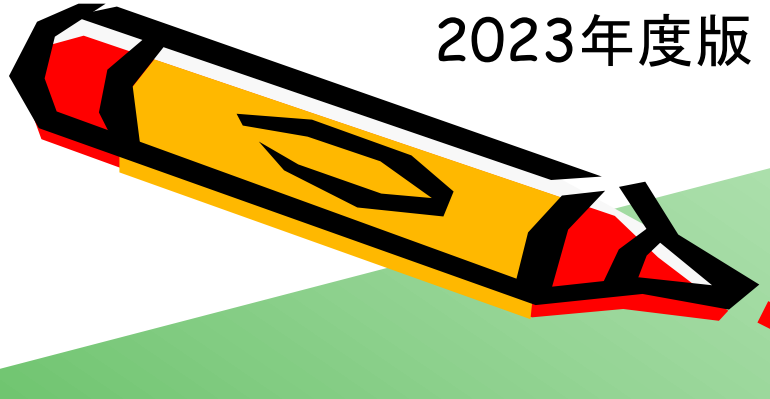


2023年度版

Ver. 2023-05-10b

Course number: CSC.T341

コンピュータ論理設計 演習(5)
Computer Logic Design Exercise(5)

情報工学系　荒堀喜貴

Yoshitaka ARAHORI, Department of Computer Science

arahori_at_c.titech.ac.jp

Computer Logic Design support page https://www.arch.cs.titech.ac.jp/lecture/CLD/

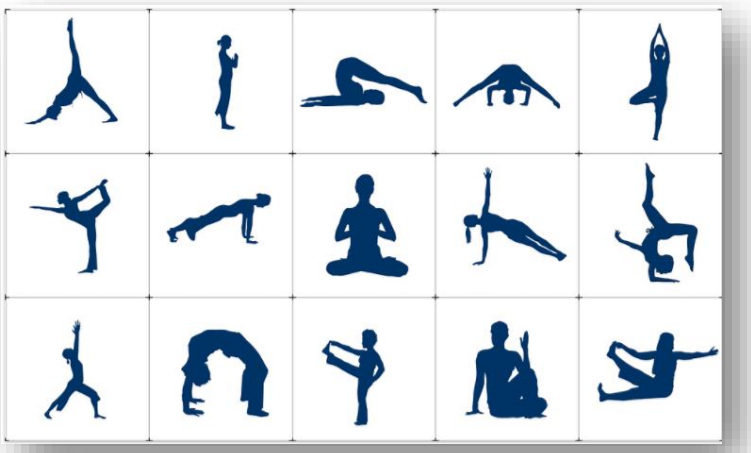
コンピュータ論理設計 演習(Exercise)の注意点

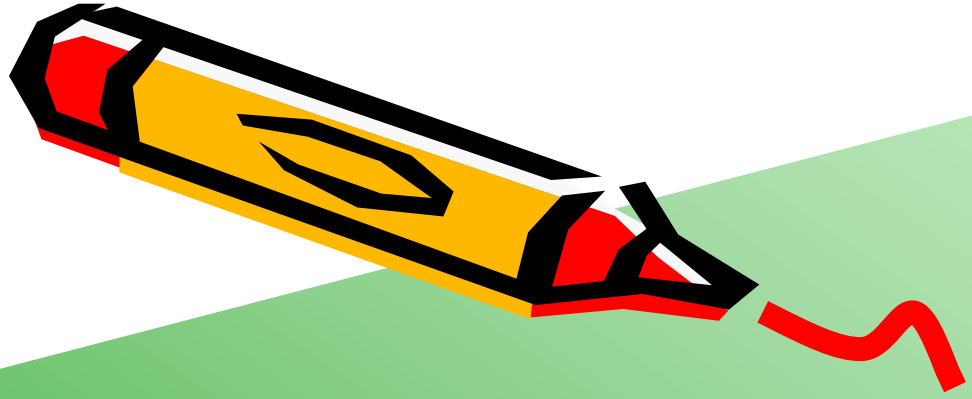
- 演習はACRiルームを利用します。
- 3～4人のグループを作成します．そのグループ内で情報を共有しながら演習を進めてください．
- 問題はグループ内で相談して解決する，あるいは，担当のTA(Teaching Assistant)や教員に質問してください．
- 演習には出席点があります．休まずにきちんと出席しましょう．
- 演習スライドにチェックポイントの図がある場所は，作業を確認してもらう場所です．すべてのチェックポイントをクリアしましょう．

