Course number: CSC.T341

コンピュータ論理設計
Computer Logic Design

11. プロセッサ設計コンテストといくつかのプロセッサアーキテクチャ
Processor Design Contest and Some Processor Architectures

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

Exercise (6) : Group Meeting

- どのようなプロセッサを設計するか相談する.
- どのように発表スライドを作成するか相談する.
- 誰が発表するか相談する.
- グループのメンバーと簡単に情報交換できる手段を構築.
- プロセッサの設計と実装のスケジュールを確認する.
- 作業を開始する.
- 次の4枚のスライドは発表スライドのサンプル. その中に発表に関する注意点がまとめられている.

Course number: CSC.T341

Processor Design Contest (2018-05-31) Presentation Slide

設計したプロセッサの名前

適切なタイトル

XX: YYの高速化を用いたZZプロセッサ

Group No: NN

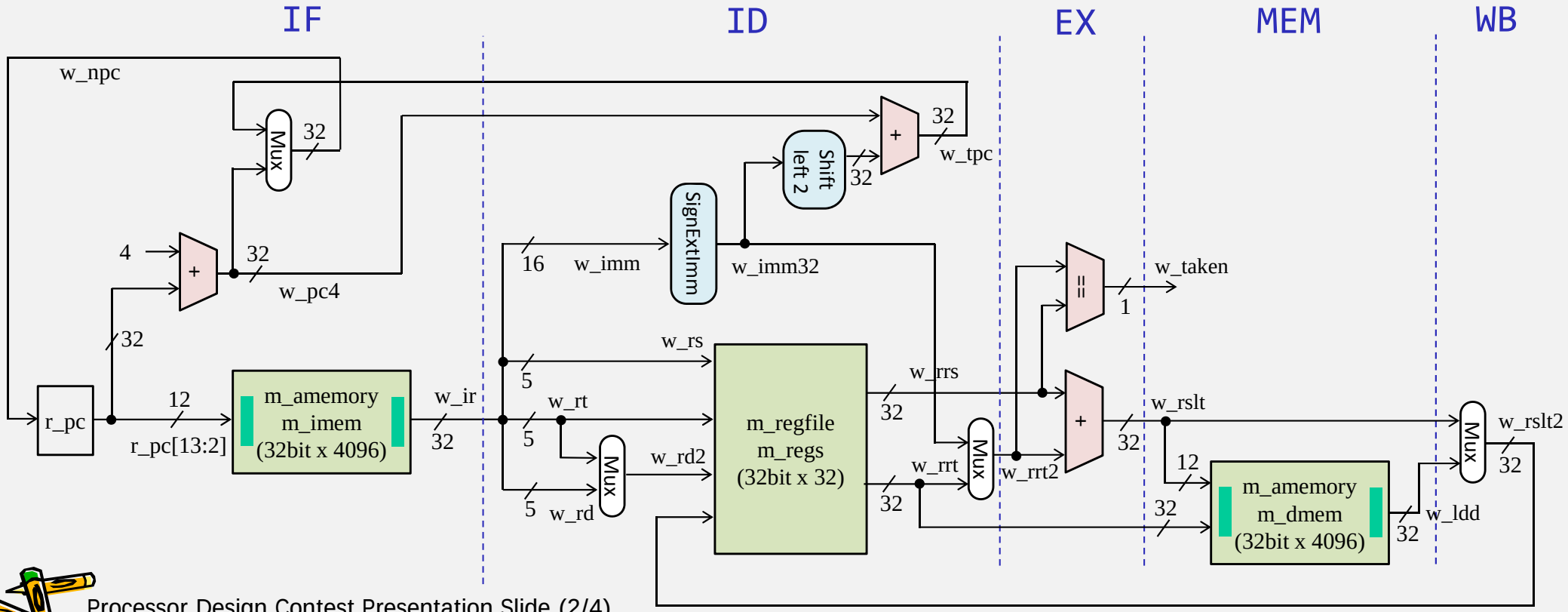
名前1，名前2，名前

名前3，名前4，名前5

設計したプロセッサと性能

- 動作周波数: X MHz, クロック周期 Y = 1000/X nsec
- プログラムの実行サイクス数: Z サイクル
- プロセッサ性能: Y x Z / 1000,000,000 = E sec

設計したプロセッサのブロック図



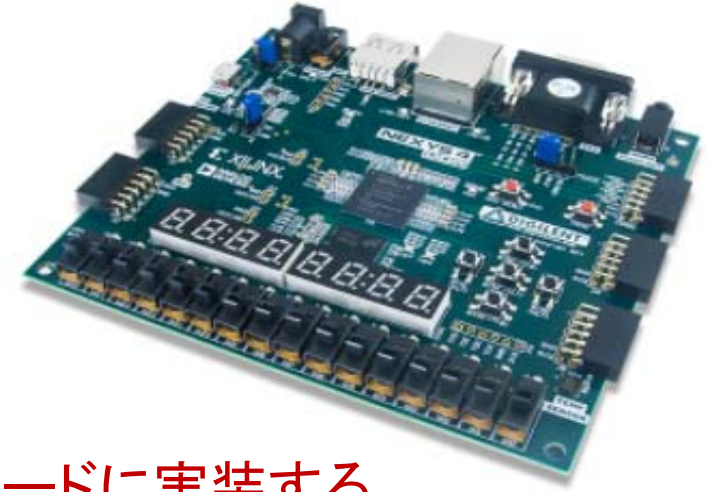
XXによる動作周波数の向上（工夫した点）

- コンテストに関する注意点．この文章は削除すること
  - スライド4枚にまとめること．
  - 発表時間は1グループ 4分とする．
  - 性能の高いプロセッサを設計したチームを「優勝」とする．
  - その他，参加者全員の投票により次の2つの賞を贈呈する．
  - 「最優秀アイデア賞」
  - 「最優秀インプリメンテーション賞」
- スライド（PowerPointファイル）の提出方法
  - 電子メールに，作成した ContestSlideXX.pptx を添付して XXはグループ番号．report_at_arch.cs.titech.ac.jp に送信する．Replace _at_ by @ .
  - 電子メールのタイトルは Computer Logic Design Slide (Group XX) とすること．
  - 電子メールの本文には，名前，（もしあれば）課題に関する質問・コメントなどを記入．
  - 2018年5月29日（火）の深夜までに提出すること．

XXによる実行サイクル数の削減（工夫した点2）

- コンテストに関する注意点．この文章は削除すること
  - 1枚目，2枚目のスライドは，このフォーマットに従うこと．
  - 3枚目，4枚目のスライドは自由に修正して良い．
  - スライドは図を用いて要点をわかりやすく伝えるように努力する．
  - チーム全員が前に出て発表するが，実際に発表するのは1人とする．
  - 誰が発表するかはチーム内で相談して決めること．
  - 発表者は事前に十分な発表練習をしておくこと．

