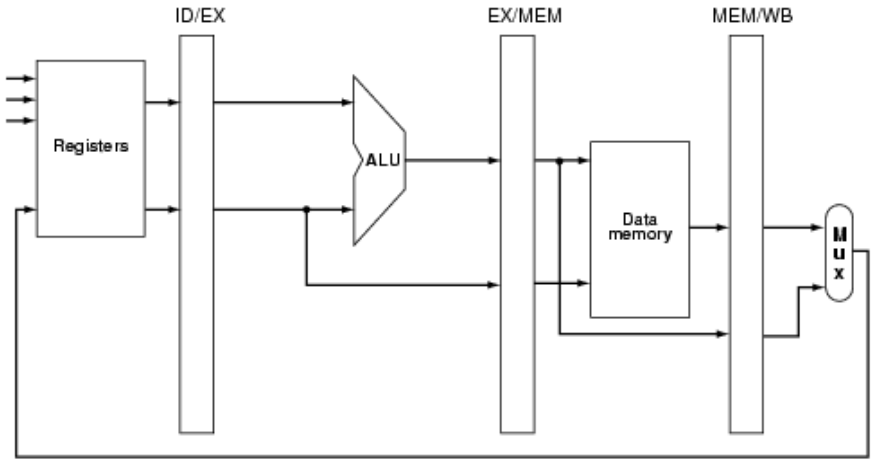
フォワーディングによるデータハザードの回避

Time (in clock cycles)

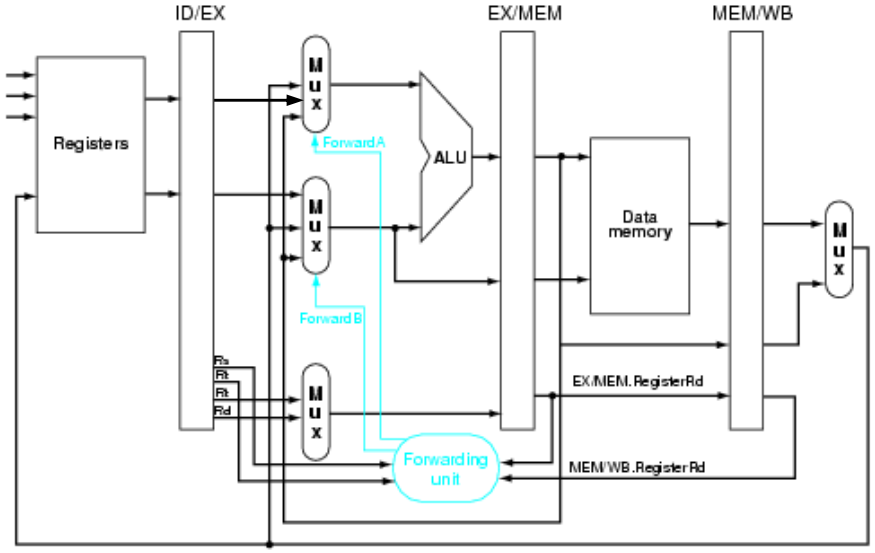
	CC 1	CC 2	CC 3	CC 4	CC 5	CC 6	CC 7	CC 8	CC 9
Value of register $2:	10	10	10	10	10/–20	–20	–20	–20	–20
Value of EX/MEM:	X	X	X	–20	X	X	X	X	X
Value of MEM/WB:	X	X	X	X	–20	X	X	X	X