- 演習時間でなくてもACRiルームを利用できます．現在は，1日に4枠（3時間 x 4枠 = 12時間）を利用できます。独自のハードウェア設計などに挑戦しましょう．

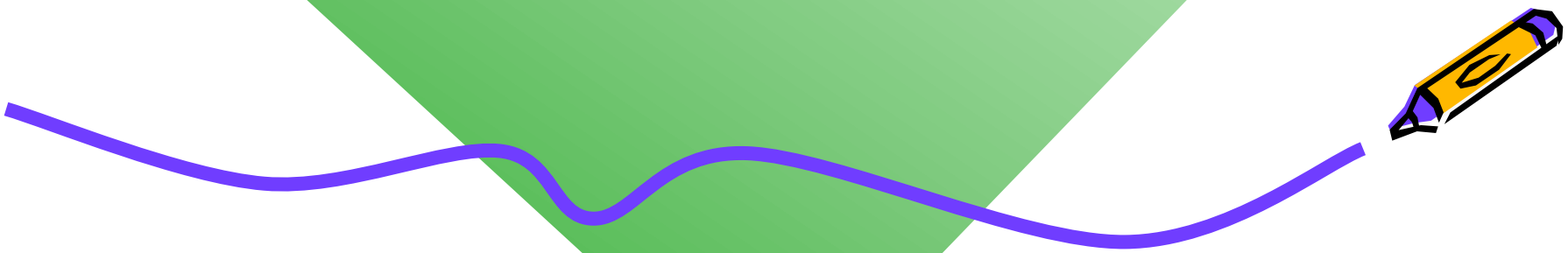
Exercise(5)

- Project_7
  - シングルサイクルプロセッサ m_proc06 の仕組みを理解する.
  - RISC-Vアセンブリ言語のプログラムを記述して，その正しさをシミュレーションで確認する.
  - m_proc06 を FPGA で動作させ，挙動が正しいことを VIO で確認する.
  - m_proc06 のクリティカルパスを確認する.





Project_7



新しい Vivado プロジェクトの作成とファイルの登録

- 前回の演習を参考に，Vivado で新しいプロジェクト project_7 を作成する．
- Ubuntu で起動したターミナルで，次のコマンドを実行してファイルをコピーする．
  - /home/tu_kise は automount のディレクトリなので，アクセスしないとファイルが見えない．tabキーによる補完がうまく動作しないことがあるので注意する．
  - 最後の ls コマンドで，code160.v, program.txt, main15.xdc が表示されることを確認．

```
$ ls /home/tu_kise
$ cd ~/cld/project_7
$ cp /home/tu_kise/cld/2023/code160.v .
$ cp /home/tu_kise/cld/2023/program.txt .
$ cp /home/tu_kise/cld/2023/main15.xdc .
$ ls
```

- Vivado で，project_7 の制約ファイルとして main15.xdc を登録する．
  - main15.xdc ファイルの内容を変更する必要はない．
- Vivado で，project_7 のVerilog HDLファイルとして code160.v を登録する

FPGA constraint file, XDC (Xilinx Design Constraints)