コンピュータ論理設計の特徴



講義2単位，演習1単位．

1人1台のFPGA (Field-Programmable Gate Array) ボードを用いた演習．

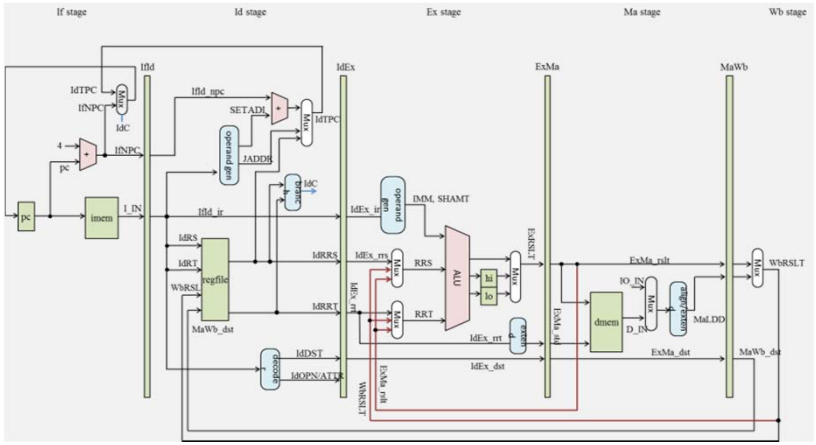
4人程度を1グループとした共同作業と問題解決．

教科書で説明されるプロセッサを ハードウェア記述言語Verilog HDLで記述し，FPGAボードに実装する．

グループとしてプロセッサの高速化に取り組み，コンテスト形式で成果を競う．

3Q開講のコンピュータアーキテクチャ(CSC.T363)のための準備．

```
module main (clk, led);
  input  wire clk;
  output wire led;

  reg [26:0] cnt;
  always @(posedge clk) cnt <= cnt + 1;

  assign led = cnt[26];
endmodule
```



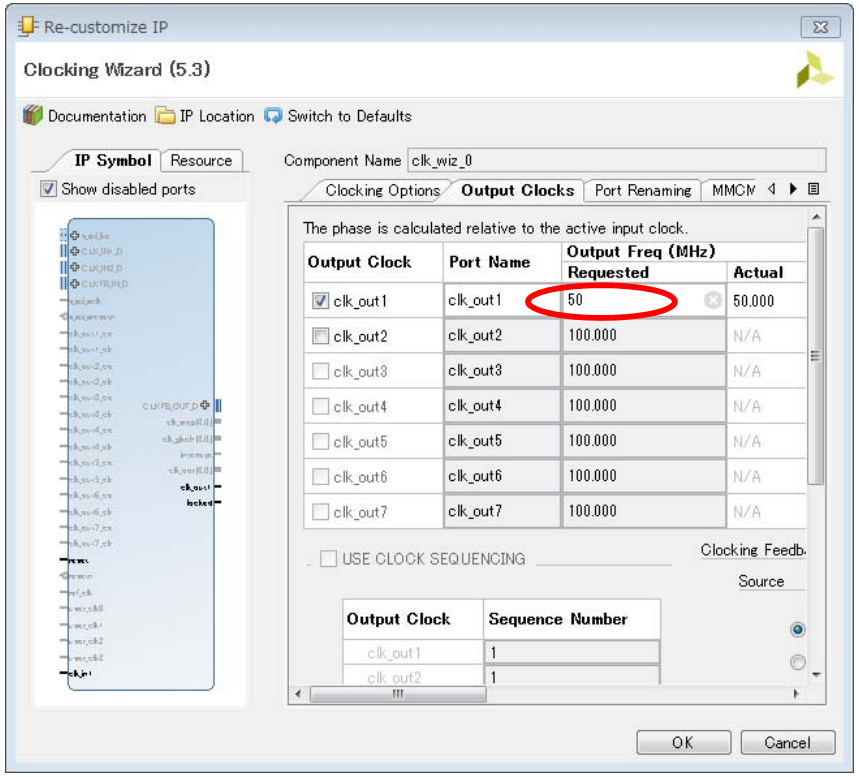
Create new project and create main.v and main.xdc

main09.xdc

```
set_property -dict { PACKAGE_PIN M18 IOSTANDARD LVCMOS33} [get_ports { w_btnu }];
set_property -dict { PACKAGE_PIN P18 IOSTANDARD LVCMOS33} [get_ports { w_btnd }];
set_property -dict { PACKAGE_PIN E3  IOSTANDARD LVCMOS33} [get_ports { w_clk }];
create_clock -add -name sys_clk -period 10.00 -waveform {0 5} [get_ports {w_clk}];

set_property -dict { PACKAGE_PIN H17 IOSTANDARD LVCMOS33} [get_ports { w_led[0] }];
set_property -dict { PACKAGE_PIN K15 IOSTANDARD LVCMOS33} [get_ports { w_led[1] }];
set_property -dict { PACKAGE_PIN J13 IOSTANDARD LVCMOS33} [get_ports { w_led[2] }];
set_property -dict { PACKAGE_PIN N14 IOSTANDARD LVCMOS33} [get_ports { w_led[3] }];
set_property -dict { PACKAGE_PIN R18 IOSTANDARD LVCMOS33} [get_ports { w_led[4] }];
set_property -dict { PACKAGE_PIN V17 IOSTANDARD LVCMOS33} [get_ports { w_led[5] }];
set_property -dict { PACKAGE_PIN U17 IOSTANDARD LVCMOS33} [get_ports { w_led[6] }];
set_property -dict { PACKAGE_PIN U16 IOSTANDARD LVCMOS33} [get_ports { w_led[7] }];
set_property -dict { PACKAGE_PIN V16 IOSTANDARD LVCMOS33} [get_ports { w_led[8] }];
set_property -dict { PACKAGE_PIN T15 IOSTANDARD LVCMOS33} [get_ports { w_led[9] }];
set_property -dict { PACKAGE_PIN U14 IOSTANDARD LVCMOS33} [get_ports { w_led[10] }];
set_property -dict { PACKAGE_PIN T16 IOSTANDARD LVCMOS33} [get_ports { w_led[11] }];
set_property -dict { PACKAGE_PIN V15 IOSTANDARD LVCMOS33} [get_ports { w_led[12] }];
set_property -dict { PACKAGE_PIN V14 IOSTANDARD LVCMOS33} [get_ports { w_led[13] }];
set_property -dict { PACKAGE_PIN V12 IOSTANDARD LVCMOS33} [get_ports { w_led[14] }];
set_property -dict { PACKAGE_PIN V11 IOSTANDARD LVCMOS33} [get_ports { w_led[15] }];

set_property -dict { PACKAGE_PIN T10 IOSTANDARD LVCMOS33} [get_ports { r_sg[6] }]; # segment a
set_property -dict { PACKAGE_PIN R10 IOSTANDARD LVCMOS33} [get_ports { r_sg[5] }]; # segment b
set_property -dict { PACKAGE_PIN K16 IOSTANDARD LVCMOS33} [get_ports { r_sg[4] }]; # segment c
set_property -dict { PACKAGE_PIN K13 IOSTANDARD LVCMOS33} [get_ports { r_sg[3] }]; # segment d
set_property -dict { PACKAGE_PIN P15 IOSTANDARD LVCMOS33} [get_ports { r_sg[2] }]; # segment e
set_property -dict { PACKAGE_PIN T11 IOSTANDARD LVCMOS33} [get_ports { r_sg[1] }]; # segment f
set_property -dict { PACKAGE_PIN L18 IOSTANDARD LVCMOS33} [get_ports { r_sg[0] }]; # segment g

set_property -dict { PACKAGE_PIN J17 IOSTANDARD LVCMOS33} [get_ports { r_an[0] }];
set_property -dict { PACKAGE_PIN J18 IOSTANDARD LVCMOS33} [get_ports { r_an[1] }];
set_property -dict { PACKAGE_PIN T9  IOSTANDARD LVCMOS33} [get_ports { r_an[2] }];
set_property -dict { PACKAGE_PIN J14 IOSTANDARD LVCMOS33} [get_ports { r_an[3] }];
set_property -dict { PACKAGE_PIN P14 IOSTANDARD LVCMOS33} [get_ports { r_an[4] }];
set_property -dict { PACKAGE_PIN T14 IOSTANDARD LVCMOS33} [get_ports { r_an[5] }];
set_property -dict { PACKAGE_PIN K2  IOSTANDARD LVCMOS33} [get_ports { r_an[6] }];
set_property -dict { PACKAGE_PIN U13 IOSTANDARD LVCMOS33} [get_ports { r_an[7] }];
```

