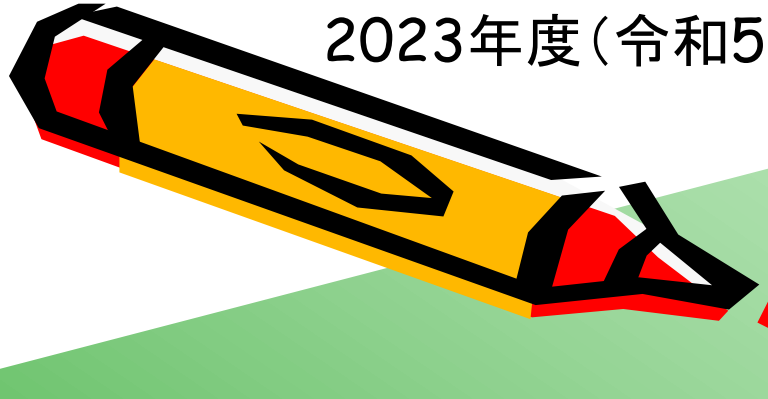


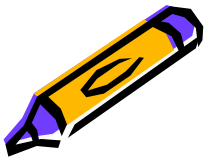
2023年度（令和5年）版

Ver. 2023-10-20a

Course number: CSC.T363

コンピュータアーキテクチャ
Computer Architecture

6. パイプラインプロセッサ
Pipelined Processor

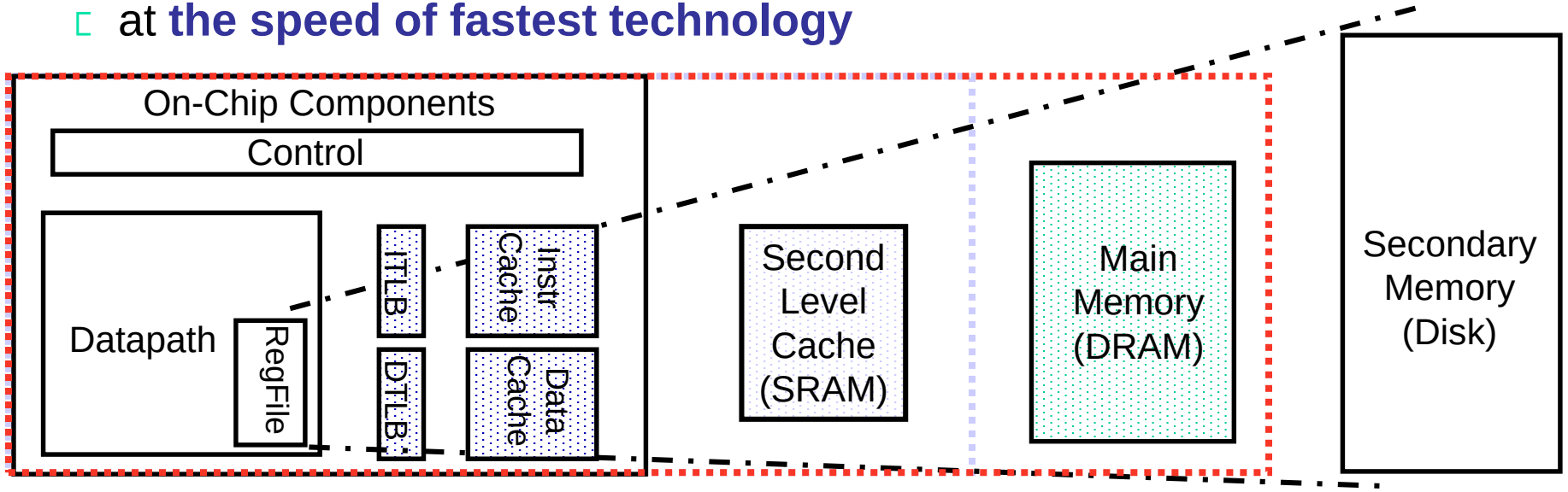


www.arch.cs.titech.ac.jp/lecture/CA/
Tue 13:30-15:10, 15:25-17:05
Fri 13:30-15:10

