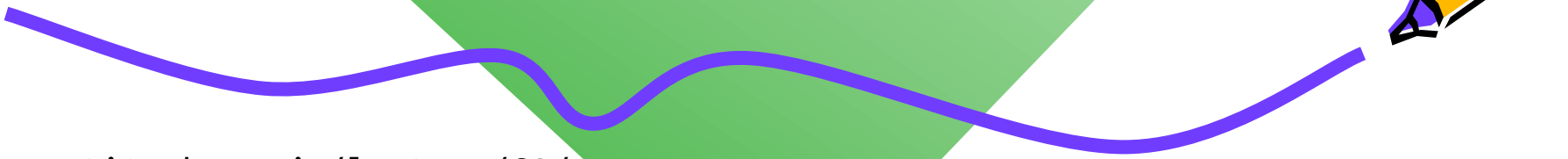
2024年度（令和6年）版

Ver. 2024-10-25b

Course number: CSC.T363

コンピュータアーキテクチャ
Computer Architecture

7. パイプラインプロセッサ
Pipelined Processor



www.arch.cs.titech.ac.jp/lecture/CA/
Tue 13:30-15:10, 15:25-17:05
Fri 13:30-15:10

吉瀬 謙二　情報工学系
Kenji Kise, Department of Computer Science
kise _at_ c.titech.ac.jp

RISC-V の命令長

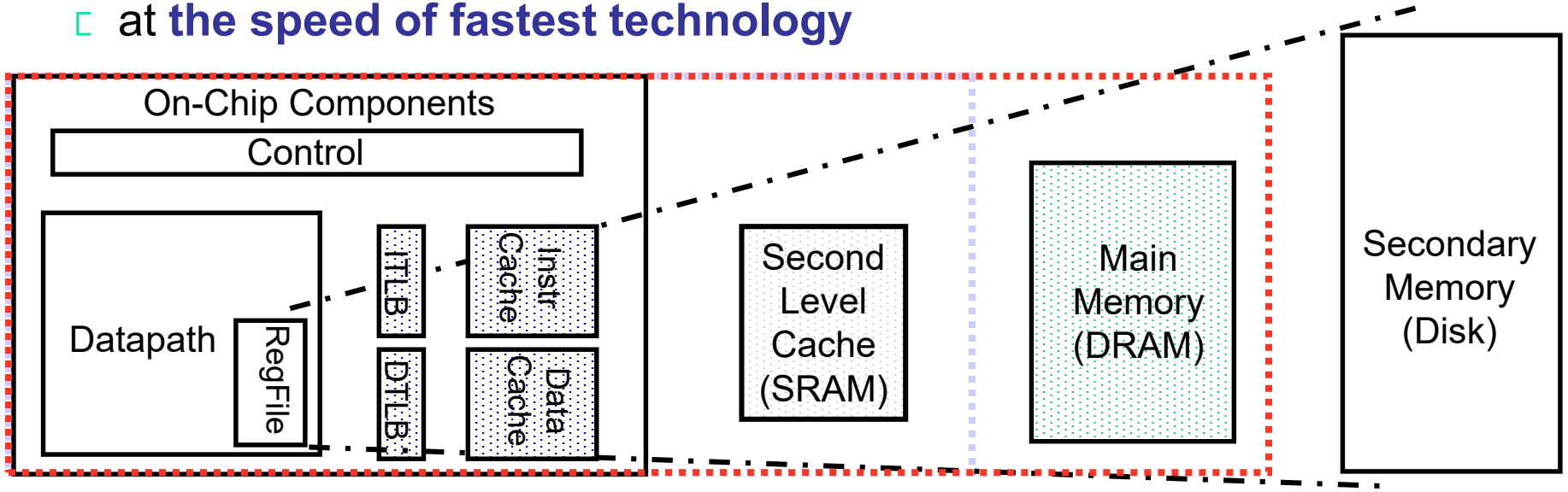
The RISC-V Instruction Set Manual
Volume I: Unprivileged ISA
Document Version 20191214-*draft*

Editors: Andrew Waterman[1], Krste Asanović[1,2]
[1]SiFive Inc.,
[2]CS Division, EECS Department, University of California, Berkeley
andrew@sifive.com, krste@berkeley.edu
November 12, 2021

Figure 1.1: RISC-V instruction length encoding. Only the 16-bit and 32-bit encodings are considered frozen at this time.

A Typical Memory Hierarchy

- By taking advantage of **the principle of locality**（局所性）
  - Present **much memory** in **the cheapest technology**
  - at **the speed of fastest technology**

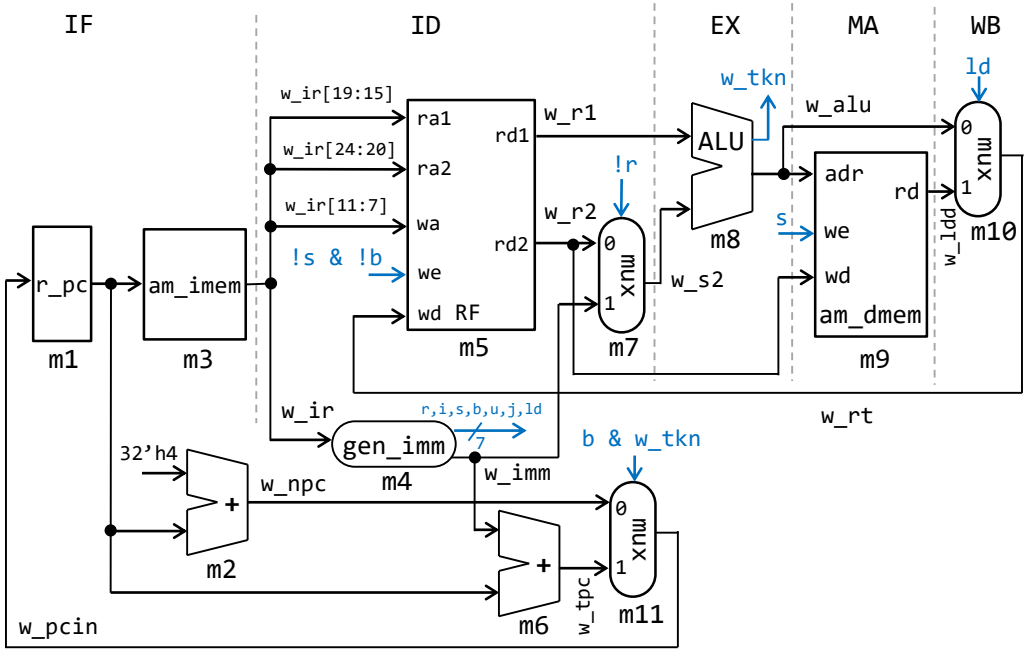


Speed (%cycles):	½'s	1's	10's	100's	1,000's
Size (bytes):	100's	K's	10K's	M's	G's to T's
Cost:	highest				lowest