Inside module m_proc07

- Project_10 を参考にして 50MHz のクロック w_clk2 を生成する.
- 論理合成, ビットストリームを生成し, FPGAをコンフィギュレーションする.



code110.v

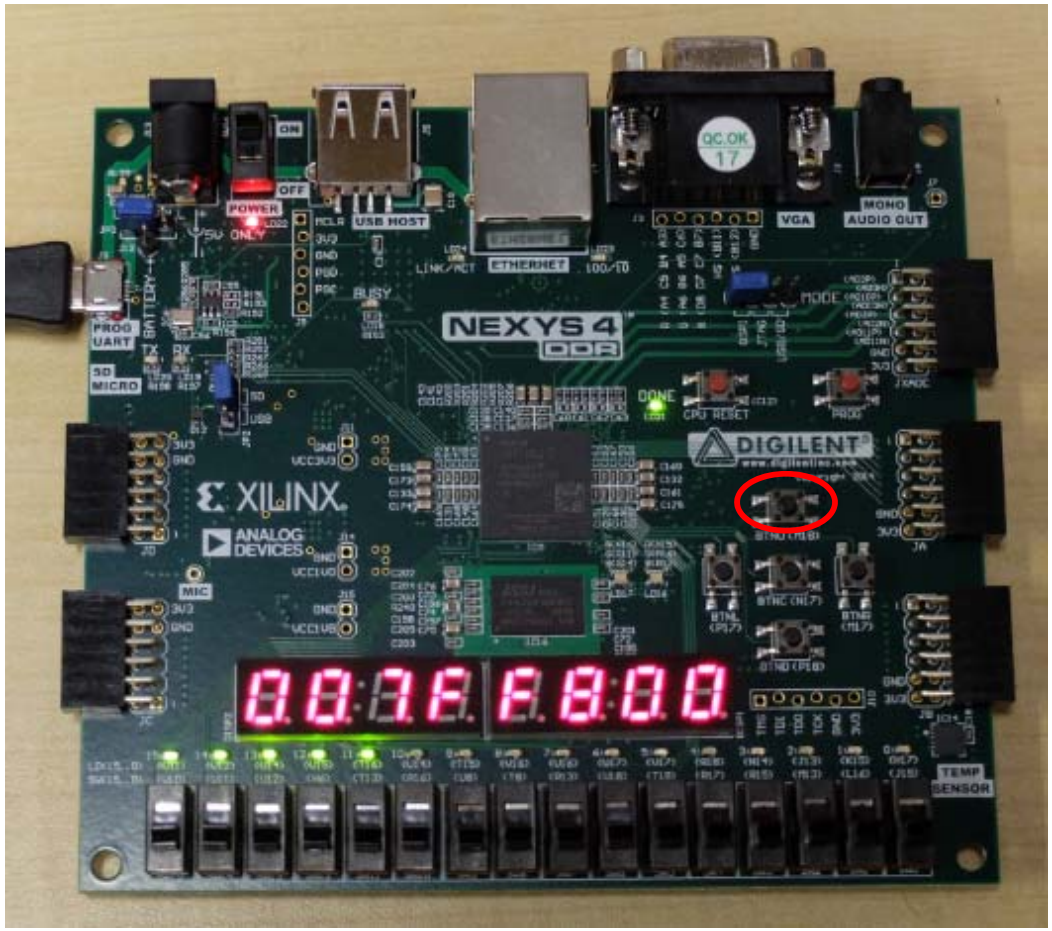
```
/*************************************************************************/
`default_nettype none

/*************************************************************************/
`define NOP  {21'h0, 11'h20} // add $0, $0, $0
`define ADD  6'h0
`define ADDI 6'h8
`define LW   6'h23
`define SW   6'h2b
`define BEQ  6'h4
`define BNE  6'h5
`define HALT 6'h11 /* this is not for MIPS */

/*************************************************************************/
module m_main (w_clk, w_btnu, w_btnd, w_led);
  input  wire w_clk, w_btnu, w_btnd;
  output wire [15:0] w_led;

  wire w_clk2, w_locked;
  clk_wiz_0 clk_wiz (w_clk2, 0, w_locked, w_clk);

  wire [31:0] w_rout;
  wire w_halt;
  wire w_rst = ~w_locked;
  m_proc07 p (w_clk2, w_rst, w_rout, w_halt);

  reg [31:0] r_cnt = 0;
  always @(posedge w_clk2) r_cnt <= (w_rst) ? 0 : (~w_halt) ? r_cnt + 1 : r_cnt;

  wire [31:0] w_data = (w_btnu) ? r_cnt : w_rout;
  assign w_led = (w_btnd) ? w_data[31:16] : w_data[15:0];
endmodule
```

Inside module m_proc07

- 論理合成，ビットストリームを生成し，FPGAをコンフィギュレーションする．
- 計算結果の 0x007FF800 が表示される．
- BTNU を押すことで，実行に要したサイクル数が表示される．

code01.c

```
#include <stdio.h>

main()
{
    int i=0, sum = 0;
    for(i=0; i<4096; i++) sum += i;
    printf("%d %x¥n", sum, sum);
}
```



ベースラインのプロセッサの仕様

- シングルサイクルのプロセッサ
- 50MHzのクロック信号
- bne, halt命令の追加. halt命令の実行が完了されるとプロセッサの出力信号 w_halt を 1 とする.
- 命令が r30 の値を読み出した時に, この読み出した値をレジスタに保存してプロセッサの出力信号 w_rout とする.
- halt命令実行時のw_routの値を計算結果とする. この値が正しい必要がある.
- 計算結果とプログラムの処理に要したサイクル数をLEDに表示する.
- サンプルプログラム
  - 4096ワードのデータメモリに, 0～4095の値をストアする.
  - 4096ワードのデータメモリを読み出し, 全ての読み出した値の合計値を求める.
  - 求めた合計値0x007FF800 を r30 に書き込み, r30 を読むことで, w_rout に出力する.
  - halt命令を実行してプロセッサの動作を止める.
  - このプログラムを実行するために要するサイクル数は 0x0000900d となる.

