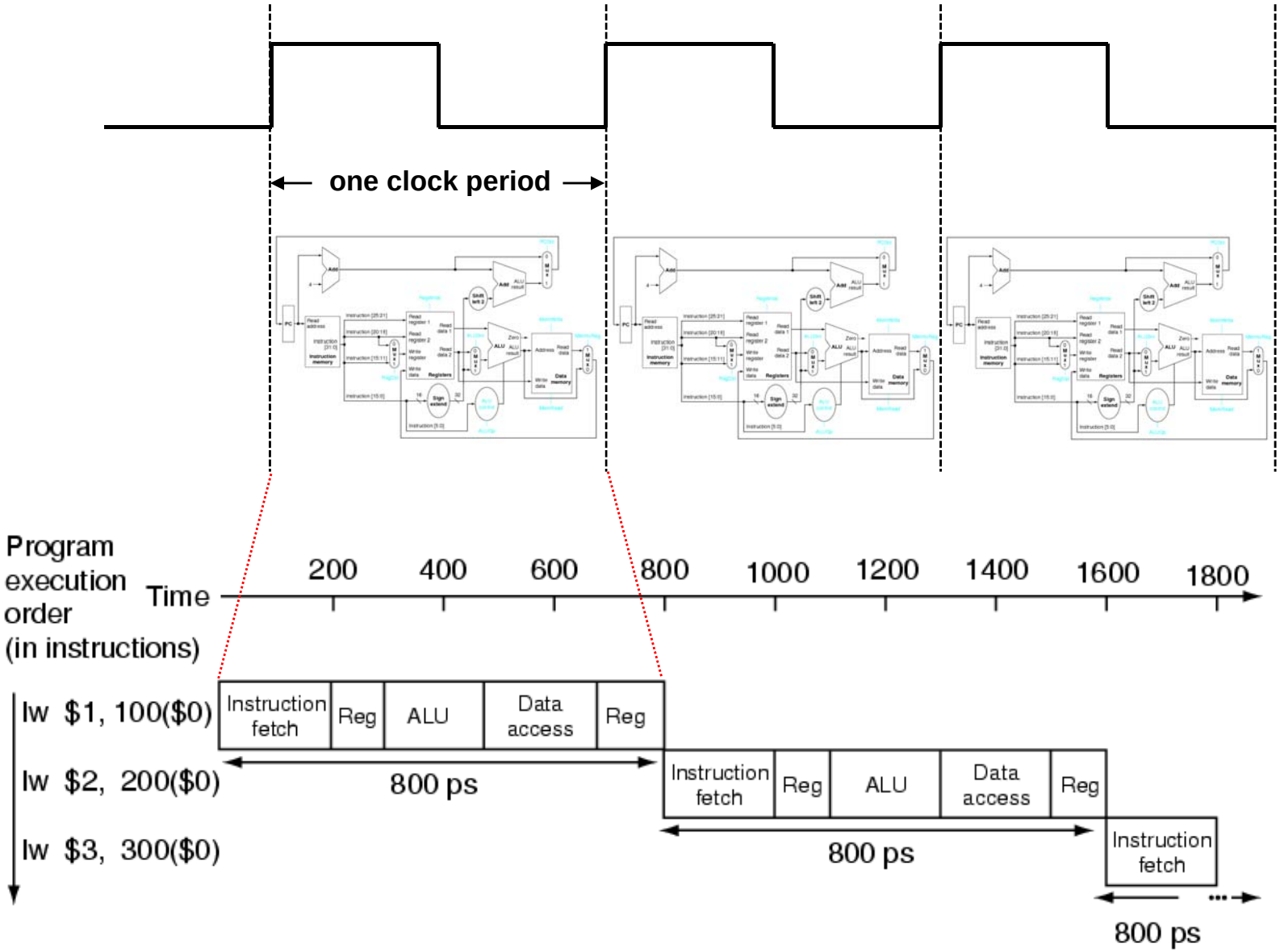
Course number: CSC.T341

コンピュータ論理設計
Computer Logic Design

12. シングルサイクルプロセッサの制御とマルチサイクルのプロセッサ
Control for Single Cycle Processor and Multi-cycle Processor

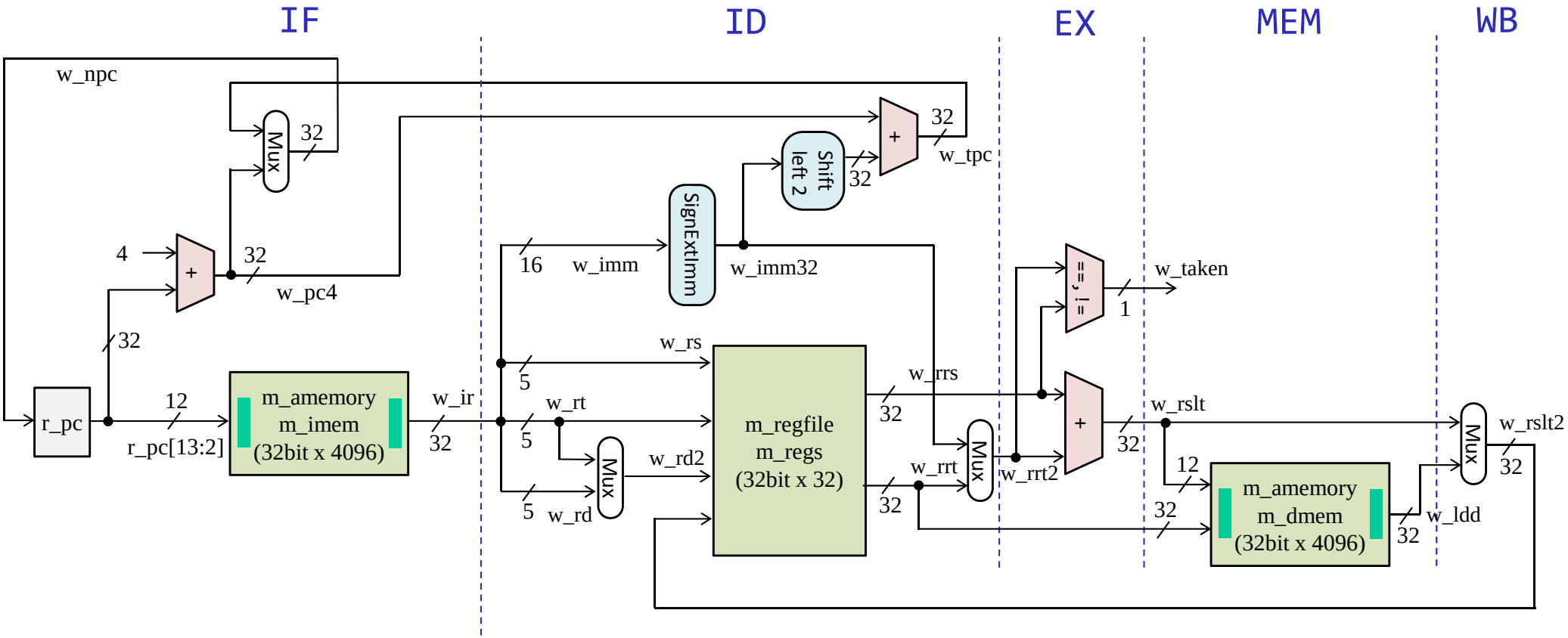
吉瀬 謙二　情報工学系
Kenji Kise, Department of Computer Science
kise _at_ c.titech.ac.jp　www.arch.cs.titech.ac.jp/lecture/CLD/
W621 講義室　月 10:45-12:15, 木 9:00-12:15