TLB: Translation Lookaside Buffer

proc5s: single-cycle processor

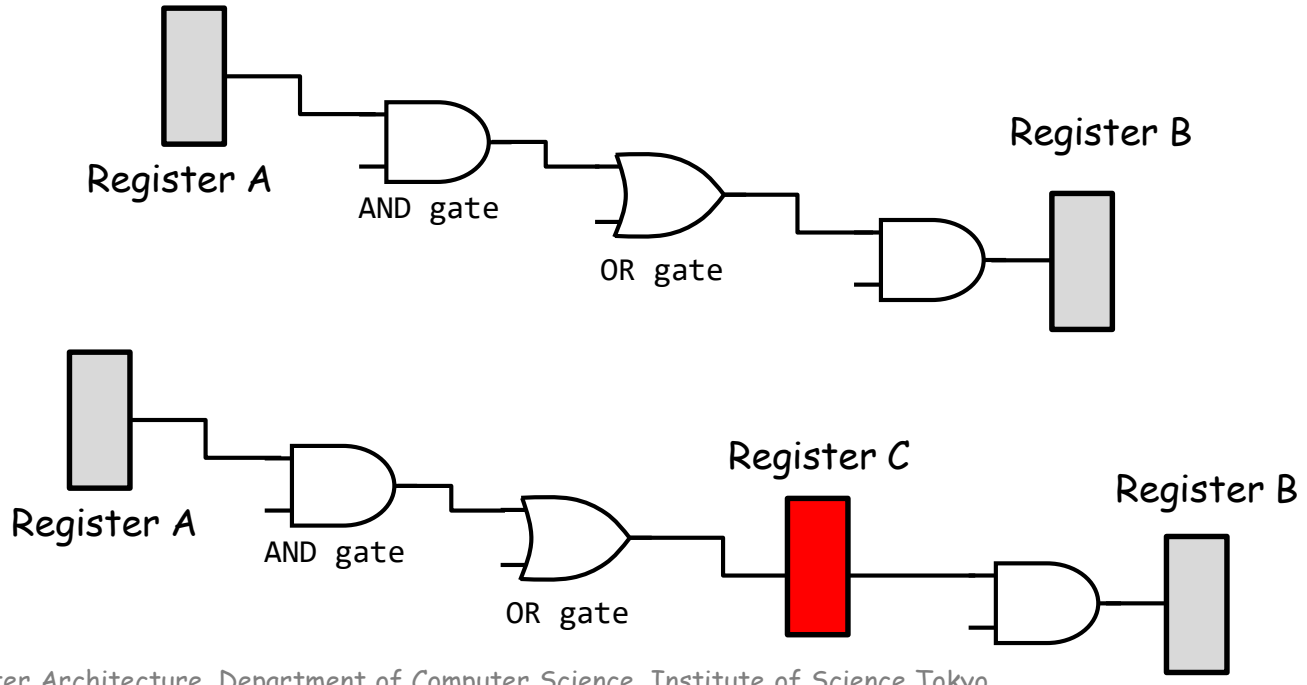
- supporting add, addi, lw, sw, bne instructions




```
module m_proc5s(w_clk);
  input wire w_clk;
  wire [31:0] w_npc, w_ir, w_imm, w_r1, w_r2, w_s2, w_rt;
  wire [31:0] w_alu, w_ldd, w_tpc, w_pcin;
  wire w_tkn;
  reg [31:0] r_pc=0; // m1
  assign w_pcin = (w_b & w_tkn) ?  w_tpc : w_npc; // m11
  assign w_npc  = r_pc + 32'h4; // m2
  m_am_imem m3 (r_pc, w_ir);
  wire w_r, w_i, w_s, w_b, w_u, w_j, w_ld;
  m_gen_imm m4 (w_ir, w_imm, w_r, w_i,
                w_s, w_b, w_u, w_j, w_ld);
  m_RF m5 (w_clk, w_ir[19:15], w_ir[24:20], w_r1, w_r2,
           w_ir[11:7], !w_s & !w_b, w_rt);
  assign w_tpc = r_pc + w_imm; // m6
  assign w_s2 = (!w_r & !w_b) ? w_imm : w_r2; // m7
  m_alu m8 (w_r1, w_s2, w_alu, w_tkn);
  m_am_dmem m9 (w_clk, w_alu, w_s, w_r2, w_ldd);
  assign w_rt = (w_ld) ? w_ldd : w_alu; // m10
  always @(posedge w_clk) r_pc <= w_pcin;
endmodule
```

Clock rate is mainly determined by

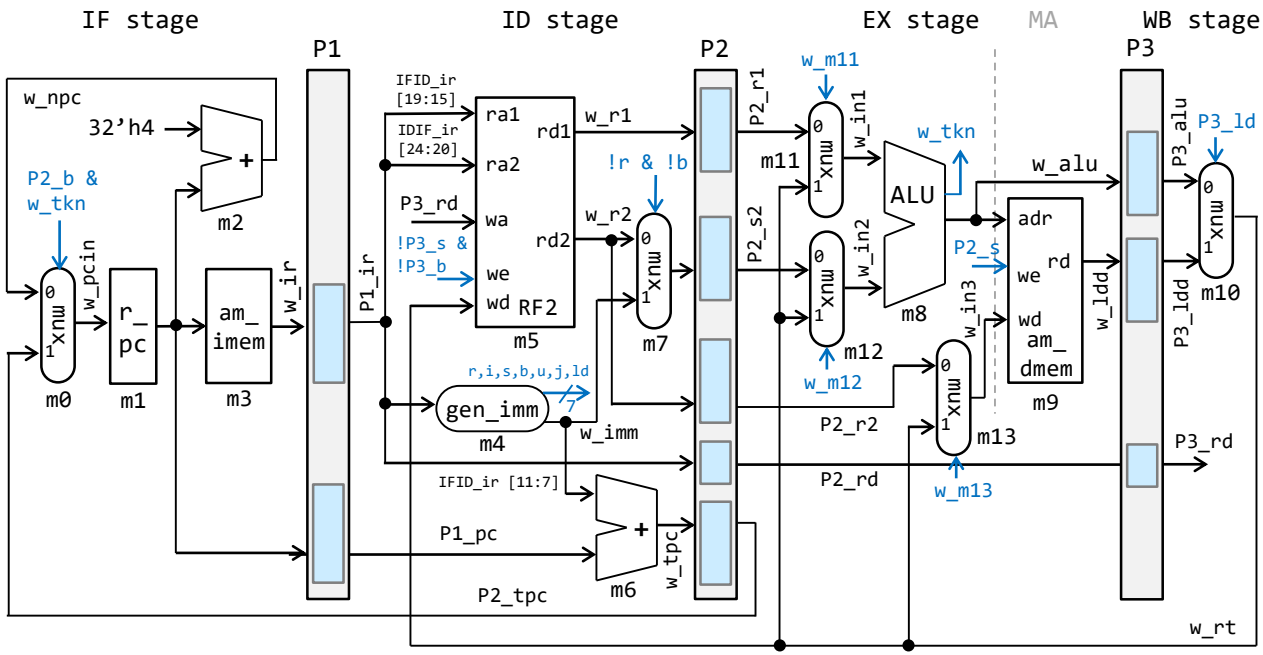
- Switching speed of gates (transistors)
- **The number of levels of gates**
  - The maximum number of gates cascaded in series in any combinational logics.
  - In this example, the number of levels of gates is 3.
- Wiring delay and fanout



プロセッサが命令を処理するための5つのステップ

- IF (Instruction Fetch)
  メモリから命令をフェッチする.
- ID (Instruction Decode)
  命令をデコード(解読)しながら, レジスタの値を読み出す.
- EX (Execution)
  命令操作の実行またはアドレスの生成を行う.
- MA (Memory Access)
  必要であれば, データ・メモリ中のオペランドにアクセスする.
- WB (Write Back)
  必要であれば, 結果をレジスタに書き込む.