フォワーディングのための変更点

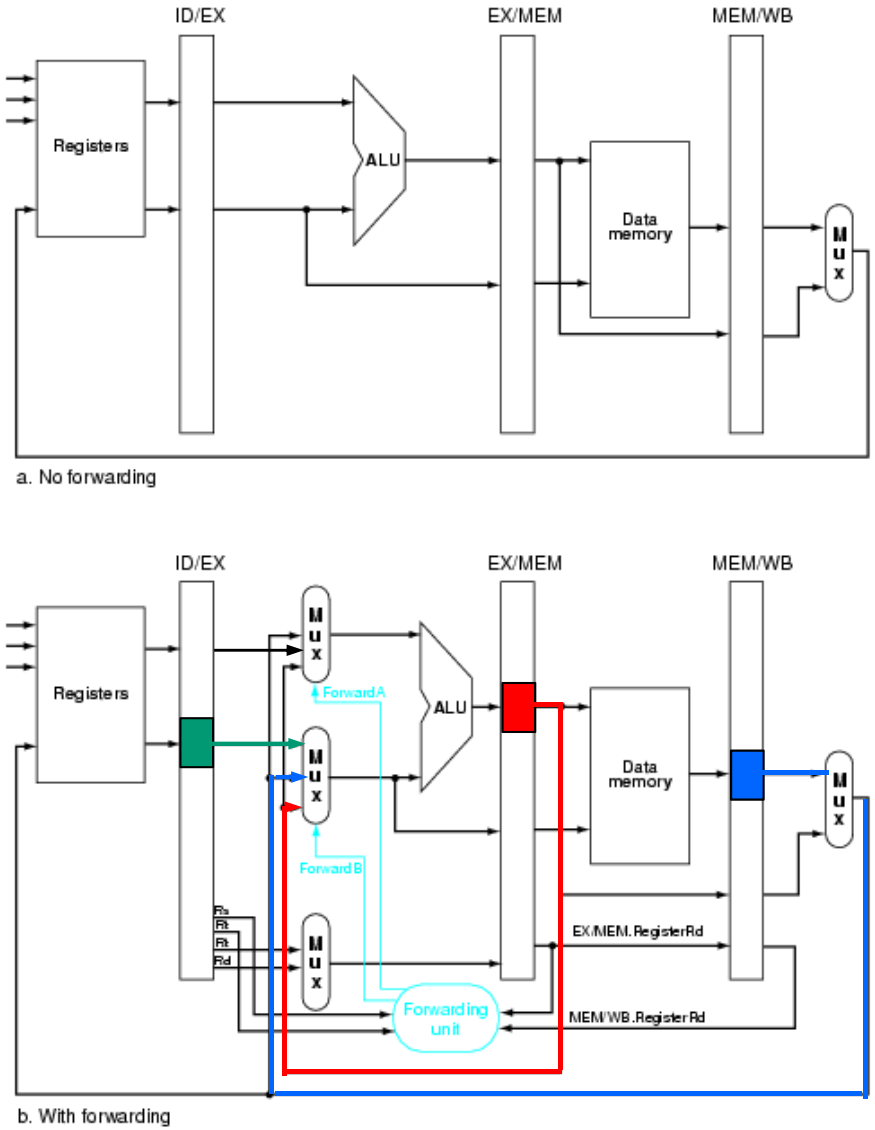


a. No forwarding

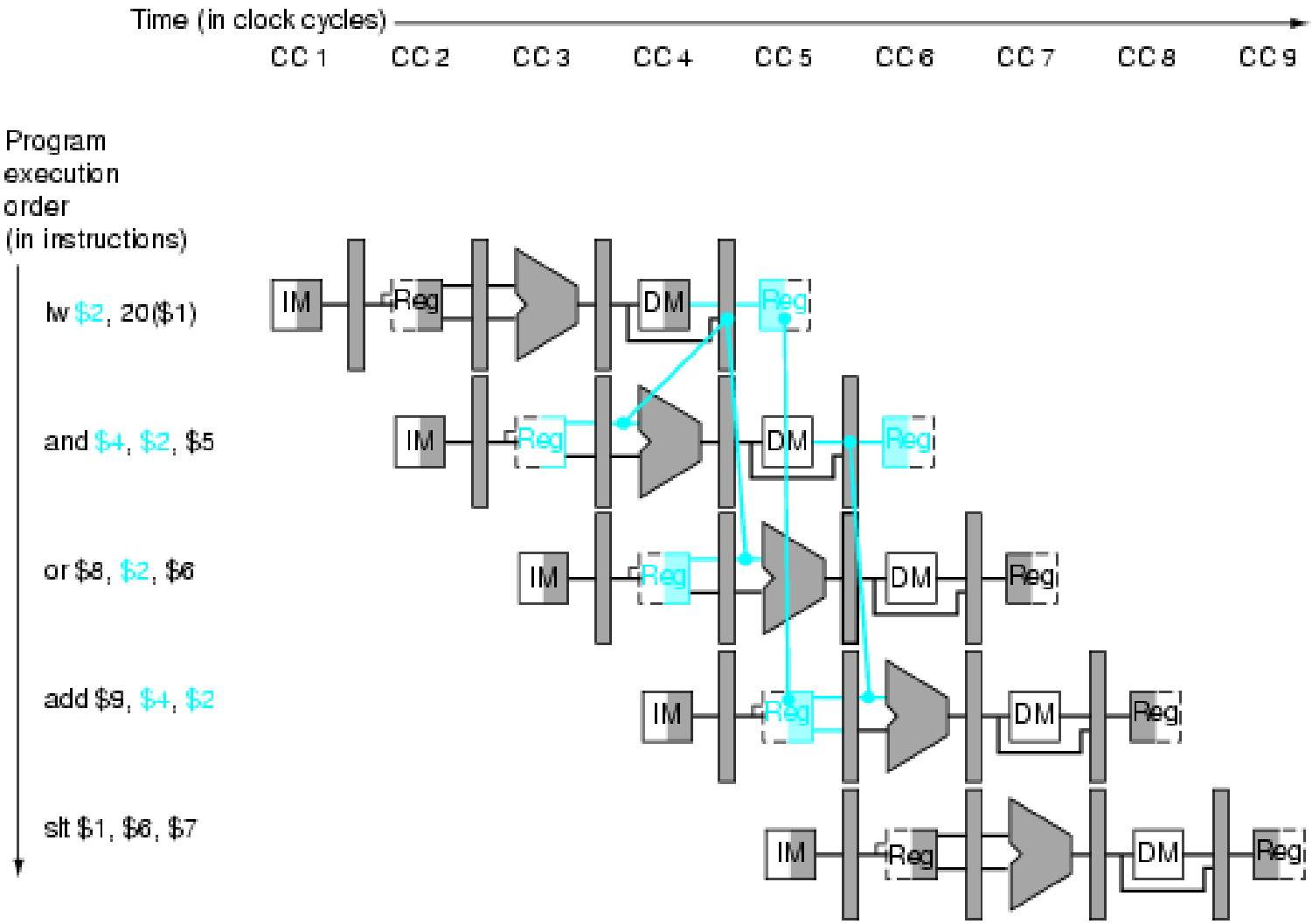


b. With forwarding

フォワーディングのための変更点



データハザードによって生じるストール



MIPS Program for the contest (program1.txt)