Single Cycle Processor (our baseline processor)



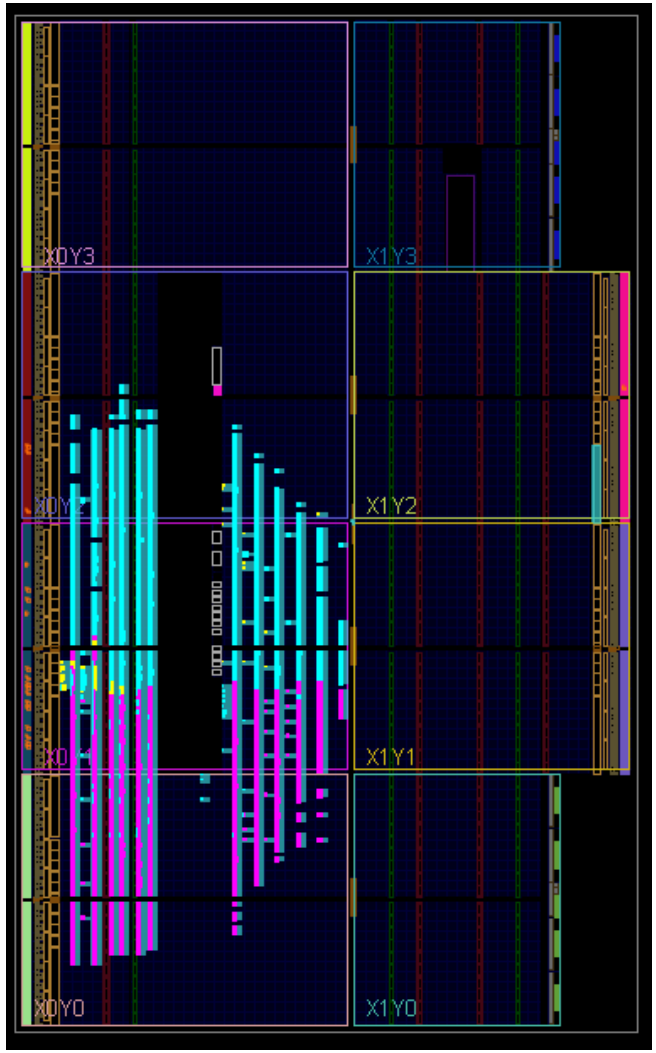
Inside module m_proc07 (single cycle, about 50MHz)

- 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, lw, sw, beq, bne, halt命令に対応したシングルサイクルのプロセッサ



OP	rs	rt	rd	sa	funct	R format
OP	rs	rt	immediate			I format

Inside module m_proc07 (control for single cycle p)