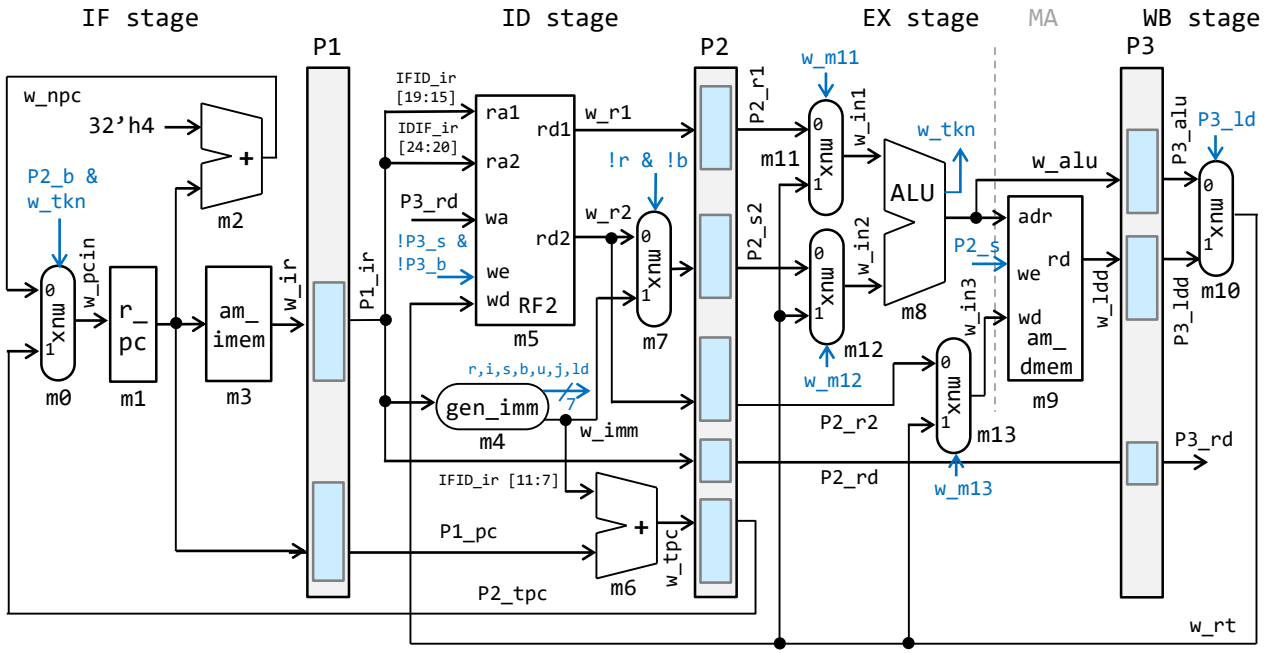
proc8: 4-stage pipelining processor

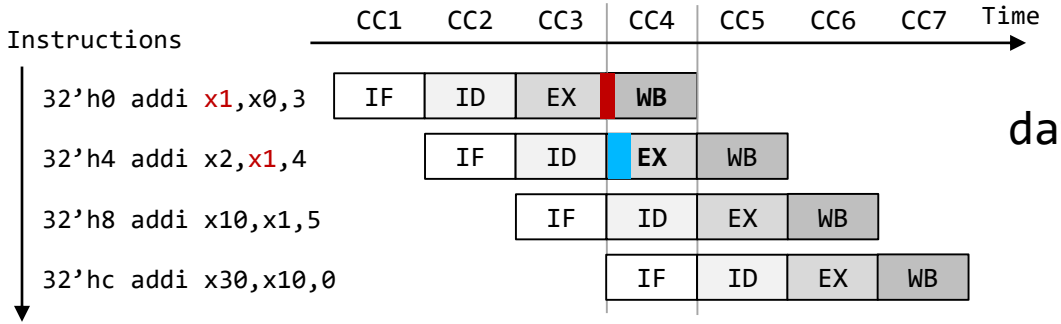


Source code is abailable here.