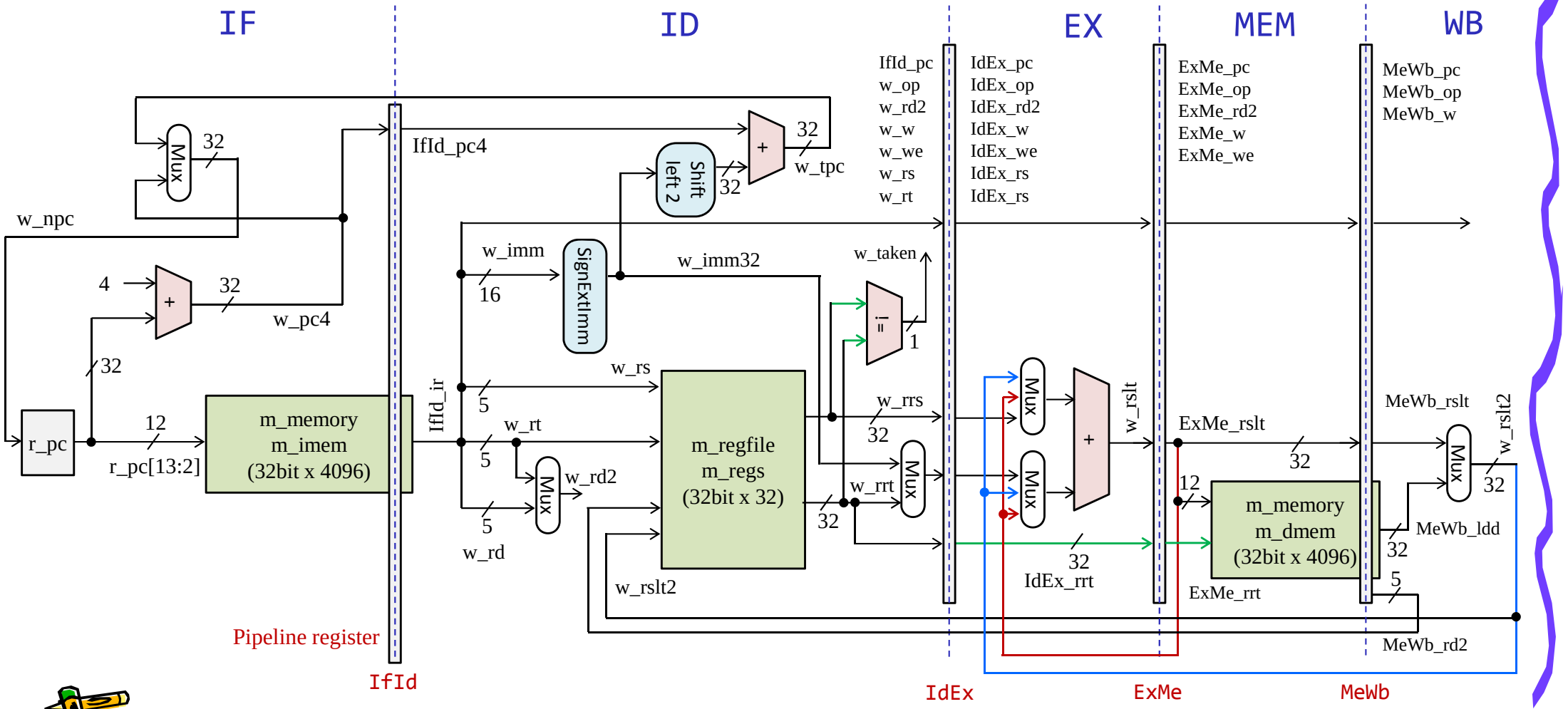
- result : 0x5ffa000

```c
#include <stdio.h>

main()
{
    int mem[4096];
    int i=0, j=0, sum = 0;

    for(j=0; j<3; j++){
        for(i=0; i<4096; i++) {
            mem[i] = i*4;
        }
        for(i=0; i<4096; i++) {
            sum += mem[i];
        }
    }
    printf("%d %x¥n", sum, sum);
}
```

```
initial begin
  cm_ram[0] ={`NOP};                          //     nop
  cm_ram[1] ={`NOP};                          //     nop
  cm_ram[2] ={`ADDI, 5'd0, 5'd20,16'd0};      //     addi $20, $0, 0
  cm_ram[3] ={`ADDI, 5'd0, 5'd21,16'd3};      //     addi $21, $0, 3
  cm_ram[4] ={`ADD,  5'd0, 5'd0, 5'd12,11'h20}; //     addi $12,$0, $0   // sum = 0;

  cm_ram[5] ={`ADDI, 5'd0, 5'd11,16'h0};      // L03 addi $11,$0, 0
  cm_ram[6] ={`ADDI, 5'd0, 5'd8, 16'd4096};   //     addi $8, $0, 4096
  cm_ram[7] ={`ADDI, 5'd0, 5'd9, 16'h0};      //     addi $9, $0, 0
  cm_ram[8] ={`ADDI, 5'd0, 5'd10,16'h0};      //     addi $10,$0, 0

  cm_ram[9] ={`SW,   5'd10,5'd11,16'd0};      // L01:sw   $11,0($10)
  cm_ram[10]={`ADDI, 5'd9, 5'd9, 16'h1};      //     addi $9, $9, 1
  cm_ram[11]={`ADDI, 5'd11,5'd11,16'h1};      //     addi $11,$11,1
  cm_ram[12]={`ADDI, 5'd11,5'd11,16'h1};      //     addi $11,$11,1
  cm_ram[13]={`ADDI, 5'd11,5'd11,16'h1};      //     addi $11,$11,1
  cm_ram[14]={`ADDI, 5'd11,5'd11,16'h1};      //     addi $11,$11,1
  cm_ram[15]={`ADDI, 5'd10,5'd10,16'h4};      //     addi $10,$10,4
  cm_ram[16]={`BNE,  5'd8, 5'd9, 16'hfff8};   //     bne  $8, $9, L01
  cm_ram[17]={`NOP};                          //     nop

  cm_ram[18]={`ADDI, 5'd0, 5'd8, 16'd4096};   //     addi $8, $0, 4096
  cm_ram[19]={`ADDI, 5'd0, 5'd9, 16'h0};      //     addi $9, $0, 0
  cm_ram[20]={`ADDI, 5'd0, 5'd10,16'h0};      //     addi $10,$0, 0

  cm_ram[21]={`LW,   5'd10,5'd11,16'd0};      // L02:lw   $11,0($10)
  cm_ram[22]={`ADDI, 5'd9, 5'd9, 16'h1};      //     addi $9, $9, 1