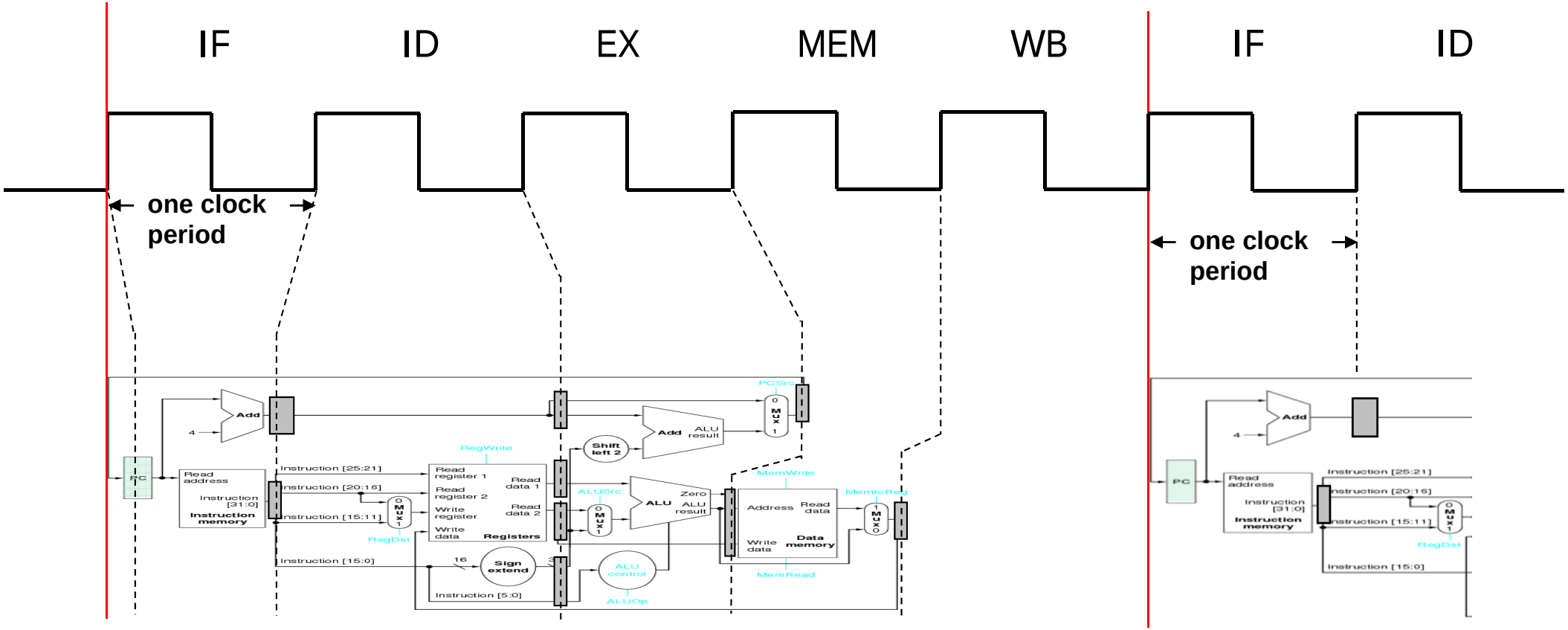


青色は命令メモリ，赤色はデータメモリ

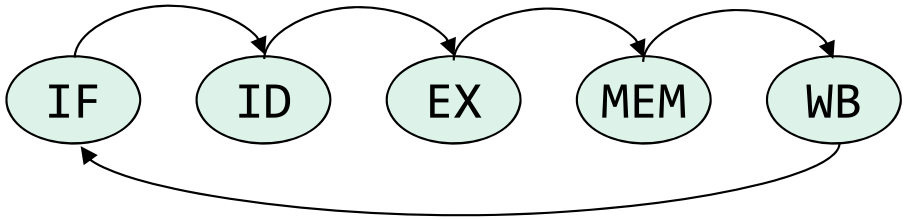
code110.v

```
module m_proc07 (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] r_pc = 0;
  wire [31:0] w_ir, w_rrs, w_rrt, w_imm32, w_rrt2, w_rslt, w_ldd, w_rslt2;
  wire  [5:0] w_op = w_ir[31:26];
  wire  [4:0] w_rs = w_ir[25:21];
  wire  [4:0] w_rt = w_ir[20:16];
  wire  [4:0] w_rd = w_ir[15:11];
  wire        w_taken = ((w_op==`BEQ && w_rrs==w_rrt2) ||
                         (w_op==`BNE && w_rrs!=w_rrt2));
  wire [31:0] w_npc = r_pc + 4;
  wire [31:0] w_tpc = w_npc + {w_imm32[29:0], 2'h0};
  always @(posedge w_clk) r_pc <= #3 (w_rst | r_halt) ? 0 : (w_taken) ? w_tpc : w_npc;
  m_aimemory m_imem (w_clk, r_pc[13:2], 0, 0, w_ir);

  wire  [4:0] w_rd2 = (w_op!=0) ? w_rt : w_rd;
  wire [15:0] w_imm = w_ir[15:0];
  wire        w_w = (w_op==0 || (w_op>6'h5 && w_op<6'h28)); // ADD, ADDI, LW
  m_regfile m_regs (w_clk, w_rs, w_rt, w_rd2, w_w, w_rslt2, w_rrs, w_rrt);

  assign w_imm32 = {{16{w_imm[15]}}, w_imm};
  assign w_rrt2  = (w_op>6'h5) ? w_imm32 : w_rrt; // ADDI, LW, SW

  assign #10 w_rslt = w_rrs + w_rrt2;

  wire        w_we = (w_op>6'h27); // SW
  m_amemory m_dmem (w_clk, w_rslt[13:2], w_we, w_rrt, w_ldd);
  assign w_rslt2 = (w_op>6'h19 && w_op<6'h28) ? w_ldd : w_rslt; // LW

  initial r_rout = 0;
  always @(posedge w_clk) r_rout <= (w_rst) ? 0 : (w_rs==30) ? w_rrs : r_rout;
  initial r_halt = 0;
  always @(posedge w_clk) if (w_op==`HALT) r_halt <= 1;
endmodule
```

Multi-cycle Processor



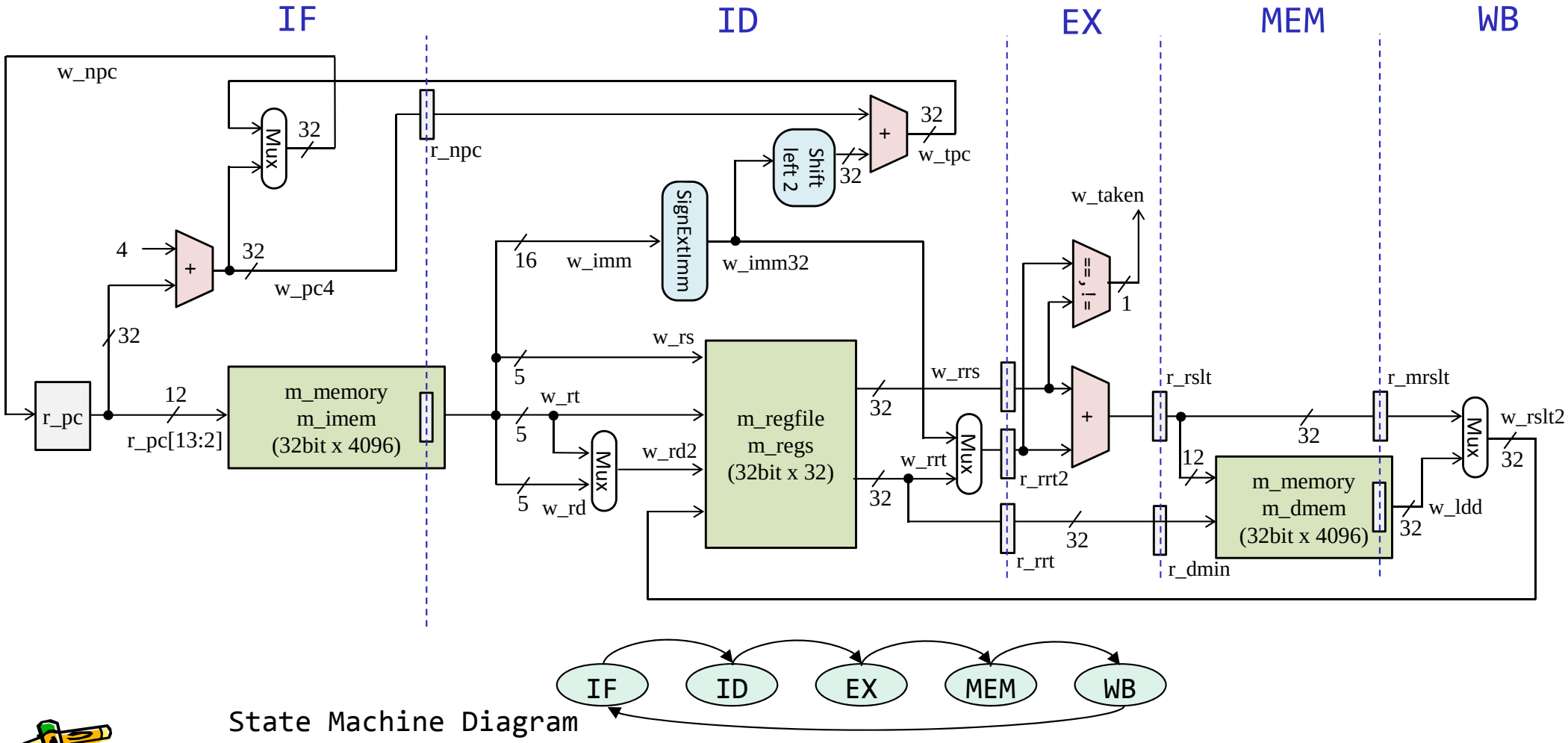
State Machine Diagram



0.1 x 3 + 0.1 x 5 + 0.8 x 4 = 4.0

Inside module m_proc08 (multi-cycle about 150MHz)

- add, addi, lw, sw, beq, bne, halt命令に対応したマルチサイクルのプロセッサ
- 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.