- main15.xdc では，ビットストリームファイルを圧縮したり，高速にFPGAに送信するための指定を追加

main15.xdc

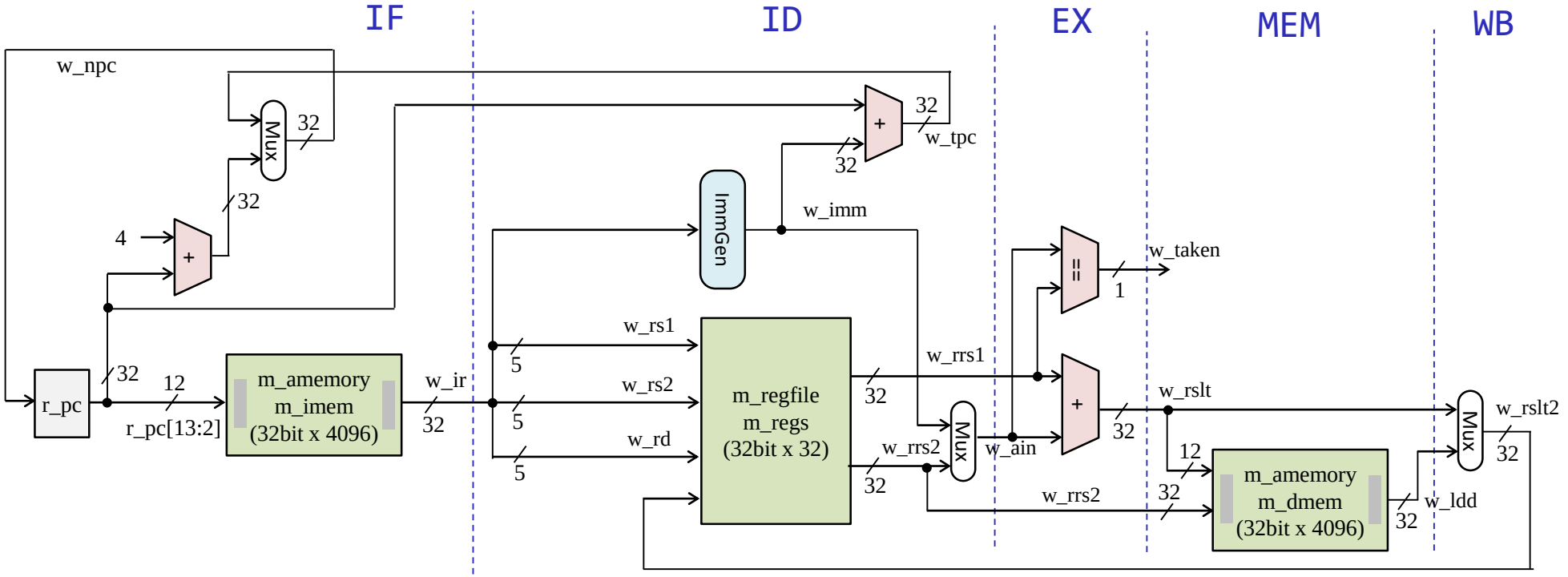
```
set_property BITSTREAM.GENERAL.COMPRESS TRUE [current_design]
set_property BITSTREAM.CONFIG.CONFIGRATE 50  [current_design]
set_property CONFIG_MODE SPIx4               [current_design]

set_property -dict { PACKAGE_PIN E3 IOSTANDARD LVCMOS33} [get_ports { w_clk }];
create_clock -add -name sys_clk -period 10.00 [get_ports {w_clk}];

set_property -dict { PACKAGE_PIN H5  IOSTANDARD LVCMOS33} [get_ports { w_led[0] }];
set_property -dict { PACKAGE_PIN J5  IOSTANDARD LVCMOS33} [get_ports { w_led[1] }];
set_property -dict { PACKAGE_PIN T9  IOSTANDARD LVCMOS33} [get_ports { w_led[2] }];
set_property -dict { PACKAGE_PIN T10 IOSTANDARD LVCMOS33} [get_ports { w_led[3] }];
```

main11.xdc

```
set_property -dict { PACKAGE_PIN E3 IOSTANDARD LVCMOS33} [get_ports { w_clk }];
create_clock -add -name sys_clk -period 10.00 [get_ports {w_clk}];

set_property -dict { PACKAGE_PIN H5  IOSTANDARD LVCMOS33} [get_ports { w_led[0] }];
set_property -dict { PACKAGE_PIN J5  IOSTANDARD LVCMOS33} [get_ports { w_led[1] }];
set_property -dict { PACKAGE_PIN T9  IOSTANDARD LVCMOS33} [get_ports { w_led[2] }];
set_property -dict { PACKAGE_PIN T10 IOSTANDARD LVCMOS33} [get_ports { w_led[3] }];
```