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

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

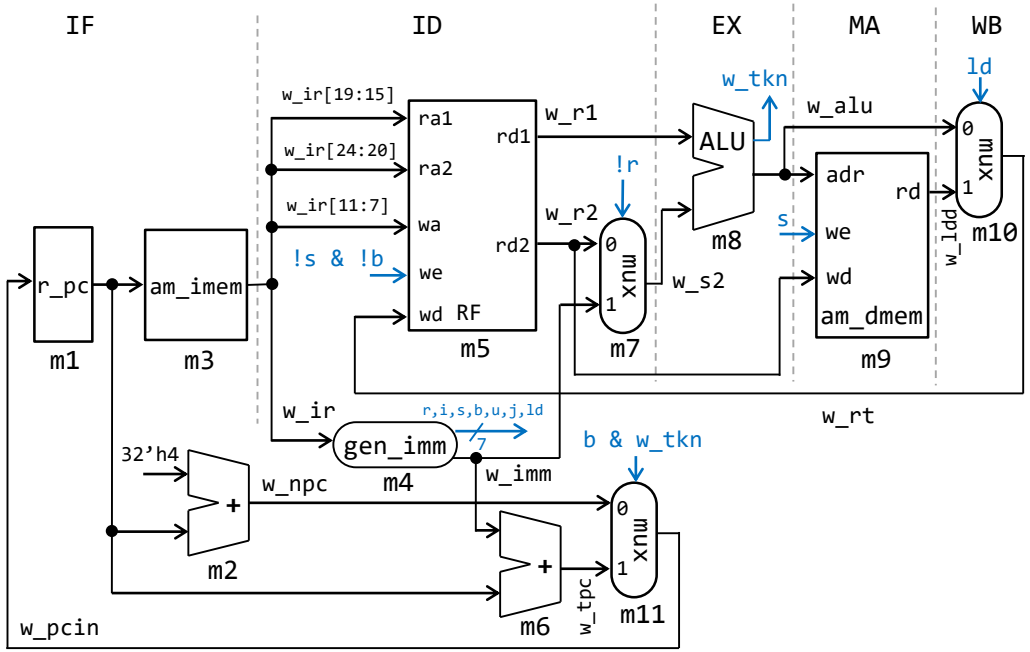
TLB: Translation Lookaside Buffer

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.

proc5: single-cycle RISC-V processor

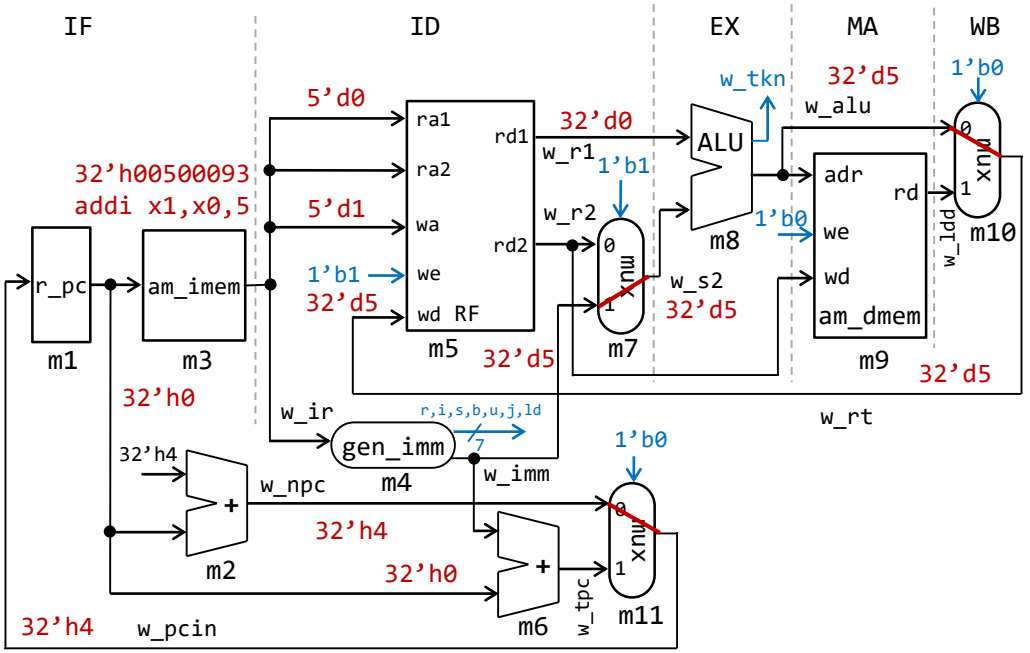
- supporting add, addi, lw, sw, bne instructions



proc5: single-cycle RISC-V processor

```
m.m3.mem[0]={12'd5,5'd0,3'h0,5'd1,7'h13};        // 00   addi x1,x0,5
m.m3.mem[1]={7'd0,5'd1,5'd1,3'h0,5'd2,7'h33};    // 04   add  x2,x1,x1
m.m3.mem[2]={12'd1,5'd1,3'd0,5'd1,7'h13};        // 08 L:addi x1,x1,1
m.m3.mem[3]={~12'd0,5'd2,5'd1,3'h1,5'h1d,7'h63};// 0c   bne  x1,x2,L
m.m3.mem[4]={12'd9,5'd1,3'd0,5'd10,7'h13};       // 10   addi x10,x1,9
```