Inside module m_proc08

code120.v

```
module m_memory (w_clk, w_addr, w_we, w_din, r_dout);
  input  wire w_clk, w_we;
  input  wire [11:0] w_addr;
  input  wire [31:0] w_din;
  output reg  [31:0] r_dout;
  reg [31:0] cm_ram [0:4095]; // 4K word (4096 x 32bit) memory
  always @(posedge w_clk) begin
    r_dout <= cm_ram[w_addr];
    if (w_we) cm_ram[w_addr] <= w_din;
  end
  initial r_dout = 0;
  initial begin
    cm_ram[0] ={`NOP};                              //     nop
    cm_ram[1] ={`ADDI, 5'd0, 5'd8, 16'd4096};       //     addi $8, $0, 4096
    cm_ram[2] ={`ADDI, 5'd0, 5'd9, 16'h0};          //     addi $9, $0, 0
    cm_ram[3] ={`ADDI, 5'd0, 5'd10,16'h0};          //     addi $10,$0, 0
    cm_ram[4] ={`SW,   5'd10,5'd9, 16'd0};          // L01:sw   $9, 0($10)
    cm_ram[5] ={`ADDI, 5'd9, 5'd9, 16'h1};          //     addi $9, $9, 1
    cm_ram[6] ={`ADDI, 5'd10,5'd10,16'h4};          //     addi $10,$10,4
    cm_ram[7] ={`BNE,  5'd8, 5'd9, 16'hfffc};       //     bne  $8, $9, L01
    cm_ram[8] ={`NOP};                              //     nop
    cm_ram[9] ={`ADD,  5'd0, 5'd0, 5'd12,11'h20}; //     addi $12,$0, $0   // sum = 0;
    cm_ram[10]={`ADDI, 5'd0, 5'd8, 16'd4096};       //     addi $8, $0, 4096
    cm_ram[11]={`ADDI, 5'd0, 5'd9, 16'h0};          //     addi $9, $0, 0
    cm_ram[12]={`ADDI, 5'd0, 5'd10,16'h0};          //     addi $10,$0, 0
    cm_ram[13]={`LW,   5'd10,5'd11,16'd0};          // L02:lw   $11,0($10)
    cm_ram[14]={`ADDI, 5'd9, 5'd9, 16'h1};          //     addi $9, $9, 1
    cm_ram[15]={`ADDI, 5'd10,5'd10,16'h4};          //     addi $10,$10,4
    cm_ram[16]={`ADD,  5'd12,5'd11,5'd12,11'h20}; //     add  $12,$12,$11  // sum += $11
    cm_ram[17]={`BNE,  5'd8, 5'd9, 16'hfffb};       //     bne  $8, $9, L02
    cm_ram[18]={`NOP};                              //     nop
    cm_ram[19]={`ADD,  5'd12,5'd0, 5'd30,11'h20}; //     add  $30,$12,$0
    cm_ram[20]={`ADD,  5'd30,5'd0, 5'd0, 11'h20}; //     add  $0, $30,$0
    cm_ram[21]={`HALT, 26'h0};                      //     halt
    cm_ram[22]={`NOP};                              //     nop
    cm_ram[23]={`NOP};                              //     nop
    cm_ram[24]={`NOP};                              //     nop
  end
endmodule
```

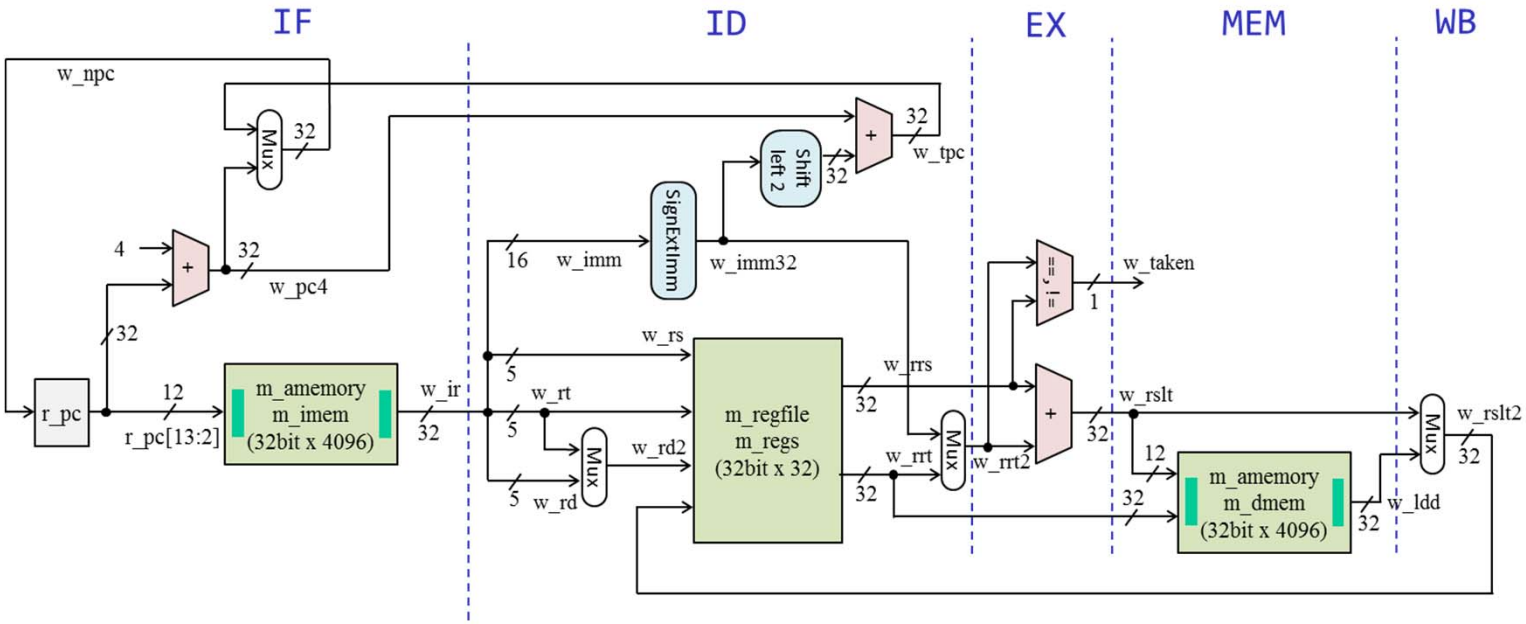
m_memory
m_imem
(32bit x 4096)

m_memory
m_dmem
(32bit x 4096)