https://gihyo.jp/book/2024/978-4-297-14008-3/support

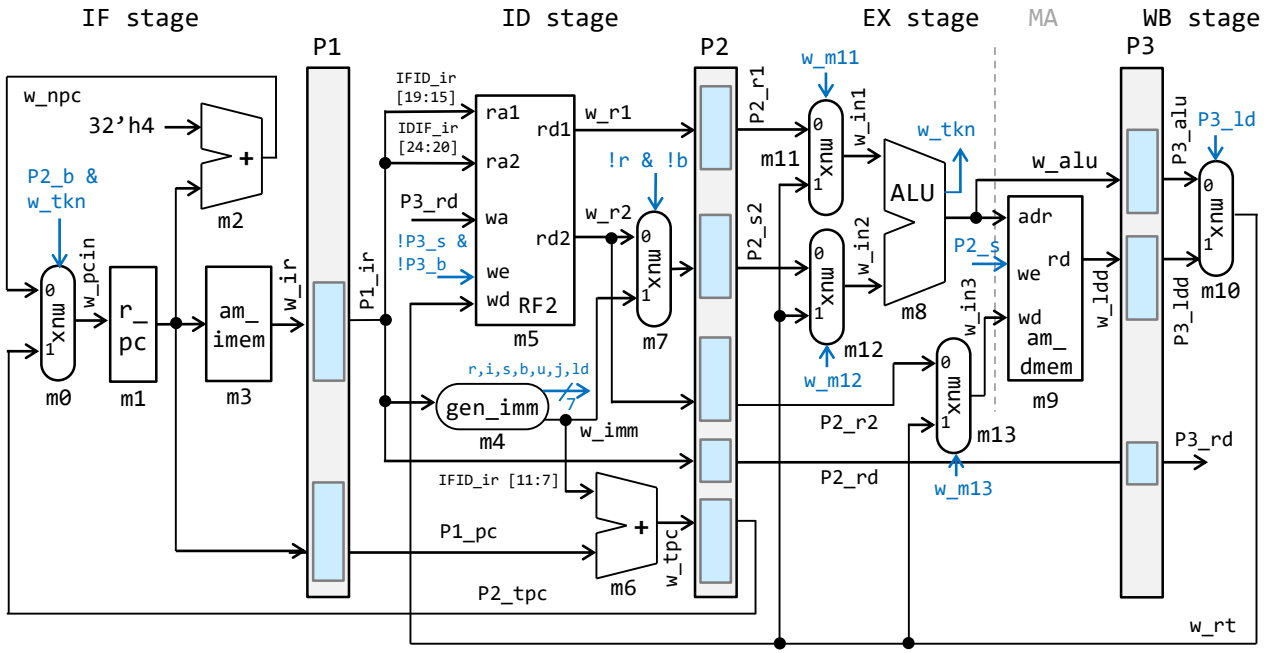
proc8: 4-stage pipelining processor

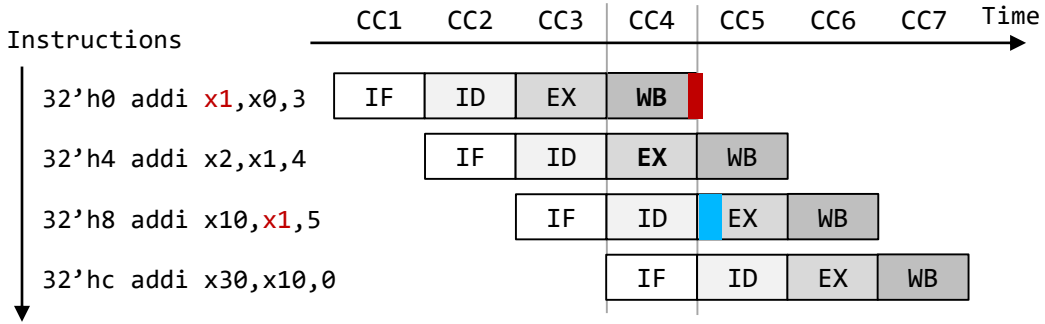




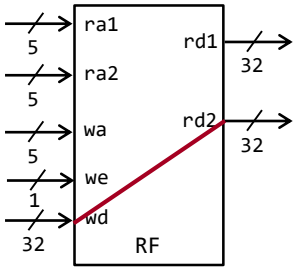
data forwarding

proc8: 4-stage pipelining processor

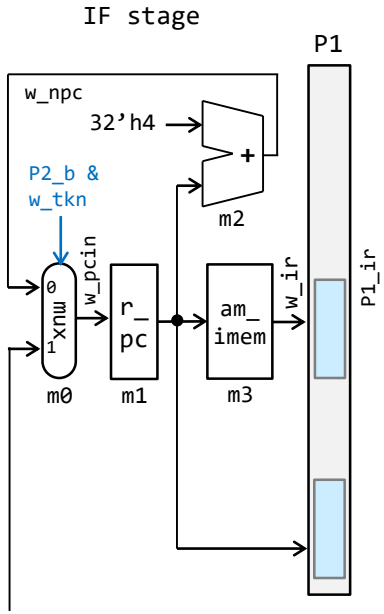




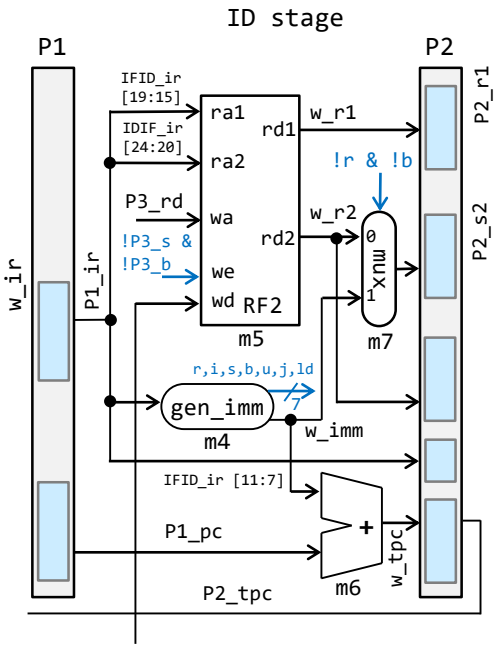
RF2 (Register File) with bypassing



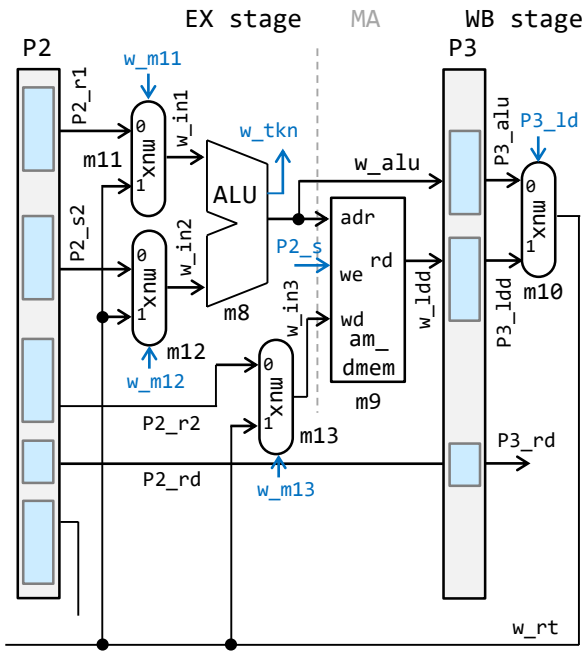

```
module m_RF2(w_clk, w_radr1, w_radr2, w_rd1, w_rd2, w_wadr, w_we, w_wd);
  input  wire w_clk, w_we;
  input  wire [4:0] w_radr1, w_radr2, w_wadr;
  output wire [31:0] w_rd1, w_rd2;
  input  wire [31:0] w_wd;
  reg [31:0] mem [0:31];
  wire w_bp1 = (w_we & w_radr1==w_wadr);
  wire w_bp2 = (w_we & w_radr2==w_wadr);
  assign w_rd1 = (w_radr1==5'd0) ? 32'd0 : (w_bp1) ? w_wd : mem[w_radr1];
  assign w_rd2 = (w_radr2==5'd0) ? 32'd0 : (w_bp2) ? w_wd : mem[w_radr2];
  always @(posedge w_clk) if (w_we) mem[w_wadr] <= w_wd;
  always @(posedge w_clk) if (w_we & w_wadr==5'd30) $finish;
  integer i; initial for (i=0; i<32; i=i+1) mem[i] = 32'd0;
endmodule
```




```
module m_proc8s(w_clk);
  input wire w_clk;
  reg [31:0] P1_ir=32'h13, P1_pc=0, P2_pc=0, P3_pc=0;
  reg [31:0] P2_r1=0, P2_s2=0, P2_r2=0, P2_tpc=0;
  reg [31:0] P3_alu=0, P3_ldd=0;
  reg P2_r=0, P2_s=0, P2_b=0, P2_ld=0, P3_s=0, P3_b=0, P3_ld=0;
  reg [4:0] P2_rd=0, P2_rs1=0, P2_rs2=0, P3_rd=0;
  reg P1_v=0, P2_v=0, P3_v=0;
  wire [31:0] w_npc, w_ir, w_imm, w_r1, w_r2, w_s2, w_rt;
  wire [31:0] w_alu, w_ldd, w_tpc, w_pcin, w_in1, w_in2, w_in3;
  wire w_r, w_i, w_s, w_b, w_u, w_j, w_ld, w_tkn;
  reg [31:0] r_pc = 0; // m1
  wire w_miss = P2_b & w_tkn & P2_v;
  assign w_pcin = (w_miss) ? P2_tpc : w_npc; // m0
  assign w_npc = r_pc + 32'h4; // m2
  m_am_imem m3 (r_pc, w_ir);
  m_gen_imm m4 (P1_ir, w_imm, w_r, w_i, w_s, w_b, w_u, w_j, w_ld);
  m_RF2 m5 (w_clk, P1_ir[19:15], P1_ir[24:20], w_r1, w_r2,
            P3_rd, !P3_s & !P3_b & P3_v, w_rt);
  assign w_tpc = P1_pc + w_imm; // m6
  assign w_s2 = (!w_r & !w_b) ? w_imm : w_r2; // m7
  always @(posedge w_clk) begin
    {P1_v, P2_v, P3_v} <= {!w_miss, !w_miss & P1_v, P2_v};
    {r_pc, P1_ir, P1_pc, P2_pc} <= {w_pcin, w_ir, r_pc, P1_pc};
    {P2_r1, P2_r2, P2_s2, P2_tpc} <= {w_r1, w_r2, w_s2, w_tpc};
    {P2_r, P2_s, P2_b, P2_ld} <= {w_r, w_s, w_b, w_ld};
    {P2_rs2, P2_rs1, P2_rd} <= {P1_ir[24:15], P1_ir[11:7]};
    {P3_pc, P3_ld} <= {P2_pc, P2_ld};
    {P3_alu, P3_ldd, P3_rd} <= {w_alu, w_ldd, P2_rd};
  end
  assign w_alu = w_in1 + w_in2;   // m8
  assign w_tkn = w_in1 != w_in2;  // m8
  m_am_dmem m9 (w_clk, w_alu, P2_s & P2_v, w_in3, w_ldd);
  assign w_rt = (P3_ld) ? P3_ldd : P3_alu; // m10
  assign w_in1 = (|P3_rd & P2_rs1==P3_rd) ? w_rt : P2_r1; // m11
  assign w_in3 = (!P3_rd & P2_rs2==P3_rd) ? w_rt : P2_r2; // m13
  assign w_in2 = (|P3_rd & (P2_r|P2_b) & P2_rs2==P3_rd) ? w_rt : P2_s2;// m12
endmodule
```



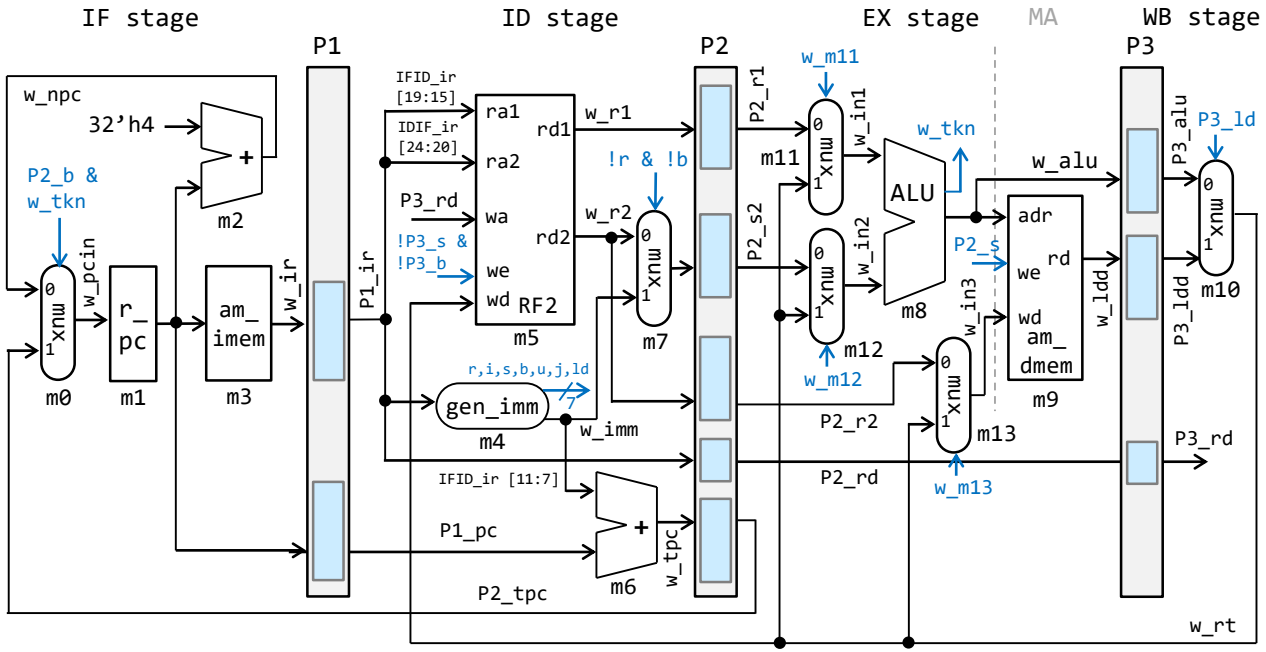

```
module m_proc8s(w_clk);
  input wire w_clk;
  reg [31:0] P1_ir=32'h13, P1_pc=0, P2_pc=0, P3_pc=0;
  reg [31:0] P2_r1=0, P2_s2=0, P2_r2=0, P2_tpc=0;
  reg [31:0] P3_alu=0, P3_ldd=0;
  reg P2_r=0, P2_s=0, P2_b=0, P2_ld=0, P3_s=0, P3_b=0, P3_ld=0;
  reg [4:0] P2_rd=0, P2_rs1=0, P2_rs2=0, P3_rd=0;
  reg P1_v=0, P2_v=0, P3_v=0;
  wire [31:0] w_npc, w_ir, w_imm, w_r1, w_r2, w_s2, w_rt;
  wire [31:0] w_alu, w_ldd, w_tpc, w_pcin, w_in1, w_in2, w_in3;
  wire w_r, w_i, w_s, w_b, w_u, w_j, w_ld, w_tkn;
  reg [31:0] r_pc = 0; // m1
  wire w_miss = P2_b & w_tkn & P2_v;
  assign w_pcin = (w_miss) ? P2_tpc : w_npc; // m0
  assign w_npc = r_pc + 32'h4; // m2
  m_am_imem m3 (r_pc, w_ir);
  m_gen_imm m4 (P1_ir, w_imm, w_r, w_i, w_s, w_b, w_u, w_j, w_ld);
  m_RF2 m5 (w_clk, P1_ir[19:15], P1_ir[24:20], w_r1, w_r2,
            P3_rd, !P3_s & !P3_b & P3_v, w_rt);
  assign w_tpc = P1_pc + w_imm; // m6
  assign w_s2 = (!w_r & !w_b) ? w_imm : w_r2; // m7
  always @(posedge w_clk) begin
    {P1_v, P2_v, P3_v} <= {!w_miss, !w_miss & P1_v, P2_v};
    {r_pc, P1_ir, P1_pc, P2_pc} <= {w_pcin, w_ir, r_pc, P1_pc};
    {P2_r1, P2_r2, P2_s2, P2_tpc} <= {w_r1, w_r2, w_s2, w_tpc};
    {P2_r, P2_s, P2_b, P2_ld} <= {w_r, w_s, w_b, w_ld};
    {P2_rs2, P2_rs1, P2_rd} <= {P1_ir[24:15], P1_ir[11:7]};
    {P3_pc, P3_ld} <= {P2_pc, P2_ld};
    {P3_alu, P3_ldd, P3_rd} <= {w_alu, w_ldd, P2_rd};
  end
  assign w_alu = w_in1 + w_in2;   // m8
  assign w_tkn = w_in1 != w_in2;  // m8
  m_am_dmem m9 (w_clk, w_alu, P2_s & P2_v, w_in3, w_ldd);
  assign w_rt = (P3_ld) ? P3_ldd : P3_alu; // m10
  assign w_in1 = (|P3_rd & P2_rs1==P3_rd) ? w_rt : P2_r1; // m11
  assign w_in3 = (!P3_rd & P2_rs2==P3_rd) ? w_rt : P2_r2; // m13
  assign w_in2 = (|P3_rd & (P2_r|P2_b) & P2_rs2==P3_rd) ? w_rt : P2_s2;// m12
endmodule
```