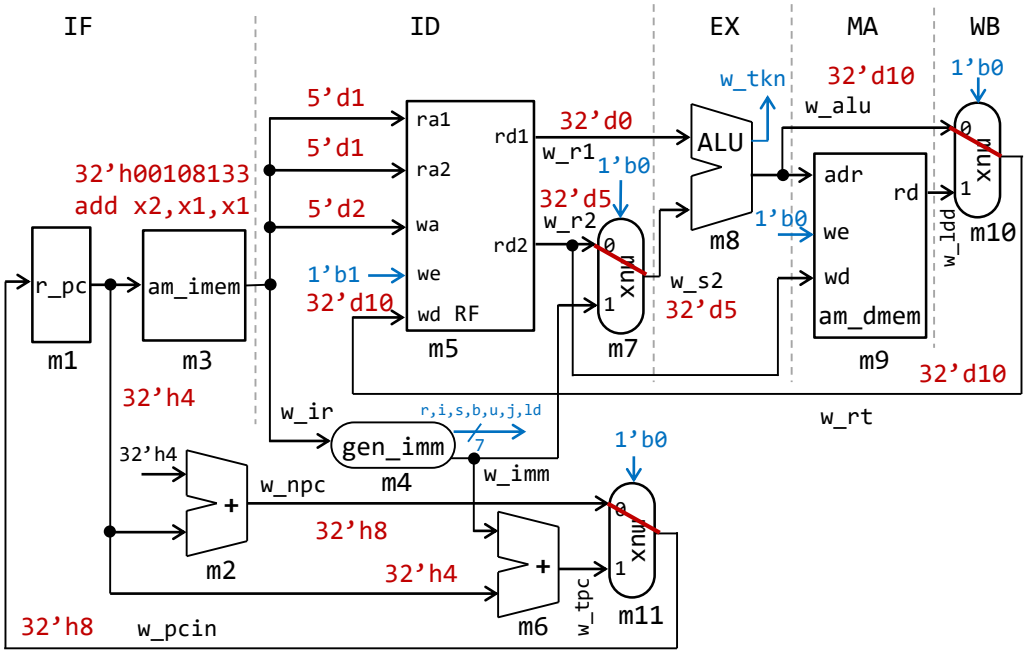
CC1



proc5: single-cycle RISC-V processor

```
m.m3.mem[0]={12'd5,5'd0,3'h0,5'd1,7'h13};        // 00   addi x1,x0,5
m.m3.mem[1]={7'd0,5'd1,5'd1,3'h0,5'd2,7'h33};    // 04   add  x2,x1,x1
m.m3.mem[2]={12'd1,5'd1,3'd0,5'd1,7'h13};        // 08 L:addi x1,x1,1
m.m3.mem[3]={~12'd0,5'd2,5'd1,3'h1,5'h1d,7'h63};// 0c   bne  x1,x2,L
m.m3.mem[4]={12'd9,5'd1,3'd0,5'd10,7'h13};       // 10   addi x10,x1,9
```