Inside module m_proc07

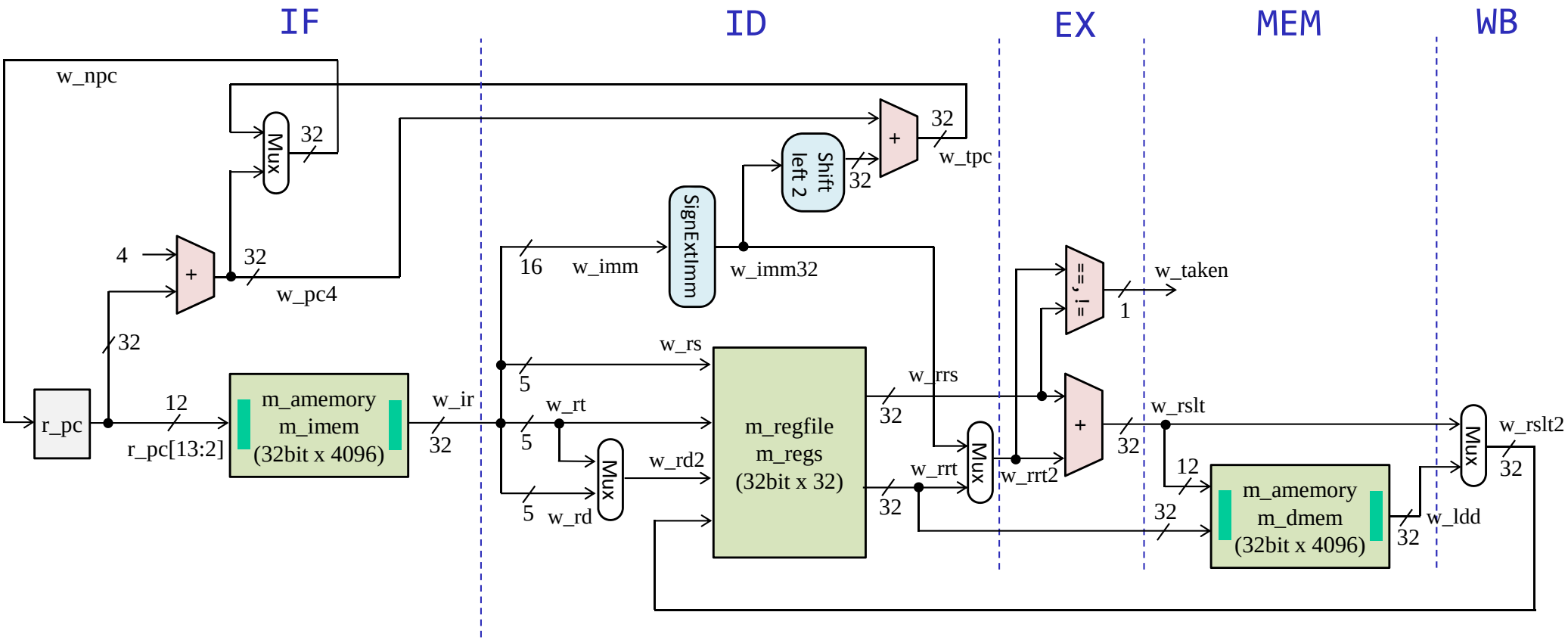
code110.v


```
module m_aimemory (w_clk, w_addr, w_we, w_din, w_dout);
  input  wire w_clk, w_we;
  input  wire [11:0] w_addr;
  input  wire [31:0] w_din;
  output wire [31:0] w_dout;
  reg [31:0] cm_ram [0:4095]; // 4K word (4096 x 32bit) memory
  always @(posedge w_clk) if (w_we) cm_ram[w_addr] <= w_din;
  assign #20 w_dout = cm_ram[w_addr];
  initial begin
    cm_ram[0] ={`NOP};                                //     nop
    cm_ram[1] ={`ADDI, 5'd0, 5'd8, 16'd4096};     //     addi $8, $0, 4095
    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, 4095
    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
    cm_ram[25]={`NOP};                                //     nop
    cm_ram[26]={`NOP};                                //     nop
  end
endmodule
```

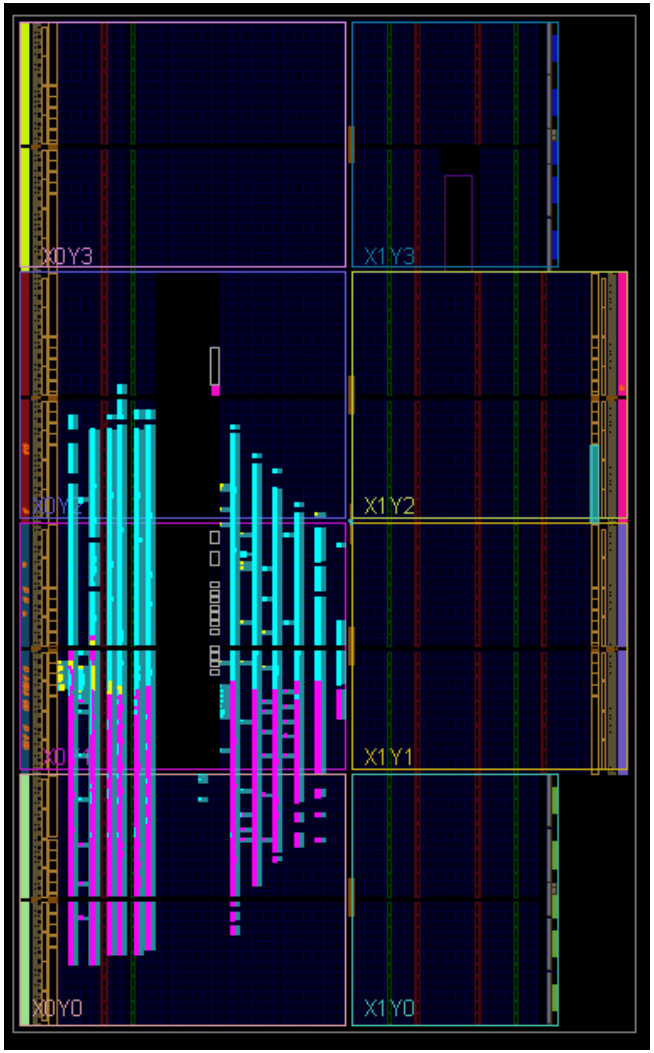
コンテストではこの赤い部分が変わる.
実行命令数の多いプログラムを用いる.