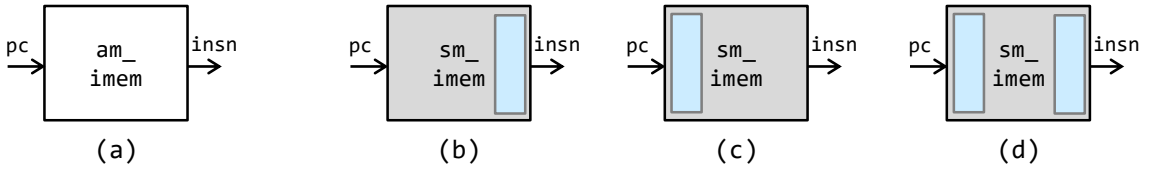

```
module m_proc8s(w_clk);
  input wire w_clk;
  reg [31:0] P1_ir=32'h13, P1_pc=0, P2_pc=0, P3_pc=0;
  reg [31:0] P2_r1=0, P2_s2=0, P2_r2=0, P2_tpc=0;
  reg [31:0] P3_alu=0, P3_ldd=0;
  reg P2_r=0, P2_s=0, P2_b=0, P2_ld=0, P3_s=0, P3_b=0, P3_ld=0;
  reg [4:0] P2_rd=0, P2_rs1=0, P2_rs2=0, P3_rd=0;
  reg P1_v=0, P2_v=0, P3_v=0;
  wire [31:0] w_npc, w_ir, w_imm, w_r1, w_r2, w_s2, w_rt;
  wire [31:0] w_alu, w_ldd, w_tpc, w_pcin, w_in1, w_in2, w_in3;
  wire w_r, w_i, w_s, w_b, w_u, w_j, w_ld, w_tkn;
  reg [31:0] r_pc = 0; // m1
  wire w_miss = P2_b & w_tkn & P2_v;
  assign w_pcin = (w_miss) ? P2_tpc : w_npc; // m0
  assign w_npc = r_pc + 32'h4; // m2
  m_am_imem m3 (r_pc, w_ir);
  m_gen_imm m4 (P1_ir, w_imm, w_r, w_i, w_s, w_b, w_u, w_j, w_ld);
  m_RF2 m5 (w_clk, P1_ir[19:15], P1_ir[24:20], w_r1, w_r2,
            P3_rd, !P3_s & !P3_b & P3_v, w_rt);
  assign w_tpc = P1_pc + w_imm; // m6
  assign w_s2 = (!w_r & !w_b) ? w_imm : w_r2; // m7
  always @(posedge w_clk) begin
    {P1_v, P2_v, P3_v} <= {!w_miss, !w_miss & P1_v, P2_v};
    {r_pc, P1_ir, P1_pc, P2_pc} <= {w_pcin, w_ir, r_pc, P1_pc};
    {P2_r1, P2_r2, P2_s2, P2_tpc} <= {w_r1, w_r2, w_s2, w_tpc};
    {P2_r, P2_s, P2_b, P2_ld} <= {w_r, w_s, w_b, w_ld};
    {P2_rs2, P2_rs1, P2_rd} <= {P1_ir[24:15], P1_ir[11:7]};
    {P3_pc, P3_ld} <= {P2_pc, P2_ld};
    {P3_alu, P3_ldd, P3_rd} <= {w_alu, w_ldd, P2_rd};
  end
  assign w_alu = w_in1 + w_in2;   // m8
  assign w_tkn = w_in1 != w_in2;  // m8
  m_am_dmem m9 (w_clk, w_alu, P2_s & P2_v, w_in3, w_ldd);
  assign w_rt = (P3_ld) ? P3_ldd : P3_alu; // m10
  assign w_in1 = (|P3_rd & P2_rs1==P3_rd) ? w_rt : P2_r1; // m11
  assign w_in3 = (!P3_rd & P2_rs2==P3_rd) ? w_rt : P2_r2; // m13
  assign w_in2 = (|P3_rd & (P2_r|P2_b) & P2_rs2==P3_rd) ? w_rt : P2_s2;// m12
endmodule
```