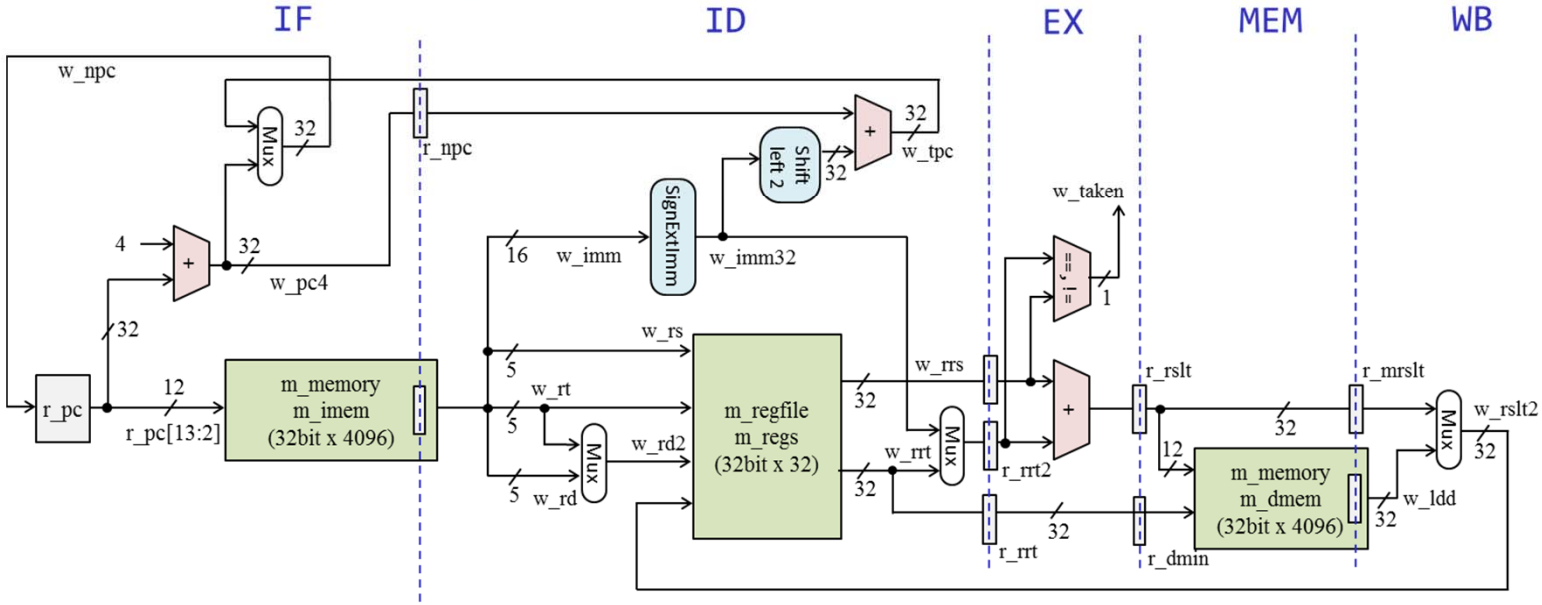
コンテストではこの赤い部分が変わる.
実行命令数の多いプログラムを用いる.