Inside module m_proc07

- 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



Inside module m_proc07



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));
  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;

  assign #10 w_rslt = w_rrs + w_rrt2;

  wire        w_we = (w_op>6'h27);
  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;

  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
```

Inside module m_proc07

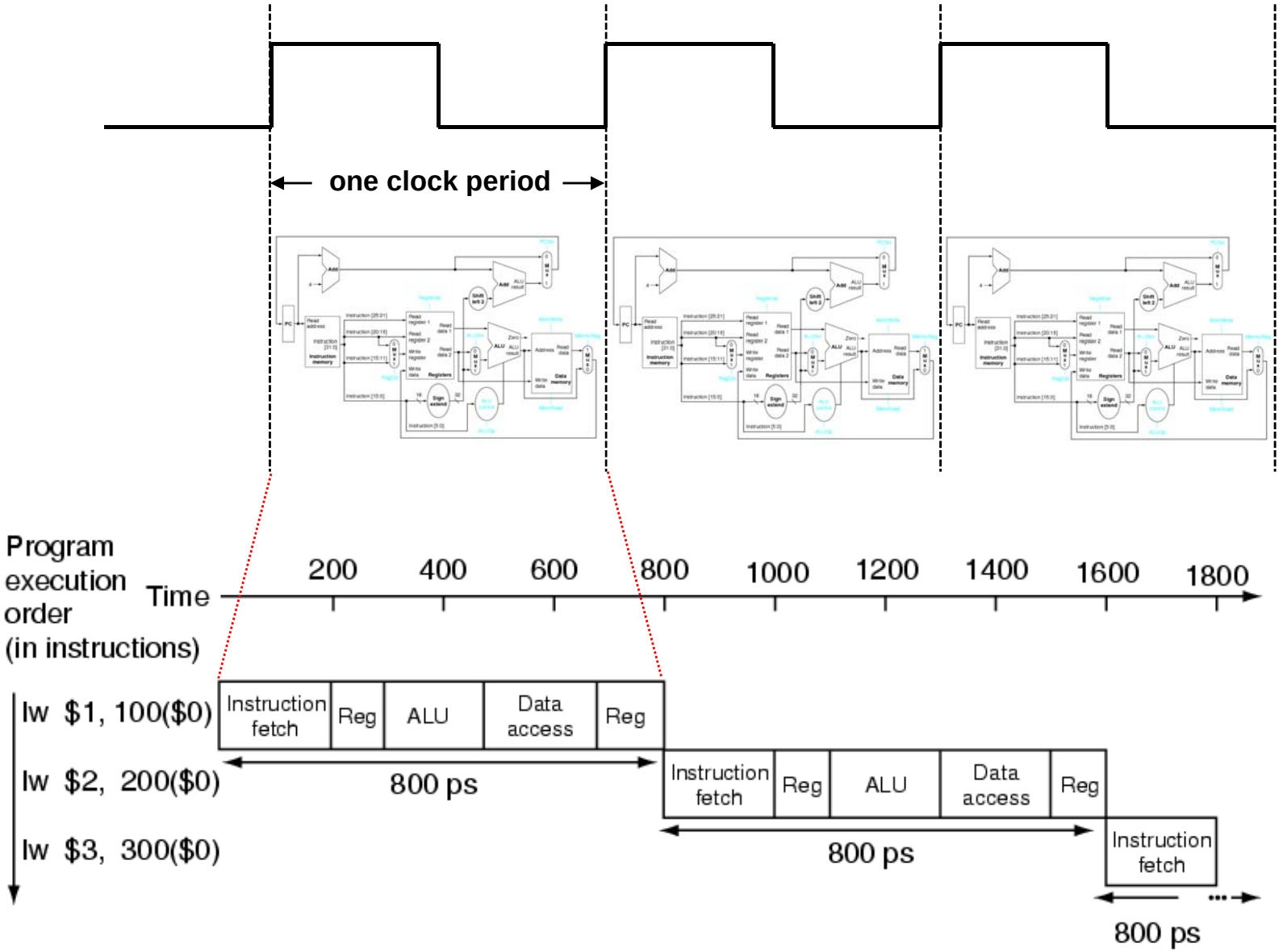
code110.v

```
module m_7segled (w_in, r_led);
  input  wire [3:0] w_in;
  output reg  [6:0] r_led;
  always @(*) begin
    case (w_in)
      4'h0  : r_led <= 7'b1111110;
      4'd0  : r_led <= 7'b0110000;
      4'h2  : r_led <= 7'b1101101;
      4'h3  : r_led <= 7'b1111001;
      4'h4  : r_led <= 7'b0110011;
      4'h5  : r_led <= 7'b1011011;
      4'h6  : r_led <= 7'b1011111;
      4'h7  : r_led <= 7'b1110000;
      4'h8  : r_led <= 7'b1111111;
      4'h9  : r_led <= 7'b1111011;
      4'ha  : r_led <= 7'b1110111;
      4'hb  : r_led <= 7'b0011111;
      4'hc  : r_led <= 7'b1001110;
      4'hd  : r_led <= 7'b0111101;
      4'he  : r_led <= 7'b1001111;
      4'hf  : r_led <= 7'b1000111;
      default:r_led <= 7'b0000000;
    endcase
  end
endmodule
```

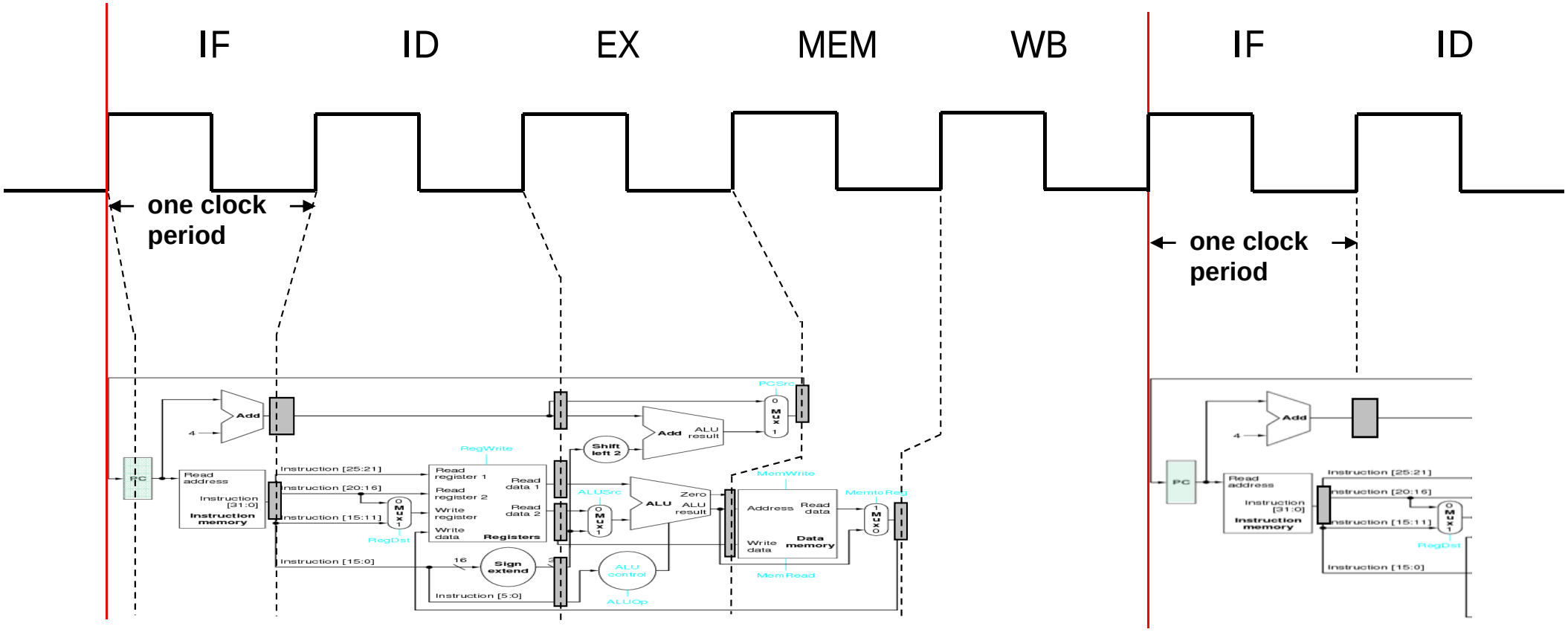
code110.v