m_proc06 add, addi, lw, sw, beq を処理するプロセッサ

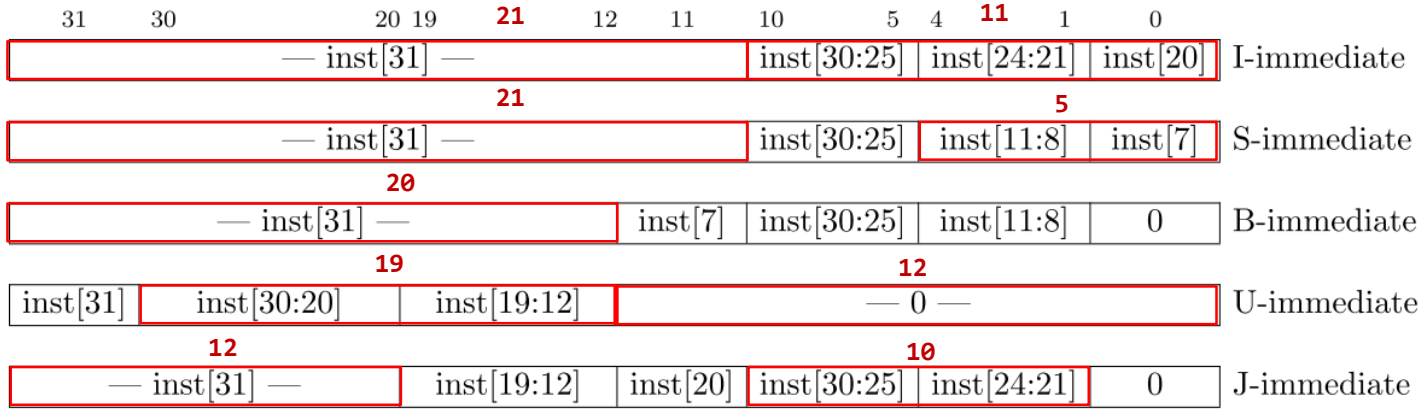
- 命令フェッチ(IF), デコードとオペランドフェッチ(ID), 実行(EX), メモリアクセス(MEM), ライトバック(WB) の処理をおこなうadd, addi, lw, sw, beq命令に対応したプロセッサ



31 30	25 24	21 20	19 15	14 12	11 8	7 6 0	
funct7	rs2		rs1	funct3	rd	opcode	R-type
imm[11:0]			rs1	funct3	rd	opcode	I-type
imm[11:5]	rs2		rs1	funct3	imm[4:0]	opcode	S-type
imm[12]	imm[10:5]	rs2	rs1	funct3	imm[4:1]	imm[11] opcode	B-type

m_immgen RISC-Vの即値を出力するモジュール

- 32ビットの命令を入力 w_i として，RISC-Vの即値 r_imm を出力するモジュール
- reg を使っているが，これは組合せ回路となる．



code160.v

```
module m_immgen(w_i, r_imm); // module immediate generator
  input  wire [31:0] w_i;    // instruction
  output reg  [31:0] r_imm;  // r_immediate

  always @(*) case (w_i[6:2])
    5'b11000: r_imm <= {{20{w_i[31]}}, w_i[7], w_i[30:25], w_i[11:8], 1'b0};   // B-type
    5'b01000: r_imm <= {{21{w_i[31]}}, w_i[30:25], w_i[11:7]};                 // S-type
    5'b11011: r_imm <= {{12{w_i[31]}}, w_i[19:12], w_i[20], w_i[30:21], 1'b0}; // J-type
    5'b01101: r_imm <= {w_i[31:12], 12'b0};                                    // U-type
    5'b00101: r_imm <= {w_i[31:12], 12'b0};                                    // U-type
    default : r_imm <= {{21{w_i[31]}}, w_i[30:20]};                  // I-type & R-type
  endcase
endmodule
```

m_amemory の修正 include “program.txt”