Compare the two processor architectures

m_proc07

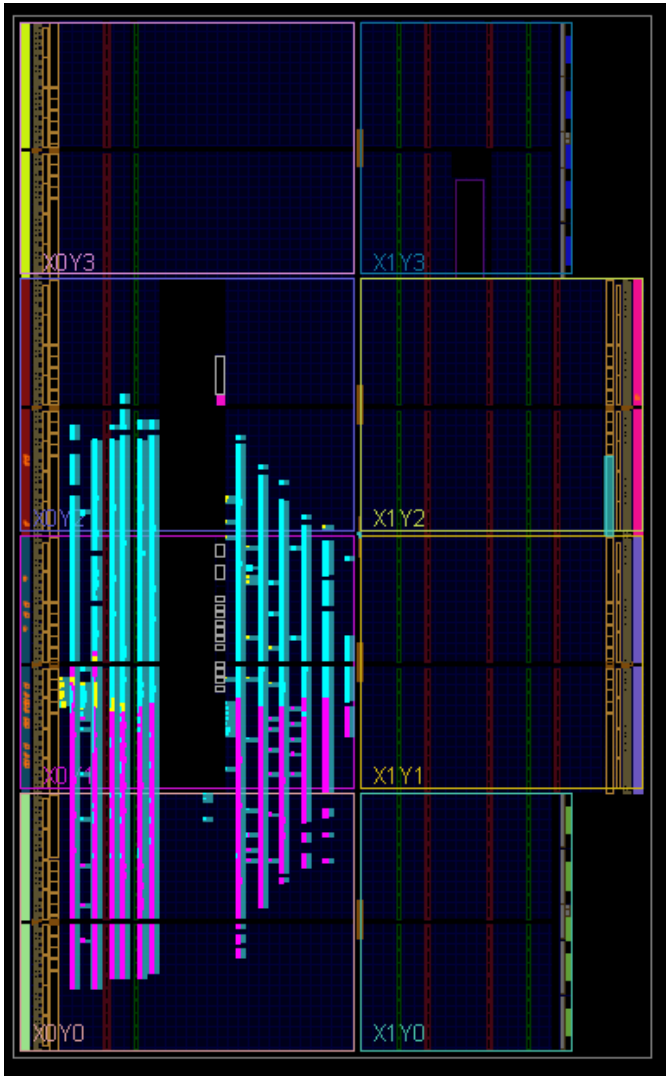


m_proc08

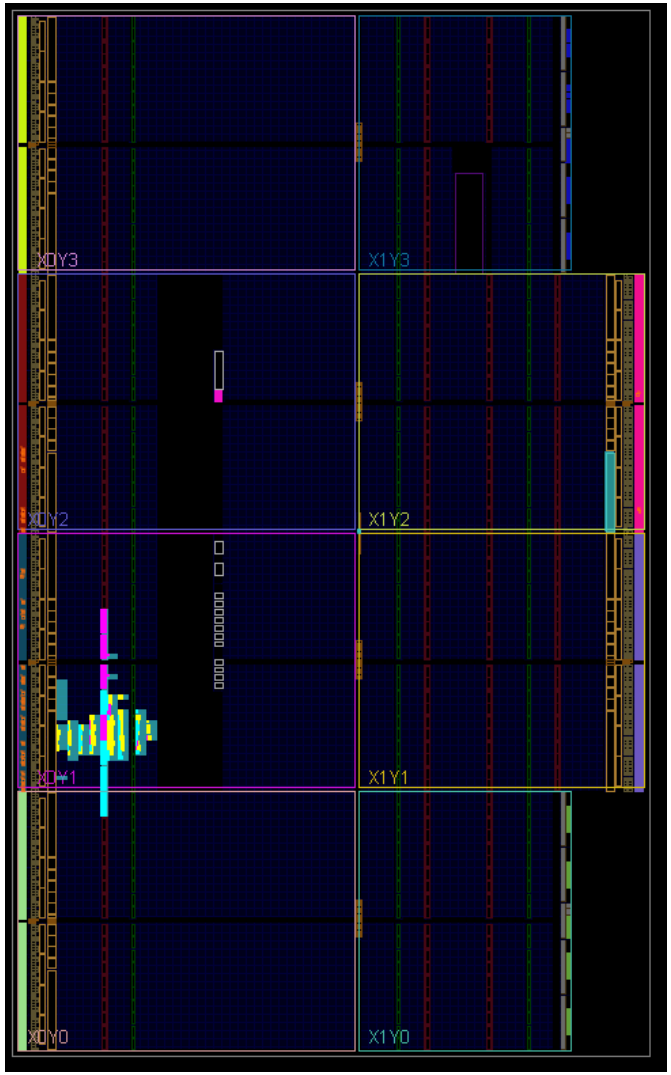


Implementation results

m_proc07



m_proc08



青色は命令メモリ，赤色はデータメモリ

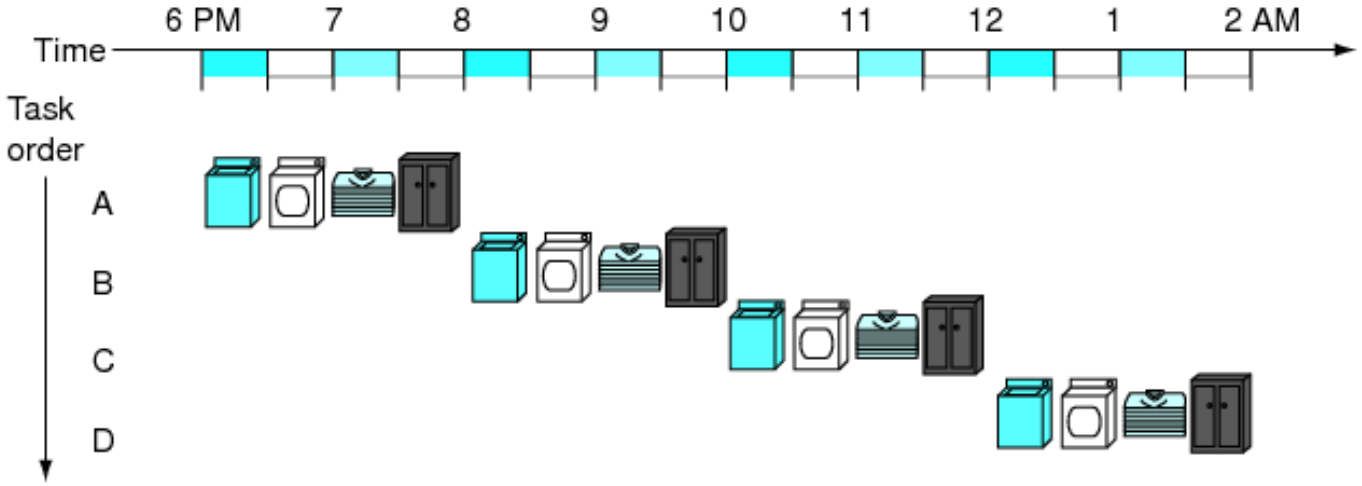
Inside module m_proc08 (multi-cycle about 150MHz)

code120.v

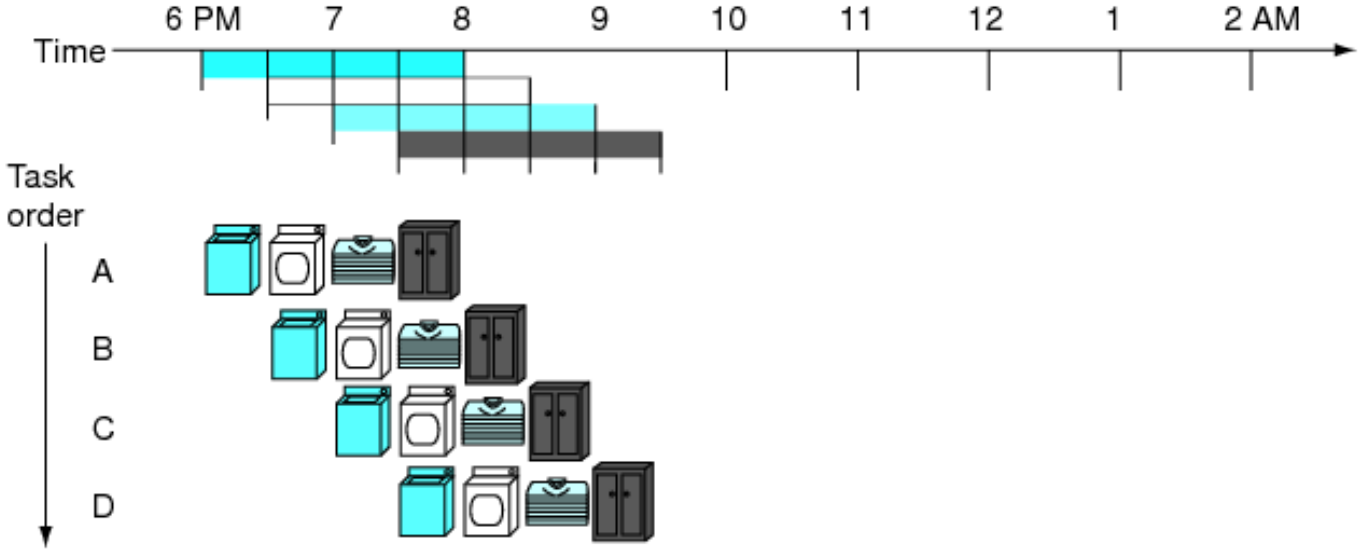
```
`define S_IF  1
`define S_ID  2
`define S_EX  3
`define S_MEM 4
`define S_WB  5

module m_proc08 (w_clk, w_rst, r_rout, r_halt);
  input  wire w_clk, w_rst;
  output reg [31:0] r_rout;
  output reg        r_halt;

  wire [31:0] w_ir, w_rrs, w_rrt, w_imm32, w_rrt2, w_rslt, w_ldd, w_rslt2;
  reg  [31:0] r_pc   = 0;
  reg  [31:0] r_rrs  = 0;
  reg  [31:0] r_rrt  = 0;
  reg  [31:0] r_rrt2 = 0;
  reg  [31:0] r_npc  = 0;
  reg   [2:0] state  = 0;
  always @(posedge w_clk) begin
    state <= #3 (w_rst | r_halt) ? 0 : (state==`S_WB) ? `S_IF : state + 1;
  end
  /**************************** IF  **********************************/
  wire  [5:0] w_op = w_ir[31:26];
  wire  [4:0] w_rs = w_ir[25:21];
  wire  [4:0] w_rt = w_ir[20:16];
  wire  [4:0] w_rd = w_ir[15:11];
  wire        w_taken = ((w_op==`BEQ && r_rrs==r_rrt2) ||
                         (w_op==`BNE && r_rrs!=r_rrt2));
  wire [31:0] w_npc = r_pc + 4;
  wire [31:0] w_tpc = r_npc + {w_imm32[29:0], 2'h0};
  always @(posedge w_clk) if(state==`S_WB) begin
    r_pc <= #3 (w_rst | r_halt) ? 0 : (w_taken) ? w_tpc : w_npc;
  end
  m_memory m_imem (w_clk, r_pc[13:2], 0, 0, w_ir);
  always @(posedge w_clk) if(state==`S_IF) begin
    r_npc <= #3 w_npc;
  end
 /**************************** ID  **********************************/
  wire  [4:0] w_rd2 = (w_op!=0) ? w_rt : w_rd;
  wire [15:0] w_imm = w_ir[15:0];
  wire        w_w = (w_op==0 || (w_op>6'h5 && w_op<6'h28));
  m_regfile m_regs (w_clk, w_rs, w_rt, w_rd2, r_w, w_rslt2, w_rrs, w_rrt);
  assign w_imm32 = {{16{w_imm[15]}}, w_imm};
  assign w_rrt2  = (w_op>6'h5) ? w_imm32 : w_rrt;
  always @(posedge w_clk) if(state==`S_ID) begin
    r_rrs  <= #3 w_rrs;
    r_rrt  <= #3 w_rrt;
    r_rrt2 <= #3 w_rrt2;
  end
  /**************************** EX  **********************************/
  wire       w_we = (w_op>6'h27);
  assign #10 w_rslt = r_rrs + r_rrt2;
  reg [31:0] r_rslt = 0;
  reg [31:0] r_dmin = 0;
  reg        r_we   = 0;
  always @(posedge w_clk) if(state==`S_EX) begin
    r_rslt <= #3 w_rslt;
    r_dmin <= #3 r_rrt;
  end
  always @(posedge w_clk) r_we <= #3 (state!=`S_EX) ? 0 : w_we;
  /**************************** MEM **********************************/
  m_memory m_dmem (w_clk, r_rslt[13:2], r_we, r_dmin, w_ldd);
  reg [31:0] r_mrslt = 0;
  always @(posedge w_clk) if(state==`S_MEM) begin
    r_mrslt <= #3 r_rslt;
  end
  /**************************** WB  **********************************/
  assign w_rslt2 = (w_op>6'h19 && w_op<6'h28) ? w_ldd : r_mrslt;
  reg r_w = 0;
  always @(posedge w_clk) begin
    r_w <= (state!=`S_MEM) ? 0 : (w_op==0 || (w_op>6'h5 && w_op<6'h28));
  end
  /*****************************************************************/
  initial r_halt = 0;
  always @(posedge w_clk) if (w_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_rout;
  always @(posedge w_clk) r_rout <= r_tmp;
endmodule
```