```
`define DELAY7SEG  100000 // 200000 for 100MHz, 100000 for 50MHz
module m_7segcon (w_clk, w_din, r_sg, r_an);
  input  wire w_clk;
  input  wire [31:0] w_din;
  output reg [6:0] r_sg;  // cathode segments
  output reg [7:0] r_an;  // common anode

  reg [31:0] r_val   = 0;
  reg [31:0] r_cnt   = 0;
  reg  [3:0] r_in    = 0;
  reg  [2:0] r_digit = 0;
  always@(posedge w_clk) r_val <= w_din;

  always@(posedge w_clk) begin
    r_cnt <= (r_cnt>=(`DELAY7SEG-1)) ? 0 : r_cnt + 1;
    if(r_cnt==0) begin
      r_digit <= r_digit+ 1;
      if      (r_digit==0) begin r_an <= 8'b11111110; r_in <= r_val[3:0];   end
      else if (r_digit==1) begin r_an <= 8'b11111101; r_in <= r_val[7:4];   end
      else if (r_digit==2) begin r_an <= 8'b11111011; r_in <= r_val[11:8];  end
      else if (r_digit==3) begin r_an <= 8'b11110111; r_in <= r_val[15:12]; end
      else if (r_digit==4) begin r_an <= 8'b11101111; r_in <= r_val[19:16]; end
      else if (r_digit==5) begin r_an <= 8'b11011111; r_in <= r_val[23:20]; end
      else if (r_digit==6) begin r_an <= 8'b10111111; r_in <= r_val[27:24]; end
      else                 begin r_an <= 8'b01111111; r_in <= r_val[31:28]; end
    end
  end
  wire [6:0] w_segments;
  m_7segled m_7segled (r_in, w_segments);