- 命令メモリとデータメモリとして用いる m_amemory の内容を program.txt で初期化.
- x30 への書き込みをレジスタに保存して，プロセッサの出力 w_led とする.
- このプログラムを実行して beq に到達した時点の w_led の値は 55 + 9 = 64 になる.

code160.v

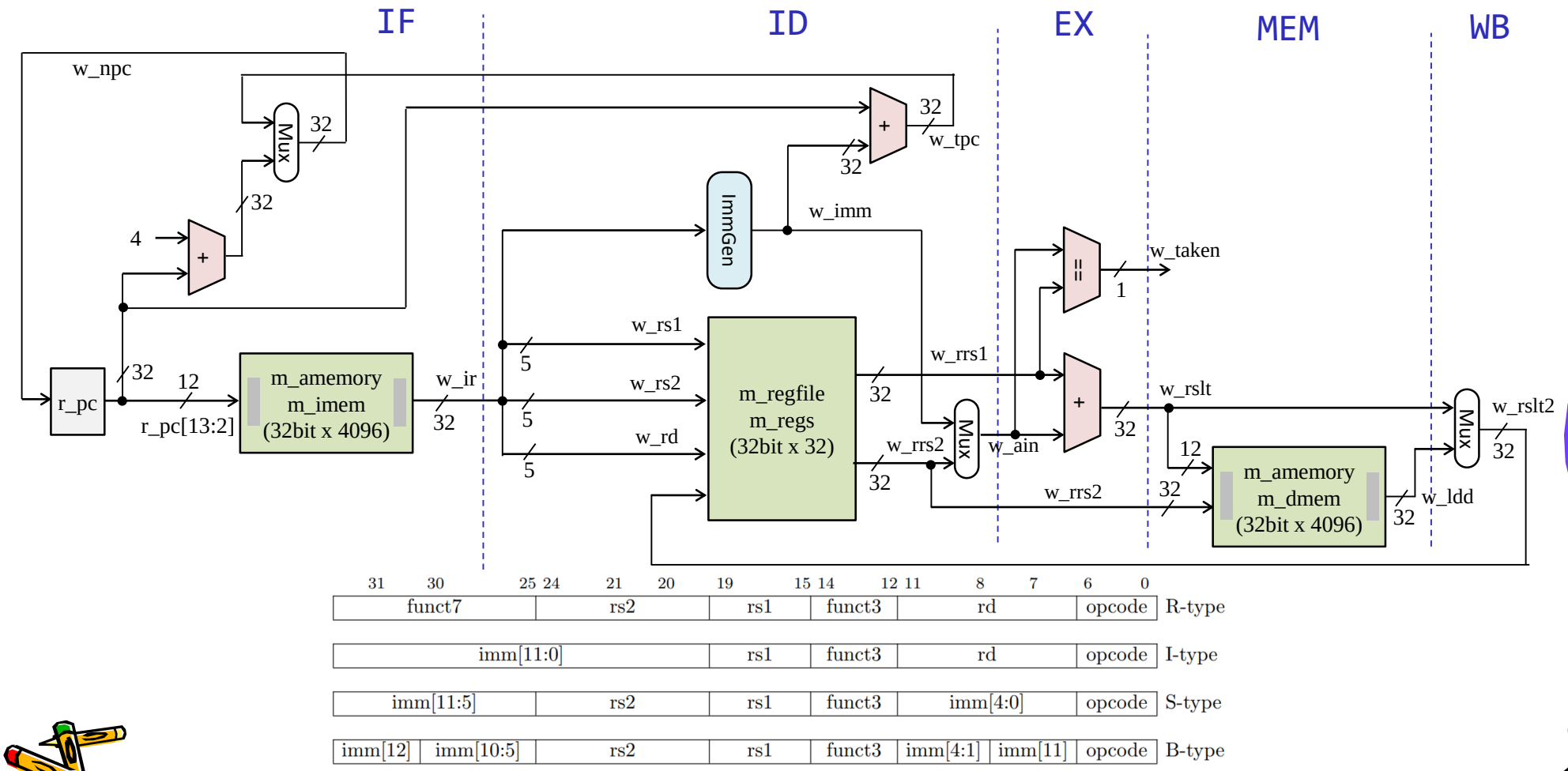
```
module m_amemory (w_clk, w_addr, w_we, w_din, w_dout);
  input  wire w_clk, w_we;
  input  wire [11:0] w_addr;
  input  wire [31:0] w_din;
  output wire [31:0] w_dout;
  reg [31:0]          cm_ram [0:4095]; // 4K word (4096 x 32bit) memory
  always @(posedge w_clk) if (w_we) cm_ram[w_addr] <= w_din;
  assign #20 w_dout = cm_ram[w_addr];
`include "program.txt"
endmodule
```