Pipelined Processor

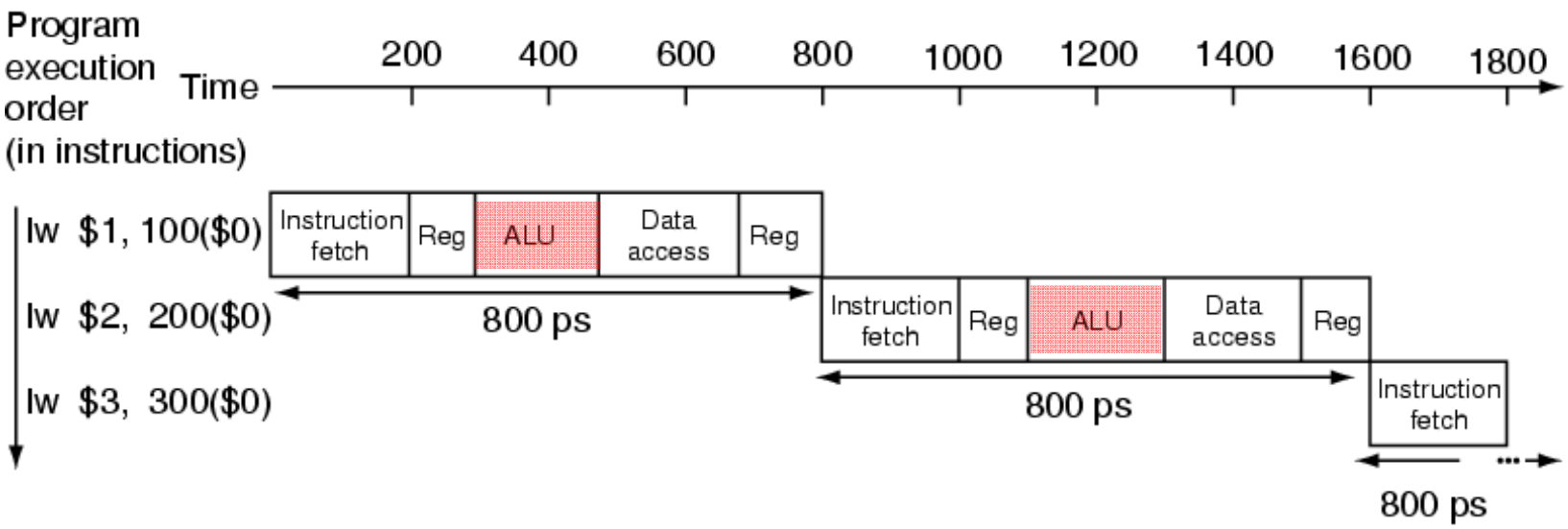
- Non pipelining (Multi-cycle)