  always@(posedge w_clk) r_sg <= ~w_segments;
endmodule
```

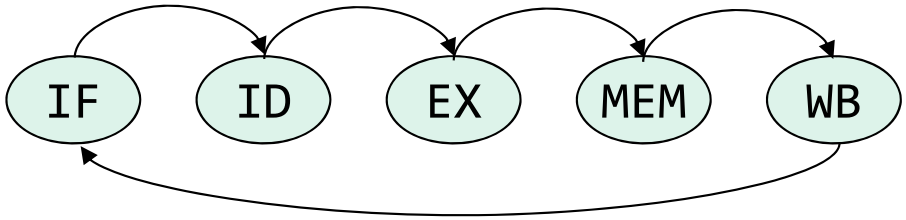
Single Cycle Processor (our baseline processor)



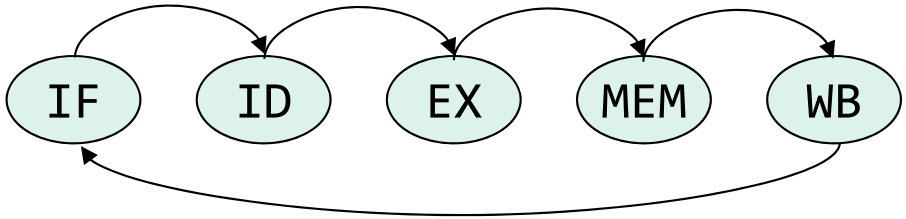
Multi-cycle Processor



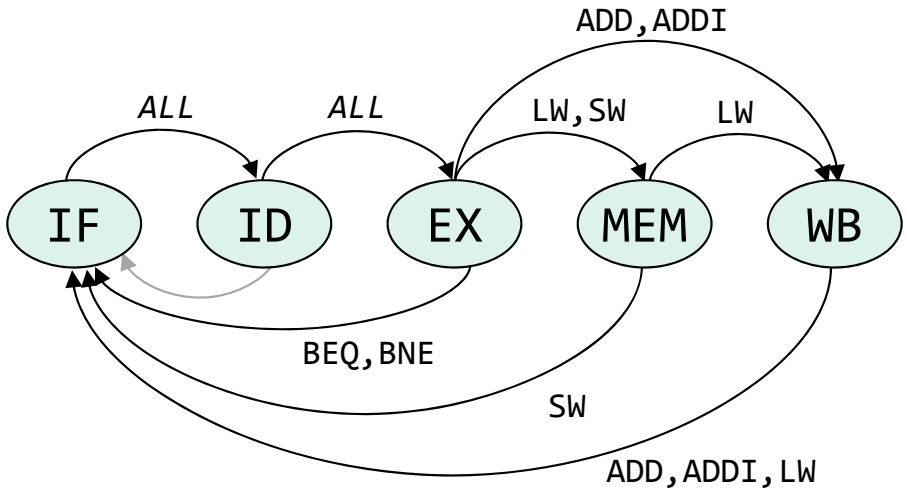
State Machine Diagram



Multi-cycle Processor



Simple State Machine Diagram

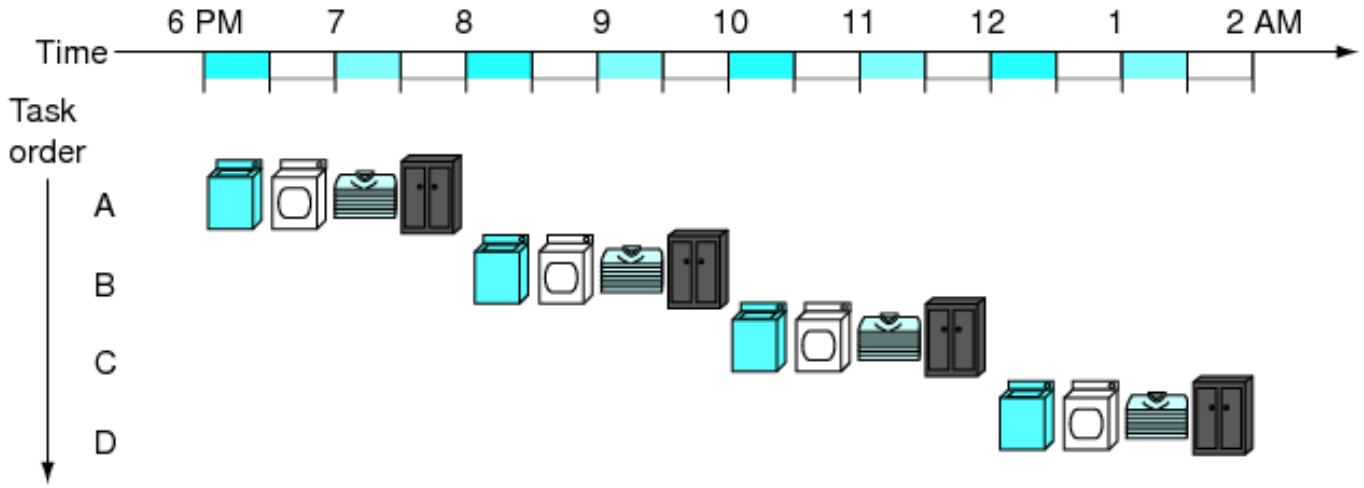


Optimized State Machine Diagram