program.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'd55,     5'd0, 3'b000, 5'd4, 7'b0010011}; //    addi x4, x0, 55  // x4 = 55
  cm_ram[2]={7'd0, 5'd4, 5'd0, 3'b010, 5'd16,7'b0100011}; //    sw   x4, 16(x0)  // m[16] = x4
  cm_ram[3]={12'd16,     5'd0, 3'b010, 5'd7, 7'b0000011}; //    lw   x7, 16(x0)  // x7 = m[16]
  cm_ram[4]={12’d9,      5'd0, 3'b000, 5'd2, 7'b0010011}; //    addi x2, x0, 2   // x2 = 9
  cm_ram[5]={7'd0, 5'd2, 5'd7, 3'b000, 5'd3, 7'b0110011}; //    add  x3, x7, x2  // x3 = x7 + x2
  cm_ram[6]={7'd0, 5'd3, 5'd0, 3'b000, 5'd30,7'b0110011}; //    add  x30,x0, x3  // led = x3
  cm_ram[7]={7'd0, 5'd0, 5'd0, 3'b000, 5'd0, 7'b1100011}; // L: beq  x0, x0, L
end
```

m_proc06 を Verilog HDL で実装する

- program.txt の命令列が正しく実行できるように code160.v を修正して，その挙動をシミュレーションにより確認すること．
- m_proc06 のモジュール以外は修正する必要はない．



m_proc06 add, addi, lw, sw, beq を処理するプロセッサ

青色の部分を記述する.

code160.v

```
module m_proc06 (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];
  wire 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;