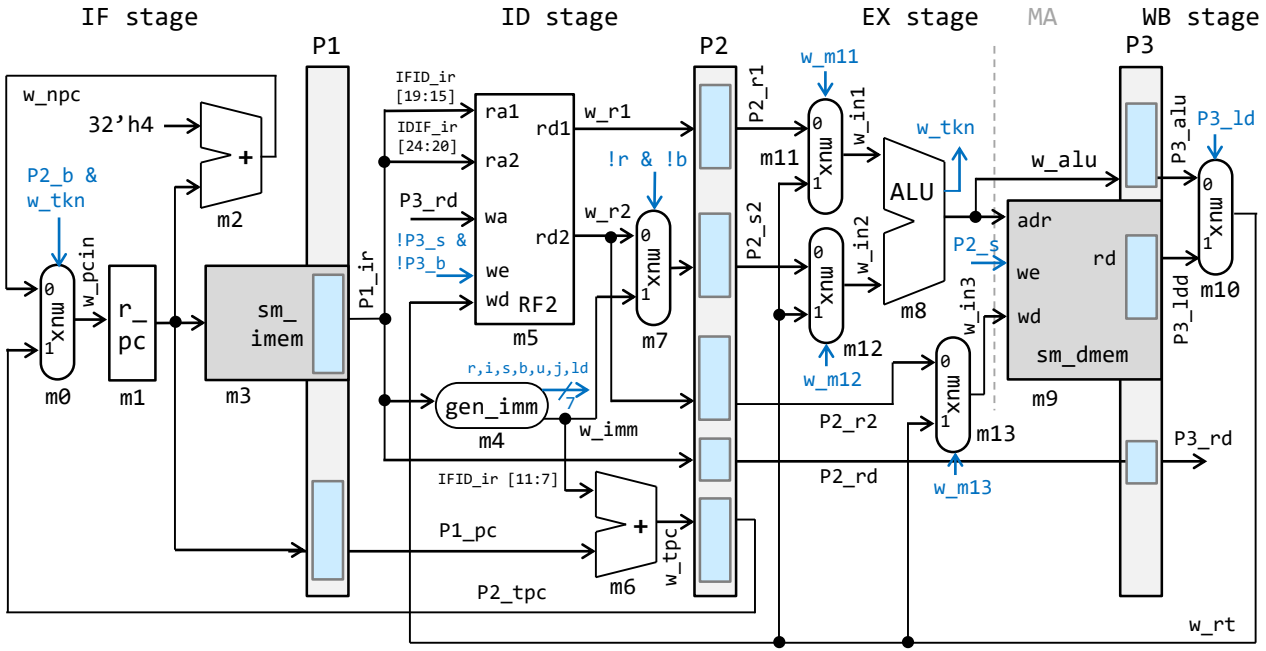
```
module m_proc8s(w_clk);
  input wire w_clk;
  reg [31:0] P1_ir=32'h13, P1_pc=0, P2_pc=0, P3_pc=0;
  reg [31:0] P2_r1=0, P2_s2=0, P2_r2=0, P2_tpc=0;
  reg [31:0] P3_alu=0, P3_ldd=0;
  reg P2_r=0, P2_s=0, P2_b=0, P2_ld=0, P3_s=0, P3_b=0, P3_ld=0;
  reg [4:0] P2_rd=0, P2_rs1=0, P2_rs2=0, P3_rd=0;
  reg P1_v=0, P2_v=0, P3_v=0;
  wire [31:0] w_npc, w_ir, w_imm, w_r1, w_r2, w_s2, w_rt;
  wire [31:0] w_alu, w_ldd, w_tpc, w_pcin, w_in1, w_in2, w_in3;
  wire w_r, w_i, w_s, w_b, w_u, w_j, w_ld, w_tkn;
  reg [31:0] r_pc = 0; // m1
  wire w_miss = P2_b & w_tkn & P2_v;
  assign w_pcin = (w_miss) ? P2_tpc : w_npc; // m0
  assign w_npc = r_pc + 32'h4; // m2
  m_am_imem m3 (r_pc, w_ir);
  m_gen_imm m4 (P1_ir, w_imm, w_r, w_i, w_s, w_b, w_u, w_j, w_ld);
  m_RF2 m5 (w_clk, P1_ir[19:15], P1_ir[24:20], w_r1, w_r2,
            P3_rd, !P3_s & !P3_b & P3_v, w_rt);
  assign w_tpc = P1_pc + w_imm; // m6
  assign w_s2 = (!w_r & !w_b) ? w_imm : w_r2; // m7
  always @(posedge w_clk) begin
    {P1_v, P2_v, P3_v} <= {!w_miss, !w_miss & P1_v, P2_v};
    {r_pc, P1_ir, P1_pc, P2_pc} <= {w_pcin, w_ir, r_pc, P1_pc};
    {P2_r1, P2_r2, P2_s2, P2_tpc} <= {w_r1, w_r2, w_s2, w_tpc};
    {P2_r, P2_s, P2_b, P2_ld} <= {w_r, w_s, w_b, w_ld};
    {P2_rs2, P2_rs1, P2_rd} <= {P1_ir[24:15], P1_ir[11:7]};
    {P3_pc, P3_ld} <= {P2_pc, P2_ld};
    {P3_alu, P3_ldd, P3_rd} <= {w_alu, w_ldd, P2_rd};
  end
  assign w_alu = w_in1 + w_in2;   // m8
  assign w_tkn = w_in1 != w_in2;  // m8
  m_am_dmem m9 (w_clk, w_alu, P2_s & P2_v, w_in3, w_ldd);
  assign w_rt = (P3_ld) ? P3_ldd : P3_alu; // m10
  assign w_in1 = (|P3_rd & P2_rs1==P3_rd) ? w_rt : P2_r1; // m11
  assign w_in3 = (!P3_rd & P2_rs2==P3_rd) ? w_rt : P2_r2; // m13
  assign w_in2 = (|P3_rd & (P2_r|P2_b) & P2_rs2==P3_rd) ? w_rt : P2_s2;// m12
endmodule
```

proc8: 4-stage pipelining processor



proc8: 4-stage pipelining processor

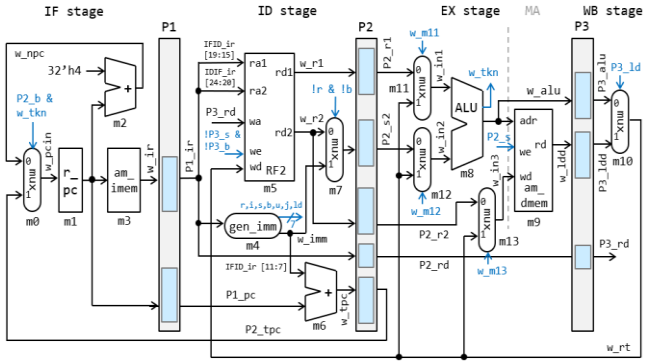




Processor simulation

```
// 00    addi x1,x0,3
// 04    add  x2,x1,x1
// 08    addi x10,x2,5
// 0c    HALT
```

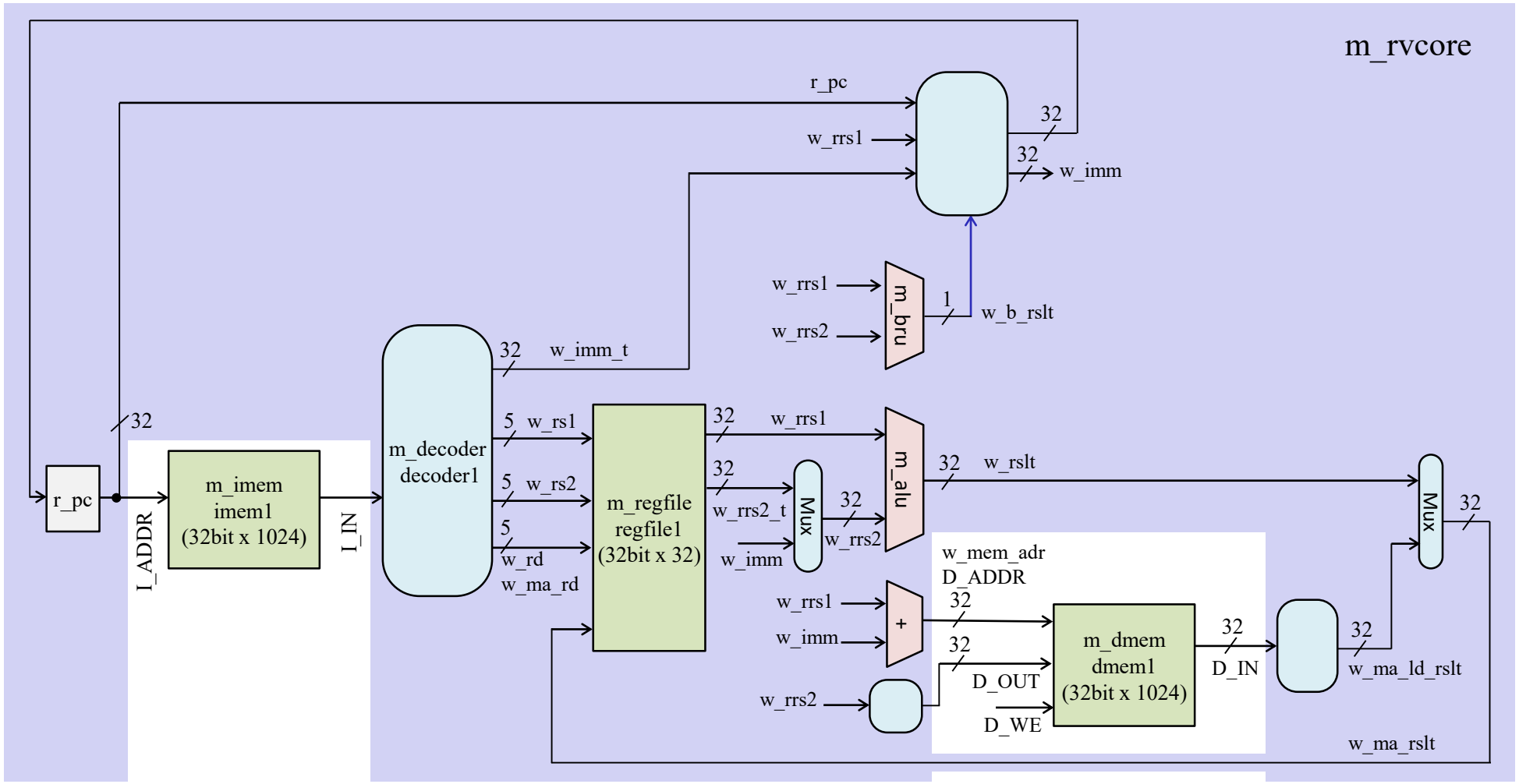
```
$ iverilog -DP1 proc8s.v
$ ./a.out

CC01 00000000 00000000 00000000 00000000          0          0          0
CC02 00000004 00000000 00000000 00000000          0          0          0
CC03 00000008 00000004 00000000 00000000          0          3          3
CC04 0000000c 00000008 00000004 00000000          3          3          6
CC05 00000010 0000000c 00000008 00000004          6          5         11
CC06 00000014 00000010 0000000c 00000008         11          0         11
CC07 00000018 00000014 00000010 0000000c          0          0          0
```




```
module m_sim(w_clk, w_cc); /* please wrap me by m_top_wrapper */
  input wire w_clk; input wire [31:0] w_cc;
  m_proc8s m (w_clk);
  initial begin
    `define MM m.m3.mem
    `include "asm.txt"
  end
  initial #99 forever #100 $display("CC%02d %h %h %h %h %d %d %d",
    w_cc, m.r_pc, m.P1_pc, m.P2_pc, m.P3_pc,
    m.w_in1, m.w_in2, m.w_alu);
endmodule
```

module m_rvcore (RV32I, single-cycle processor)