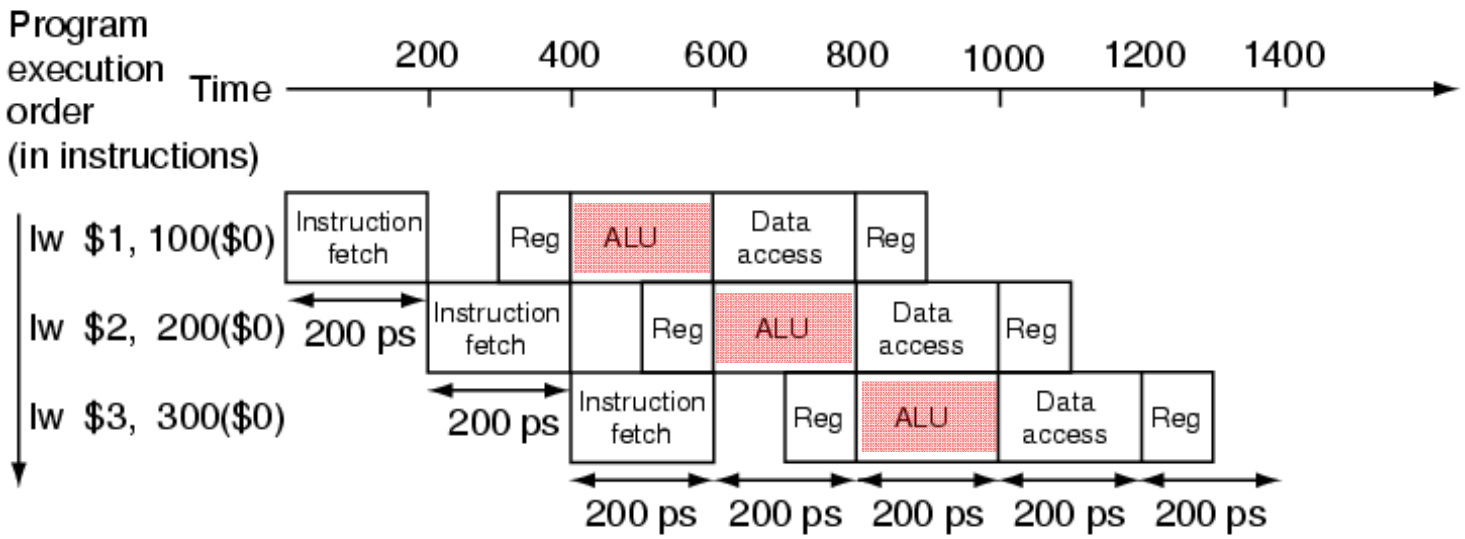


- Pipelining



Pipelined Processor





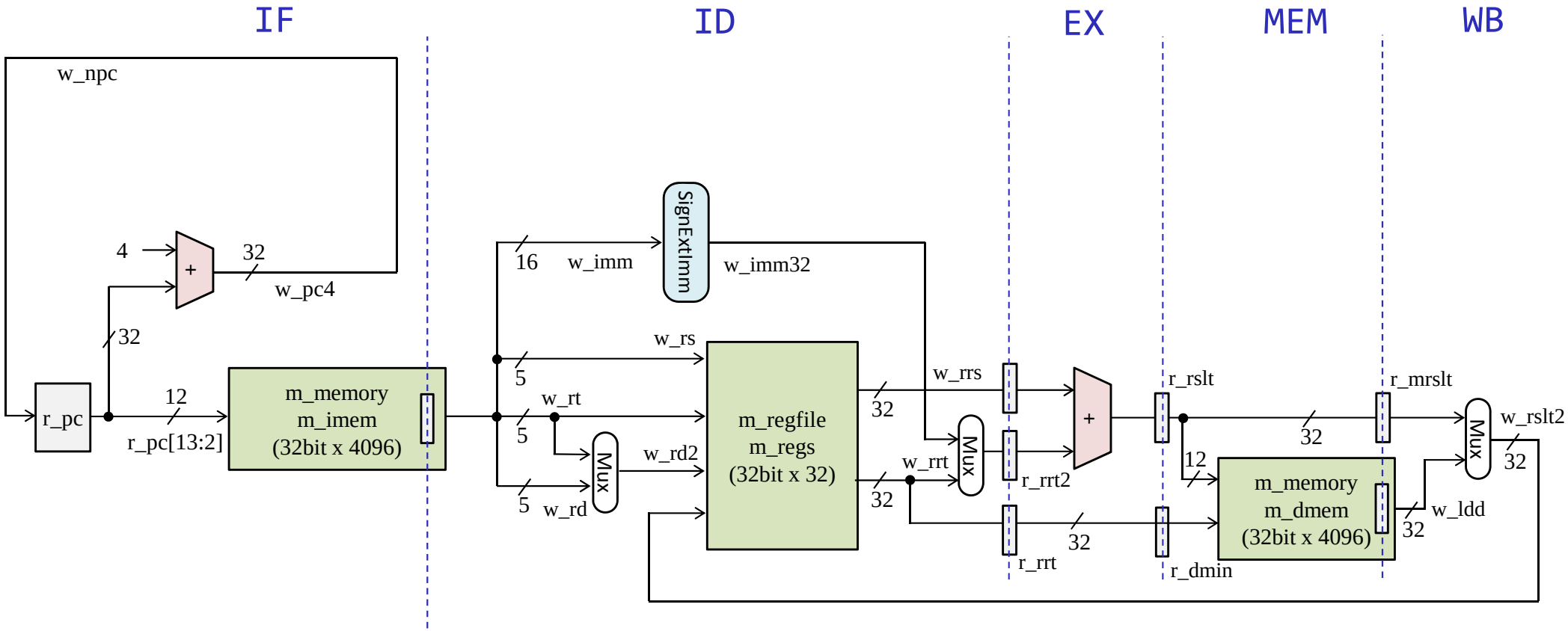
Hazards make pipelining hard

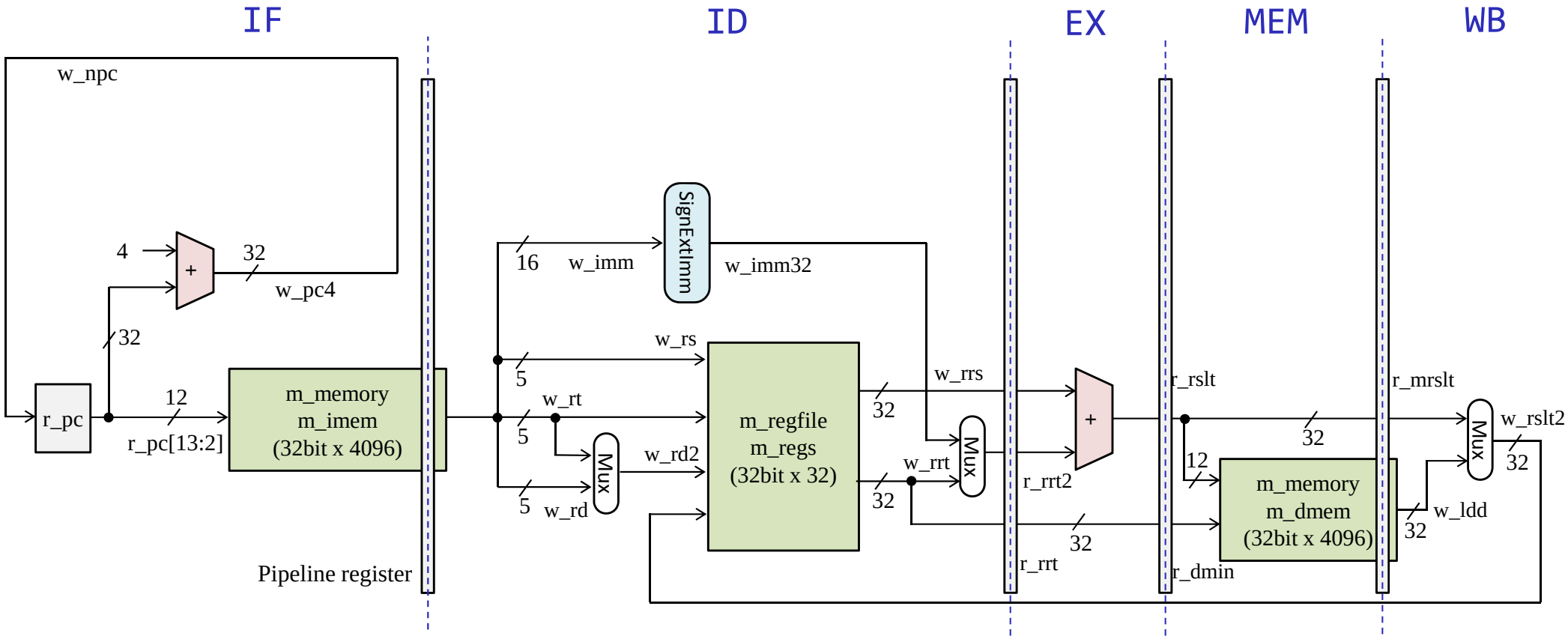
- 命令を適切なサイクルで実行できないような状況が存在する．これをハザード(hazard)と呼ぶ．
  - 構造ハザード (structural hazard)
    - オーバラップ実行する命令の組み合わせをハードウェアがサポートしていない場合．資源不足により生じる．
  - データ・ハザード(data hazard)
    - データの受け渡しの制約によって生じるハザード
  - 制御ハザード(control hazard)
    - 分岐命令，ジャンプ命令によって生じるハザード

- 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．

Inside module m_proc10 (pipelined processor)

- add, addi, lw, sw, 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処理のプロセッサ
- 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_proc10 (pipelined processor)

- add, addi, lw, sw, 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処理のプロセッサ
- 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.