  /* Please describe here by yourself */


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

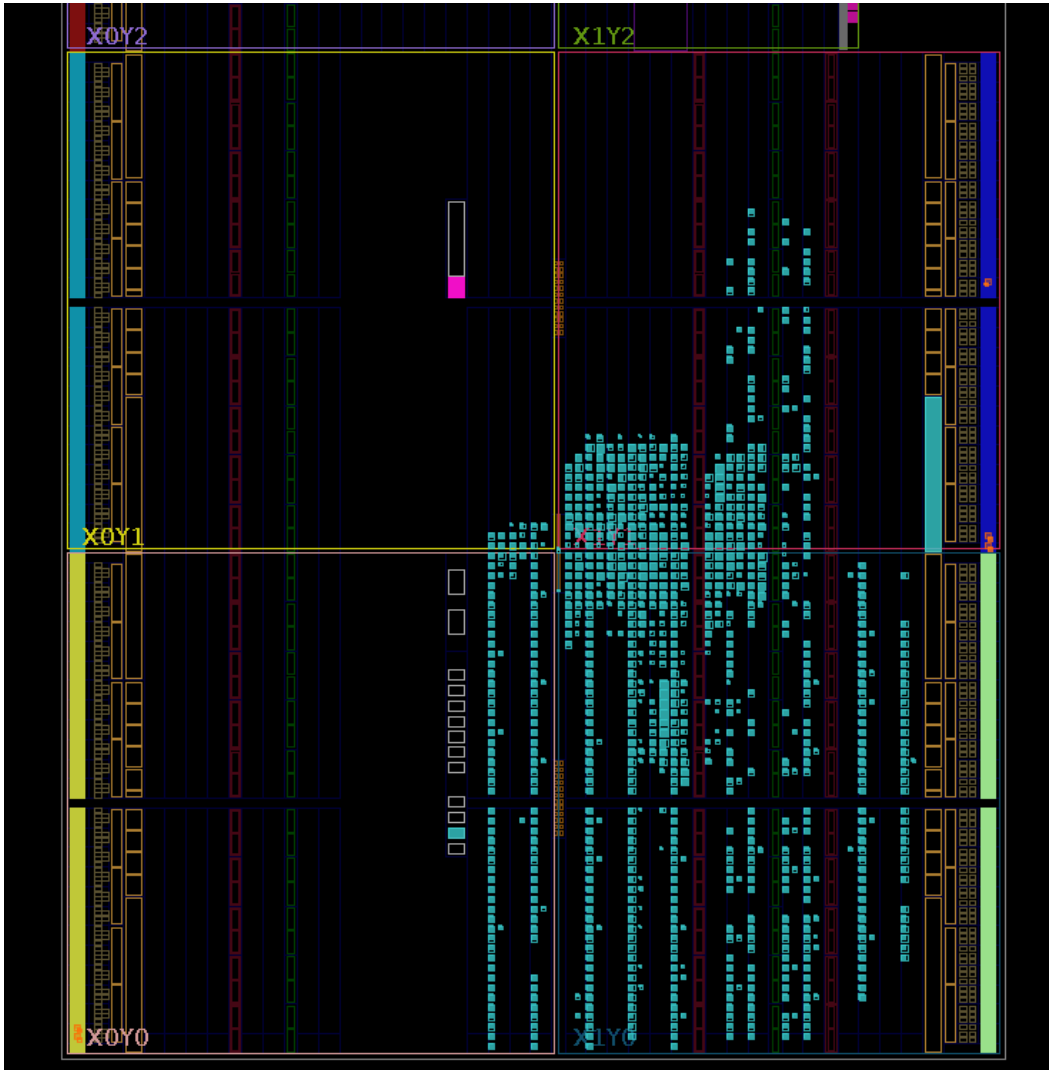
```
cm_ram[0]={7'd0, 5'd0, 5'd0, 3'b000, 5'd0, 7'b0110011}; //    add  x0, x0, x0  // NOP
cm_ram[1]={12'd55,     5'd0, 3'b000, 5'd4, 7'b0010011}; //    addi x4, x0, 55  // x4 = 55
cm_ram[2]={7'd0, 5'd4, 5'd0, 3'b010, 5'd16,7'b0100011}; //    sw   x4, 16(x0)  // m[16] = x4
cm_ram[3]={12'd16,     5'd0, 3'b010, 5'd7, 7'b0000011}; //    lw   x7, 16(x0)  // x7 = m[16]
cm_ram[4]={12'd9,      5'd0, 3'b000, 5'd2, 7'b0010011}; //    addi x2, x0, 9   // x2 = 9
cm_ram[5]={7'd0, 5'd2, 5'd7, 3'b000, 5'd3, 7'b0110011}; //    add  x3, x7, x2  // x3 = x7 + x2
cm_ram[6]={7'd0, 5'd3, 5'd0, 3'b000, 5'd30,7'b0110011}; //    add  x30,x0, x3  // led = x3
cm_ram[7]={7'd0, 5'd0, 5'd0, 3'b000, 5'd0, 7'b1100011}; // L: beq  x0, x0, L
```

m_proc06 の論理合成，配置配線の結果

Utilization Post-Synthesis | **Post-Implementation**

Graph | **Table**

Resource	Utilization	Available	Utilization %
LUT	3095	20800	14.88
LUTRAM	2184	9600	22.75
FF	1199	41600	2.88
IO	5	210	2.38
BUFG	3	32	9.38
MMCM	1	5	20.00



この配置配線の結果は，実装によって変わる．

code160.v のFPGAへの実装と動作確認

- 修正した code160.v のシミュレーション用のモジュール m_top をコメントアウトして，実装用のモジュール m_main のコメントアウトを外す（有効にする）.
- 過去の演習を参考に，35MHzのクロック信号を生成する Clocking Wizard のモジュール clk_wiz_0 を生成する.
- 過去の演習を参考に，プロセッサからの32ビットの出力 w_dout の値を表示するためのVIO のモジュール vio_0 のインスタンス vio_00 を生成する.
- 論理合成，配置配線してFPGAで動作させると，シミュレーションと同様の結果が VIO で確認できる.
- 記述した code160.v と VIO の値を「担当の教員あるいはTA」に確認してもらうこと.