- 50MHz operating frequency



RVCore_Base, RISC-V RV32I Bas

```
/*************************************************************************************************/
/* RVCore_Base, RISC-V RV32I Baseline Version v2024-10-22a          ArchLab, Science Tokyo */
/*************************************************************************************************/
`default_nettype none

`define VERIFY                  // define this to generate verify.txt
`define MEM_TXT   "sample1.txt" // file name of memory initialization file
`define MEM_SIZE  4096          // 4KB instructio memory and 4KB data memory
`define START_PC  0             // initial PC value
`define UART_CNT  50            // UART wait count, 50MHz / 500 = 1Mbaud
`define D_I_TYPE  0
`define D_R_TYPE  1
`define D_S_TYPE  2
`define D_B_TYPE  3
`define D_J_TYPE  4
`define D_U_TYPE  5
`define D_JA__IS  6
`define D_LD__IS  7
`define D_MUL_IS  8
`define D_DIV_IS  9
```

```
    /*************************************************************************************************/
`ifdef VERIFY
    initial $write("¥n== VERIFY is defined and generate verify.txt¥n");
    integer fd;
    initial fd = $fopen("verify.txt", "w");
    reg [31:0] r_tc = 1;
    always @(posedge w_clk) if(!r_rst) r_tc <= r_tc + 1;
    integer i, j;
    always @(posedge w_clk) if(!r_rst) begin
        $fwrite(fd, "%08d %08x %08x¥n", r_tc, r_pc, w_ir);
        for (i=0; i<4; i=i+1) for (j=0; j<8; j=j+1) begin
            $fwrite(fd, "%08x", ((i*8+j == [27'd0, w_rd]) && (i*8+j != 0)) ?
                    w_ma_rslt : regfile1.mem[i * 8 + j]);
            $fwrite(fd, "%s", (j != 7 ? " " : "¥n"));
        end
    end
`endif
endmodule
```

```
/*************************************************************************************************/
module m_rvcore ( ///// RVCore Simple Version
    input  wire        w_clk, w_rst,
    input  wire [31:0] I_IN, D_IN,
    output wire [31:0] I_ADDR, D_ADDR,
    output wire [31:0] D_OUT,
    output wire  [3:0] D_WE
);
    reg r_rst = 1;
    always @(posedge w_clk) r_rst <= w_rst;

    reg [31:0] r_pc = `START_PC;
    always @(posedge w_clk) r_pc <= (r_rst) ? `START_PC : (w_b_rslt) ? w_tkn_pc : r_pc+4;
    assign I_ADDR = r_pc;

    wire [31:0] w_ir = I_IN;
    wire [31:0] w_rrs1, w_rrs2_t, w_imm_t, w_rslt;
    wire [ 4:0] w_rs1, w_rs2, w_rd;
    wire        w_op_im;
    wire [9:0]  w_itype;
    wire [10:0] w_alu_c;
    wire [6:0]  w_bru_c;
    wire        w_b_rslt;
    wire        w_ja   = w_itype[6];
    wire [4:0]  w_op   = w_ir[6:2];
    wire [31:0] w_imm  = (w_ja) ? r_pc+4 : (w_op==5'b00101) ? r_pc+w_imm_t : w_imm_t;
    wire [31:0] w_rrs2 = (w_op_im) ? w_imm : w_rrs2_t;

    m_decode decode1 (w_ir, w_rd, w_rs1, w_rs2, w_op_im, w_itype, w_alu_c, w_bru_c, w_imm_t);

    m_regfile regfile1 (w_clk, w_rs1, w_rs2, w_rrs1, w_rrs2_t, w_ma_rd, w_ma_rslt);

    m_alu alu1 (w_rrs1, w_rrs2, w_alu_c, w_rslt);

    /***** for branch and jump instructions *****/
    wire [31:0] w_tkn_pc = (w_ir[6:2]==5'b11001) ? w_rrs1+w_imm_t : r_pc+w_imm_t;
    m_bru bru0(w_rrs1, w_rrs2, w_bru_c, w_b_rslt);

    /***** for load and store instructions *****/
    wire [2:0]  w_fct3 = w_ir[14:12];
    wire [31:0] w_mem_adr = w_rrs1 + w_imm;
    wire        w_we0 = (w_itype[`D_S_TYPE] & w_fct3[1:0]==0);
    wire        w_we1 = (w_itype[`D_S_TYPE] & w_fct3[0]);
    wire        w_we2 = (w_itype[`D_S_TYPE] & w_fct3[1]);
    wire [3:0]  w_we_sb = (w_we0) ? 4'b0001 << D_ADDR[1:0]       : 0;
    wire [3:0]  w_we_sh = (w_we1) ? 4'b0011 << {D_ADDR[1], 1'b0} : 0;
    wire [3:0]  w_we_sw = (w_we2) ? 4'b1111                      : 0;

    assign D_WE   = (w_we_sh | w_we_sw | w_we_sb);
    assign D_ADDR = w_mem_adr;
    assign D_OUT  = (w_fct3[0]) ? {2{w_rrs2[15:0]}} :
                    (w_fct3[1]) ? w_rrs2 : {4{w_rrs2[7:0]}};

    /***** for load instructions *****/
    wire [15:0] w_ma_lh_t = (w_mem_adr[1]) ? D_IN[31:16]     : D_IN[15:0];     // align
    wire [7:0]  w_ma_lb_t = (w_mem_adr[0]) ? w_ma_lh_t[15:8] : w_ma_lh_t[7:0]; // align
    wire [31:0] w_ma_lb  = (w_fct3==3'b000) ? {[24{w_ma_lb_t[7]}},  w_ma_lb_t[7:0]}  : 0;
    wire [31:0] w_ma_lbu = (w_fct3==3'b100) ? {24'd0,               w_ma_lb_t[7:0]}  : 0;
    wire [31:0] w_ma_lh  = (w_fct3==3'b001) ? {[16{w_ma_lh_t[15]}], w_ma_lh_t[15:0]} : 0;
    wire [31:0] w_ma_lhu = (w_fct3==3'b101) ? {16'd0,               w_ma_lh_t[15:0]} : 0;
    wire [31:0] w_ma_lw  = (w_fct3==3'b010) ? D_IN                                   : 0;
    wire [31:0] w_ma_ld_rslt = w_ma_lb | w_ma_lbu | w_ma_lh | w_ma_lhu | w_ma_lw;

    wire [31:0] w_ma_rslt_t = (w_itype[`D_LD__IS]) ? w_ma_ld_rslt : w_rslt;

    wire [4:0]  w_ma_rd   = w_rd ;
    wire [31:0] w_ma_rslt = (w_rd!=0) ? w_ma_rslt_t : 0;
```

proc.v

RVCore_Base, RISC-V RV32I Baseline Version v2024-10-22a

```
    /*************************************************************************************/
`ifdef VERIFY
    initial $write("¥n== VERIFY is defined and generate verify.txt¥n");
    integer fd;
    initial fd = $fopen("verify.txt", "w");
    reg [31:0] r_tc = 1;
    always @(posedge w_clk) if(!r_rst) r_tc <= r_tc + 1;
    integer i, j;
    always @(posedge w_clk) if(!r_rst) begin
        $fwrite(fd, "%08d %08x %08x¥n", r_tc, r_pc, w_ir);
        for (i=0; i<4; i=i+1) for (j=0; j<8; j=j+1) begin
            $fwrite(fd, "%08x", ((i*8+j == {27'd0, w_rd}) && (i*8+j != 0)) ?
                    w_ma_rslt : regfile1.mem[i * 8 + j]);
            $fwrite(fd, "%s", (j != 7 ? " " : "¥n"));
        end
    end
`endif
endmodule
```

proc.v

```
00000090 000000dc 00150513
00000000 00000968 00000f20 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000bc7 00000000 00000000 40008000 00010000 00000075
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000091 000000e0 fe0798e3
00000000 00000968 00000f20 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000bc7 00000000 00000000 40008000 00010000 00000075
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000092 000000d0 00e7e7b3
00000000 00000968 00000f20 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000bc7 00000000 00000000 40008000 00010000 00010075
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000093 000000d4 00f6a023
00000000 00000968 00000f20 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000bc7 00000000 00000000 40008000 00010000 00010075
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000094 000000d8 00154783
00000000 00000968 00000f20 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000bc7 00000000 00000000 40008000 00010000 00000065
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000095 000000dc 00150513
00000000 00000968 00000f20 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000bc8 00000000 00000000 40008000 00010000 00000065
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000096 000000e0 fe0798e3
```