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

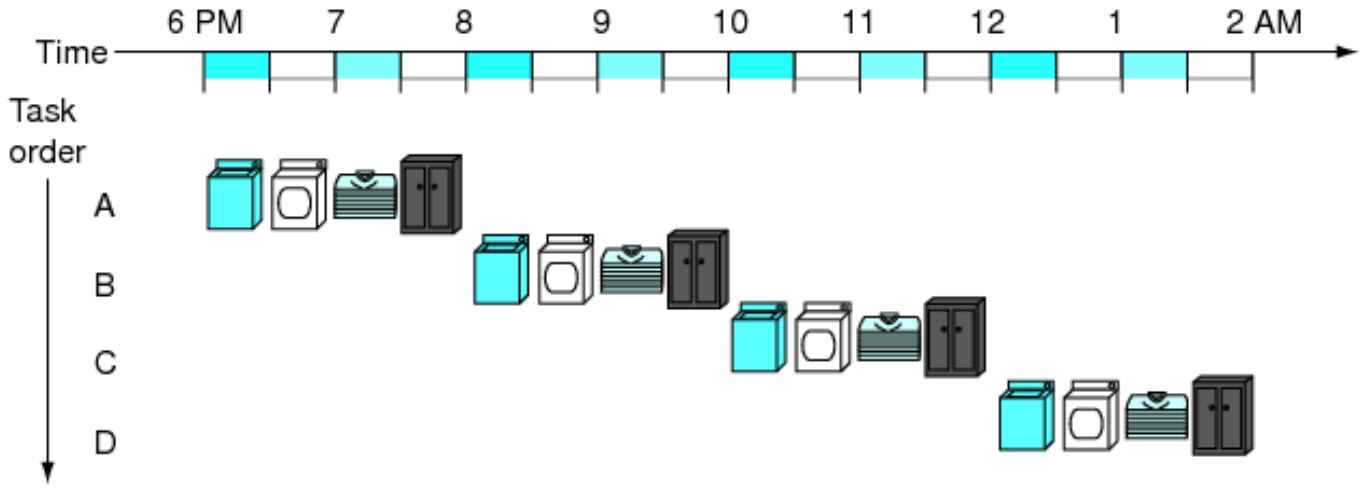
Pipelined Processor

- Non pipelining (Multi-cycle)

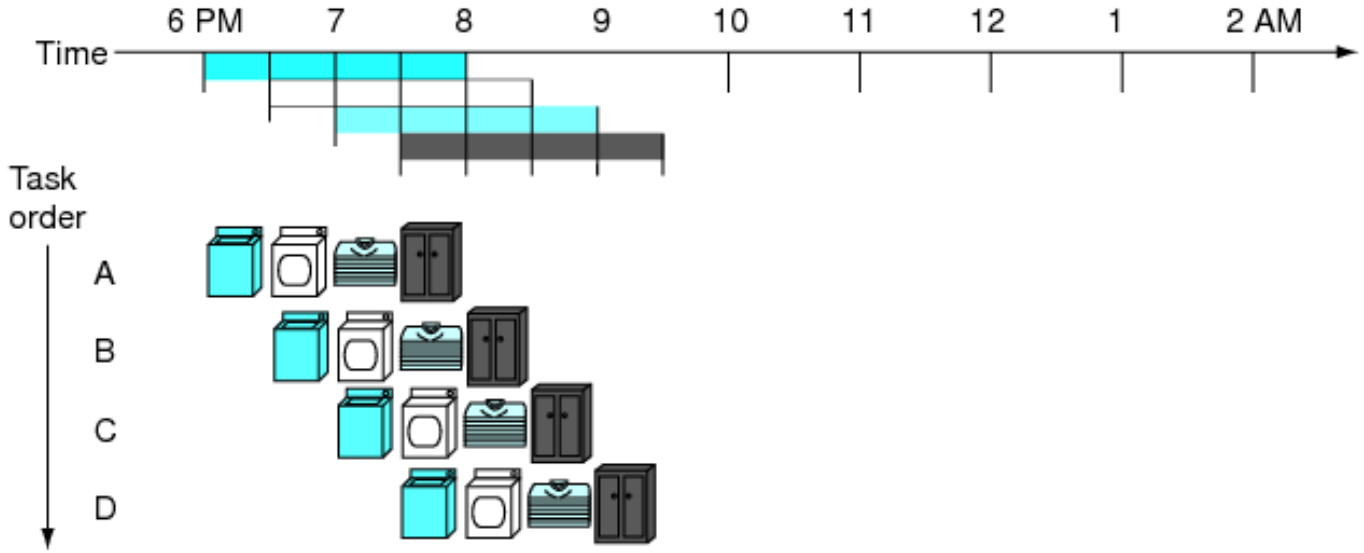


Pipelined Processor

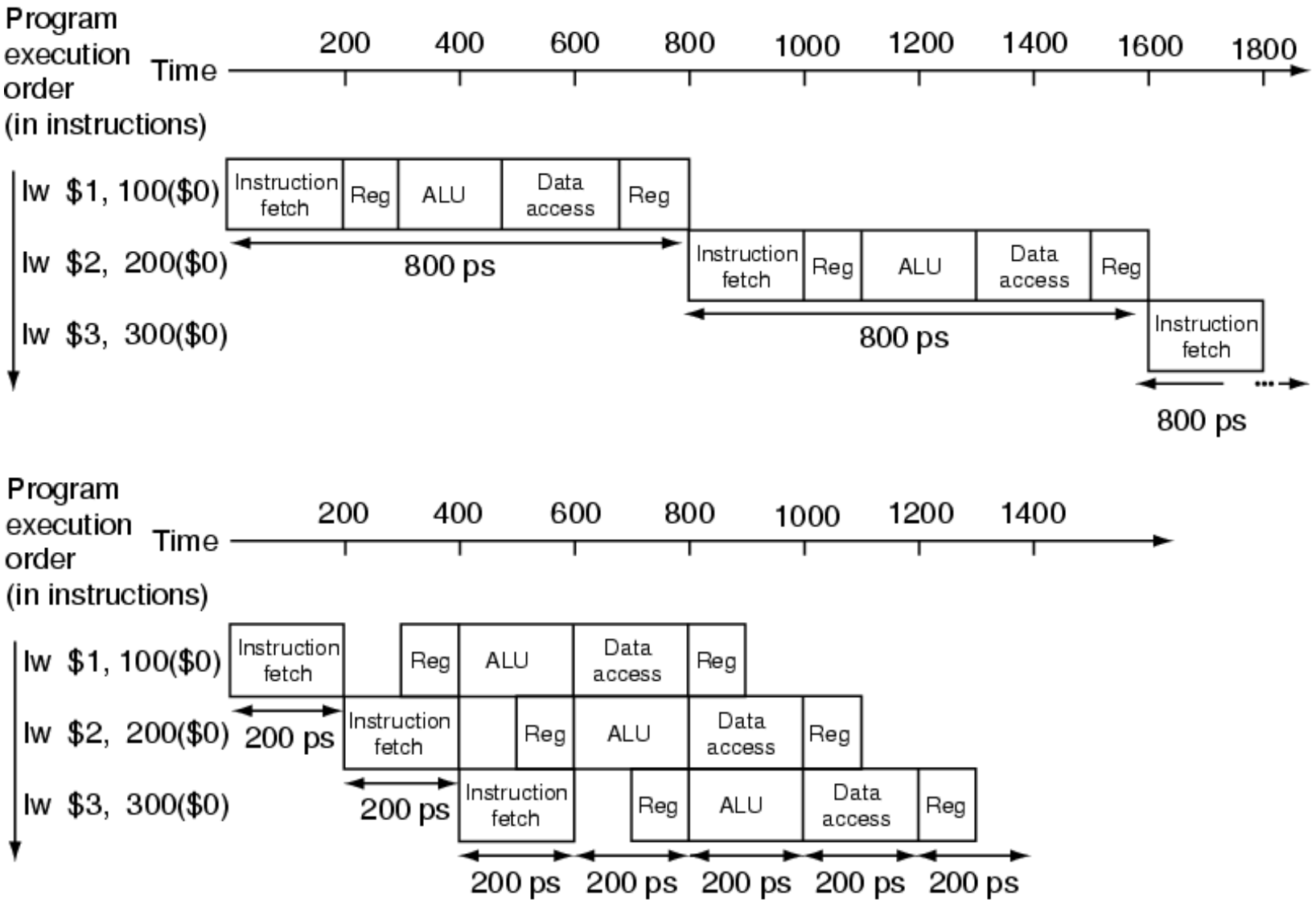
- Non pipelining (Multi-cycle)



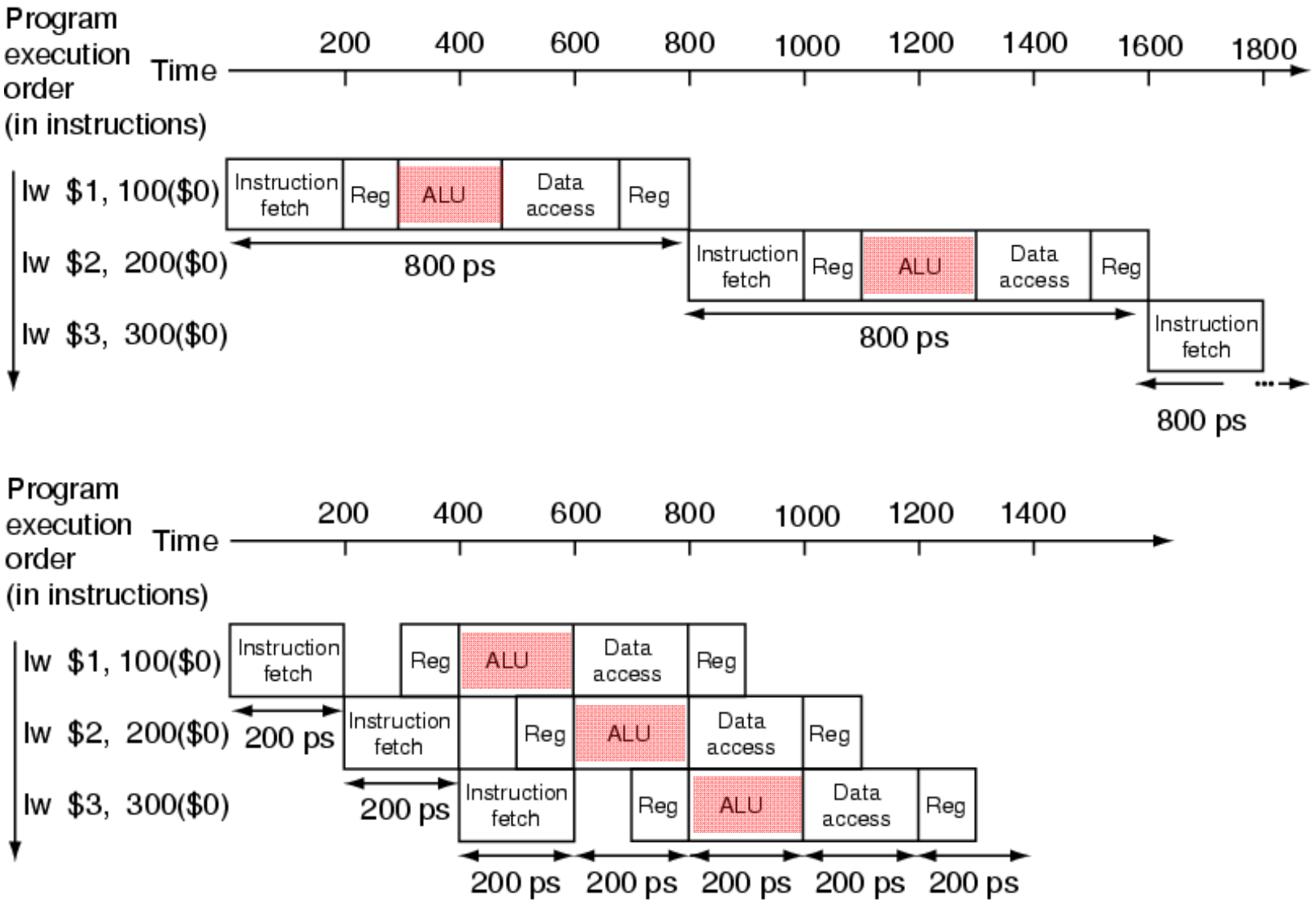
- Pipelining



Pipelined Processor



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