code160.v

```
module m_main (w_clk, w_led);
   input  wire w_clk;
   output wire [3:0] w_led;

   wire w_clk2, w_locked;
   clk_wiz_0 clk_w0 (w_clk2, 0, w_locked, w_clk);

   wire [31:0] w_dout;
   m_proc06 p (w_clk2, w_locked, w_dout);

   vio_0 vio_00(w_clk2, w_dout);

   reg [3:0] r_led = 0;
   always @(posedge w_clk2)
     r_led <= {^w_dout[31:24], ^w_dout[23:16], ^w_dout[15:8], ^w_dout[7:0]};
   assign w_led = r_led;
endmodule
```

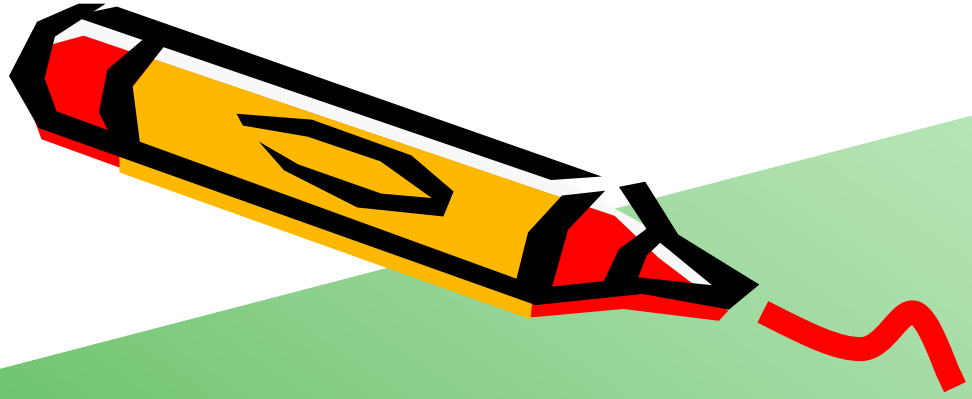
Check Point 7

ヒント

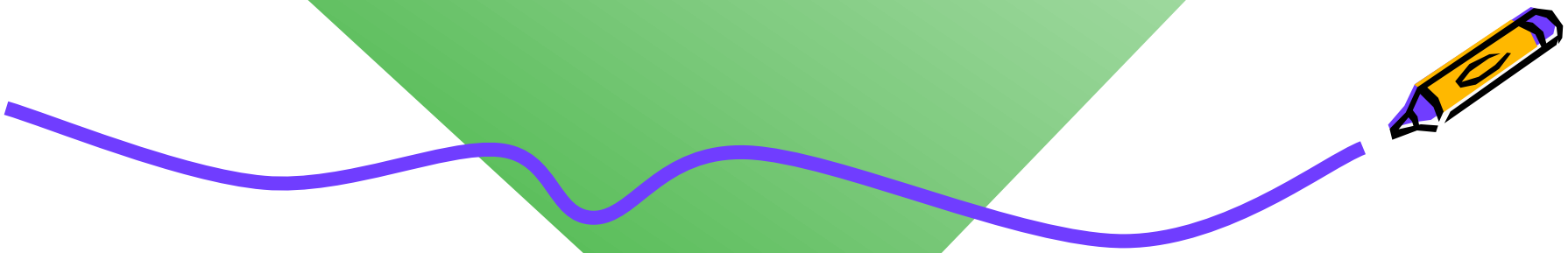
青色の部分を少し記述した.

code160.v

```
module m_proc06 (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];
  wire 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;

  /* Please describe here by yourself */
  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 w_ain = (w_op5==5'b01100) ? w_rrs2 : w_imm;
  assign w_rslt = w_rrs1 + w_ain;

  // about data memory here

  // about r_pc update here



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



References



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