verify.txt

RVCore_Base, RISC-V RV32I Baseline Version v2024-10-22a

proc.v

```
/*****************************************************************
module m_imem (
    input  wire        w_clk,
    input  wire        w_we,
    input  wire [9:0]  w_adr,
    input  wire [31:0] w_din,
    output wire [31:0] w_dout
);
    reg [31:0] mem [0:1023];
`ifndef SYNTHESIS
initial begin
`include `MEM_TXT
end
`endif

    always @(posedge w_clk) if (w_we) mem[w_adr] <= w_din;
    assign w_dout = mem[w_adr];
endmodule

/*****************************************************************
module m_dmem (
    input  wire        w_clk,
    input  wire [3:0]  w_we,
    input  wire [9:0]  w_adr,
    input  wire [31:0] w_din,
    output wire [31:0] w_dout
);
    reg [31:0] mem [0:1023];
`ifndef SYNTHESIS
initial begin
`include `MEM_TXT
end
`endif

    always @(posedge w_clk) begin
        if (w_we[0]) mem[w_adr][ 7: 0] <= w_din[ 7: 0];
        if (w_we[1]) mem[w_adr][15: 8] <= w_din[15: 8];
        if (w_we[2]) mem[w_adr][23:16] <= w_din[23:16];
        if (w_we[3]) mem[w_adr][31:24] <= w_din[31:24];
    end
    assign w_dout = mem[w_adr];
endmodule
```

sample1.txt

```
mem[   0] = 32'h00000013;
mem[   1] = 32'h00000033;
mem[   2] = 32'h000000b3;
mem[   3] = 32'h00000133;
mem[   4] = 32'h000001b3;
mem[   5] = 32'h00000233;
mem[   6] = 32'h000002b3;
mem[   7] = 32'h00000333;
mem[   8] = 32'h000003b3;
mem[   9] = 32'h00000433;
mem[  10] = 32'h000004b3;
mem[  11] = 32'h00000533;
mem[  12] = 32'h000005b3;
mem[  13] = 32'h00000633;
mem[  14] = 32'h000006b3;
mem[  15] = 32'h00000733;
mem[  16] = 32'h000007b3;
mem[  17] = 32'h00000833;
mem[  18] = 32'h000008b3;
mem[  19] = 32'h00000933;
mem[  20] = 32'h000009b3;
mem[  21] = 32'h00000a33;
mem[  22] = 32'h00000ab3;
mem[  23] = 32'h00000b33;
mem[  24] = 32'h00000bb3;
mem[  25] = 32'h00000c33;
mem[  26] = 32'h00000cb3;
mem[  27] = 32'h00000d33;
mem[  28] = 32'h00000db3;
mem[  29] = 32'h00000e33;
mem[  30] = 32'h00000eb3;
mem[  31] = 32'h00000f33;
```

RVCore_Base, RISC-V RV32I Baseline Version v2024-10-22a

```
module m_main (
    input  wire w_clk,      // 100MHz clock signal
    input  wire w_uart_rx,  // UART rx, data line from PC to FPGA
    output wire w_uart_tx   // UART tx, data line from FPGA to PC
);
    wire w_clk50m, locked;
    clk_wiz_0 clk_wiz1 (w_clk50m, locked, w_clk);

    reg r_rst = 1;
    always @(posedge w_clk50m) r_rst <= ~locked;

    reg core_rst = 1;
    always @(posedge w_clk50m) core_rst <= (r_rst | r_done==0);

    wire w_en;
    wire [7:0] w_char;
    m_uart_rx uart_rx1 (w_clk50m, w_uart_rx, w_char, w_en);

    reg r_done = `DONE_INIT;
    reg r_we = 0;
    reg [31:0] r_wcnt = 0, r_adr = 0, r_data = 0;
    always @(posedge w_clk50m) if (r_done==0) begin
        if(w_en) begin
            r_adr  <= r_wcnt;
            r_data <= {w_char, r_data[31:8]};
            r_we   <= (r_wcnt[1:0]==3);
            r_wcnt <= r_wcnt + 1;
        end else begin
            r_we <= 0;
            if (r_wcnt>=`MEM_SIZE) r_done <= 1;
        end
    end

    wire [3:0]  D_WE;
    wire [31:0] I_IN, I_ADDR, D_IN, D_OUT, D_ADDR;
    m_rvcore rvcore1 (w_clk50m, core_rst, I_IN, D_IN, I_ADDR, D_ADDR, D_OUT, D_WE);

    m_imem imem1 (w_clk50m, r_we, r_done ? I_ADDR[11:2] : r_adr[11:2], r_data, I_IN);
    m_dmem dmem1 (w_clk50m, {4{r_we}} | D_WE, r_done ? D_ADDR[11:2] : r_adr[11:2],
                  r_done ? D_OUT : r_data, D_IN);

    wire [1:0] tohost_cmd  = ((D_ADDR==32'h40008000) && D_WE==4'hf) ? D_OUT[17:16] : 0;
    wire [7:0] tohost_char = D_OUT[7:0];

    reg [7:0] ubuf [0:4095];
    reg [11:0] r_head=0, r_tail=0;
    always @(posedge w_clk50m) if (core_rst==0 && tohost_cmd==1) begin
        ubuf[r_tail] <= tohost_char;
        r_tail <= r_tail + 1;
    end

    reg [31:0] r_cnt = 1;
    always @(posedge w_clk50m) r_cnt <= (r_cnt>=(`UART_CNT*12)) ? 1 : r_cnt + 1;

    wire w_uart_we = ((core_rst==0) && (r_cnt==1) && (r_head!=r_tail));
    wire w_uart_tx_t;
    always @(posedge w_clk50m) if(core_rst==0 && w_uart_we) r_head <= r_head + 1;
    m_uart_tx uart_tx1 (w_clk50m, w_uart_we, ubuf[r_head], w_uart_tx);

    always@(posedge w_clk50m) begin ///// for simulation
        if (tohost_cmd==1) $write("%c", tohost_char);
        if (tohost_cmd==2) begin
            $write("¥nsimulation finished.¥n");
            $finish();
        end
    end
endmodule
```

proc.v

RVCore_Base, RISC-V RV32I Baseline Version v2024-10-22a

```
module m_uart_tx (
    input  wire       w_clk,     // 50MHz clock signal
    input  wire       w_we,      // write enable
    input  wire [7:0] w_din,     // data in
    output wire       w_uart_tx  // UART tx, data line from FPGA to PC
);
    reg [8:0] r_buf = 9'b1_1111_1111;
    reg [7:0] r_cnt = 1;
    always @(posedge w_clk) begin
        r_cnt <= (w_we) ? 1 : (r_cnt==`UART_CNT) ? 1 : r_cnt + 1;
        r_buf <= (w_we) ? {w_din, 1'b0} : (r_cnt==`UART_CNT) ? {1'b1, r_buf[8:1]} : r_buf;
    end
    assign w_uart_tx = r_buf[0];
endmodule
```

```
module m_uart_rx (
    input  wire       w_clk,   // 100MHz clock signal
    input  wire       w_rxd,   // UART rx, data line from PC to FPGA
    output wire [7:0] w_char,  // 8-bit data received
    output reg        r_en = 0 // data enable
);
    reg [2:0] r_detect_cnt = 0; /* to detect the start bit */
    always @(posedge w_clk) r_detect_cnt <= (w_rxd) ? 0 : r_detect_cnt + 1;
    wire w_detected = (r_detect_cnt>2);

    reg       r_busy = 0; // r_busy is set while receiving 9-bits data
    reg [3:0] r_bit  = 0; // the number of received bits
    reg [7:0] r_cnt  = 0; // wait count for 1Mbaud
    always@(posedge w_clk) r_cnt <= (r_busy==0) ? 1 : (r_cnt==`UART_CNT) ? 1 : r_cnt + 1;

    reg [8:0] r_data = 0;
    always@(posedge w_clk) begin
        if (r_busy==0) begin
            {r_data, r_bit, r_en} <= 0;
            if (w_detected) r_busy <= 1;
        end
        else if (r_cnt>= `UART_CNT) begin
            r_bit <= r_bit + 1;
            r_data <= {w_rxd, r_data[8:1]};
            if (r_bit==8) begin r_en <= 1; r_busy <= 0; end
        end
    end
    assign w_char = r_data[7:0];
endmodule
```

proc.v