  cm_ram[23]={`ADDI, 5'd10,5'd10,16'h4};      //     addi $10,$10,4
  cm_ram[24]={`ADD,  5'd12,5'd11,5'd12,11'h20}; //     add  $12,$12,$11  // sum += $11
  cm_ram[25]={`ADDI, 5'd12,5'd12,16'h1};      //     addi $12,$12,1    // sum ++;
  cm_ram[26]={`ADDI, 5'd12,5'd12,16'hffff};   //     addi $12,$12,-1   // sum --;
  cm_ram[27]={`ADDI, 5'd12,5'd12,16'h1};      //     addi $12,$12,1    // sum ++;
  cm_ram[28]={`ADDI, 5'd12,5'd12,16'h1};      //     addi $12,$12,1    // sum ++;
  cm_ram[29]={`ADDI, 5'd12,5'd12,16'hffff};   //     addi $12,$12,-1   // sum --;
  cm_ram[30]={`ADDI, 5'd12,5'd12,16'h1};      //     addi $12,$12,1    // sum ++;
  cm_ram[31]={`ADDI, 5'd12,5'd12,16'hfffe};   //     addi $12,$12,-2   // sum -= 2;
  cm_ram[32]={`BNE,  5'd8, 5'd9, 16'hfff4};   //     bne  $8, $9, L02
  cm_ram[33]={`NOP};                          //     nop

  cm_ram[34]={`ADDI, 5'd20,5'd20,16'h1};      //     addi $20,$20,1    // j++
  cm_ram[35]={`ADDI, 5'd8, 5'd8,16'h11};      //     addi $8, $8, 0x11 //
  cm_ram[36]={`ADDI, 5'd8, 5'd8,16'h12};      //     addi $8, $8, 0x12 //
  cm_ram[37]={`ADDI, 5'd8, 5'd8,16'h13};      //     addi $8, $8, 0x13 //
  cm_ram[38]={`ADDI, 5'd8, 5'd8,16'h14};      //     addi $8, $8, 0x14 //
  cm_ram[39]={`BNE,  5'd20,5'd21,16'hffdd};   //     bne  $20,$21,L03  // (j<4) ?
  cm_ram[40]={`NOP};                          //     nop
  cm_ram[41]={`ADD,  5'd12,5'd0, 5'd30,11'h20}; //     add  $30,$12,$0
  cm_ram[42]={`ADD,  5'd30,5'd0, 5'd0, 11'h20}; //     add  $0, $30,$0 // 5ffa000
  cm_ram[43]={`HALT, 26'h0};                  //     halt
  cm_ram[44]={`NOP};                          //     nop
  cm_ram[45]={`NOP};                          //     nop
  cm_ram[46]={`NOP};                          //     nop
  cm_ram[47]={`NOP};                          //     nop
  cm_ram[48]={`NOP};                          //     nop
end
```

Inside module m_proc12 (pipelined processor)

- add, addi, lw, sw, bne, halt命令に対応したパイプライン版
- 図では省略しているが，分岐のための比較，データメモリの入力データにもフォワーディングが必要



Syllabus (1/3)

講義の概要とねらい

本講義では、「論理回路理論」の講義で習得した知識をベースに、より実用的なディジタル回路について学ぶ。また、簡単なコンピュータを例題として、コンピュータの基本原理とその論理設計の方法を学習する。
演習では、学んだ組合せ回路と順序回路をVerilog HDL等のハードウェア記述言語で記述し、シミュレーションによる回路の動作検証、FPGAが搭載されたハードウェアボード等に実装して動作確認をおこなう。

到達目標

本講義を履修することによって以下を習得する。
・コンピュータシステムの基本構成
・シングルサイクルプロセッサの論理設計に関する知識
・パイプライン処理をおこなうプロセッサの論理設計に関する知識
・ハードウェア記述言語を用いたシンプルなコンピュータシステムの設計能力

キーワード

コンピュータ，命令セットアーキテクチャ，プロセッサ、パイプライン処理，ハードウェア記述言語，Verilog HDL、FPGA

学生が身につける力

国際的教養力	コミュニケーション力	専門力	課題設定力	実践力または解決力
-	-	✓	-	✓

授業の進め方

原則として、90分×2コマの講義の後、90分×1コマのFPGAボードを用いた演習をおこないます。

Syllabus (3/3)

教科書

デイビッド・A. パターソン、ジョン・L. ヘネシー (著)、成田光彰 (翻訳)『コンピュータの構成と設計 第5版 上/下』日経BP社

参考書、講義資料等

無し。

成績評価の基準及び方法

講義で扱うコンピュータ論理設計に関する理解、ハードウェア記述言語を用いたコンピュータシステム実装への応用力を評価する。演習（30%）と期末試験（70%）により評価する。

関連する科目

CSC.T252 ： 論理回路理論
CSC.T262 ： アセンブリ言語
CSC.T372 ： コンパイラ構成
CSC.T363 ： コンピュータアーキテクチャ
CSC.T433 ： 先端コンピュータアーキテクチャ

履修の条件(知識・技能・履修済科目等)

履修条件は特に設けないが、関連する科目の論理回路理論を履修していることが望ましい。

連絡先（メール、電話番号）　※"[at]"を"@"(半角)に変換してください。

吉瀬謙二: kise[at]c.titech.ac.jp

オフィスアワー

メールで事前予約すること。

アナウンス

- 各グループの担当者は，プロセッサ設計コンテストのスライドを2018年5月29日（火）の深夜までに提出すること．
- 2018年5月31日(木)にプロセッサ設計コンテストの発表会を実施します．
  9:00 までに情報工学系の計算機室に集まって下さい．
  9:00 – 11:30 の実施を予定しています．

- 期末試験は2018年6月7日（木） 10:45 – 12:15 W621 です．
  - 試験では MIPS Reference Card を配布します．この内容を覚える必要はありません．
  - 鉛筆や消しゴムなどの筆記用具のみ持ち込み可とします．
  - ただし，自筆のA4サイズ 1枚の紙のみは参照してよいものとします．