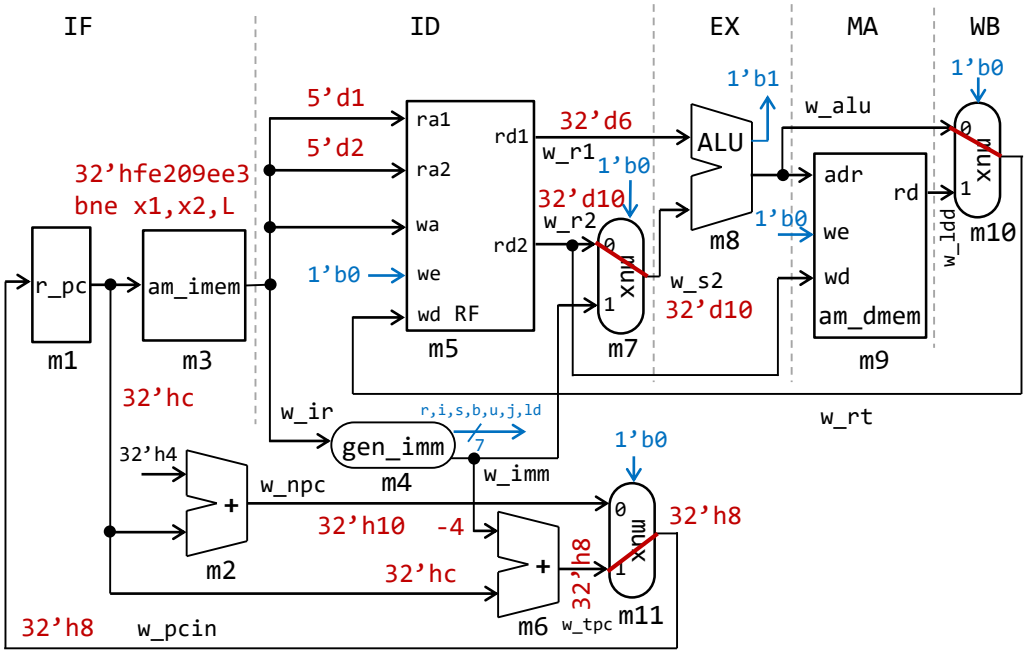
CC2



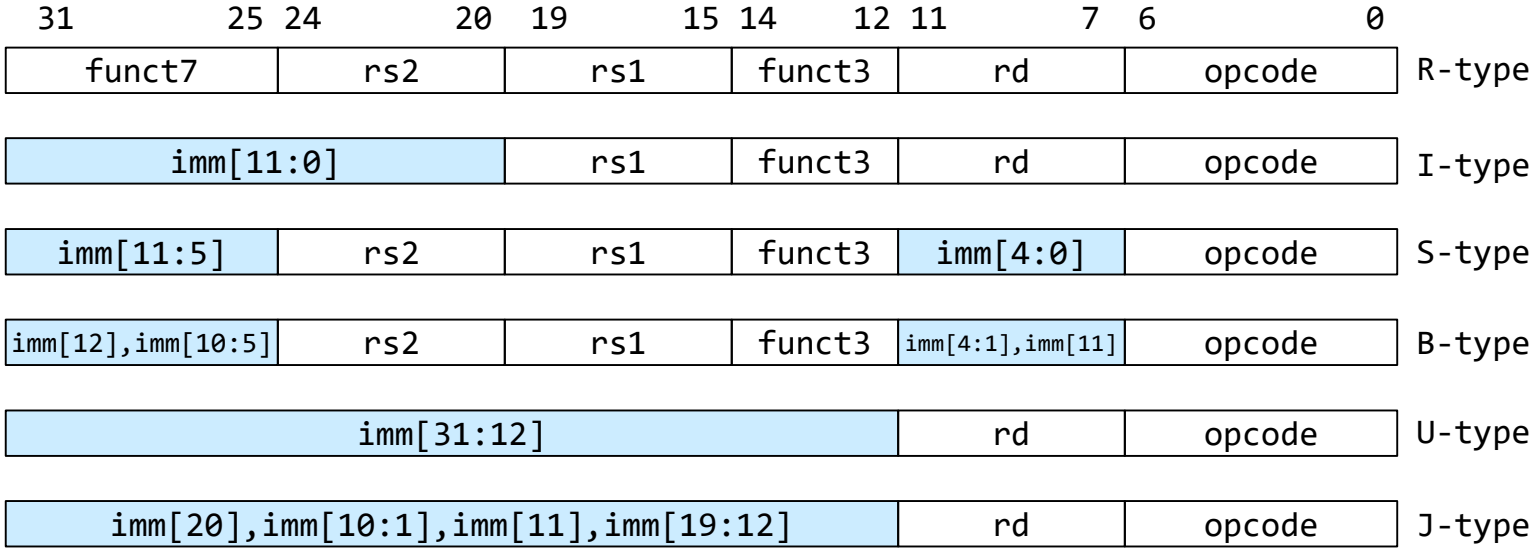
proc5: single-cycle RISC-V processor

```
m.m3.mem[0]={12'd5,5'd0,3'h0,5'd1,7'h13};        // 00   addi x1,x0,5
m.m3.mem[1]={7'd0,5'd1,5'd1,3'h0,5'd2,7'h33};    // 04   add  x2,x1,x1
m.m3.mem[2]={12'd1,5'd1,3'd0,5'd1,7'h13};        // 08 L:addi x1,x1,1
m.m3.mem[3]={~12'd0,5'd2,5'd1,3'h1,5'h1d,7'h63};// 0c   bne  x1,x2,L
m.m3.mem[4]={12'd9,5'd1,3'd0,5'd10,7'h13};       // 10   addi x10,x1,9
```

CC4



RISC-V RV32I instruction format



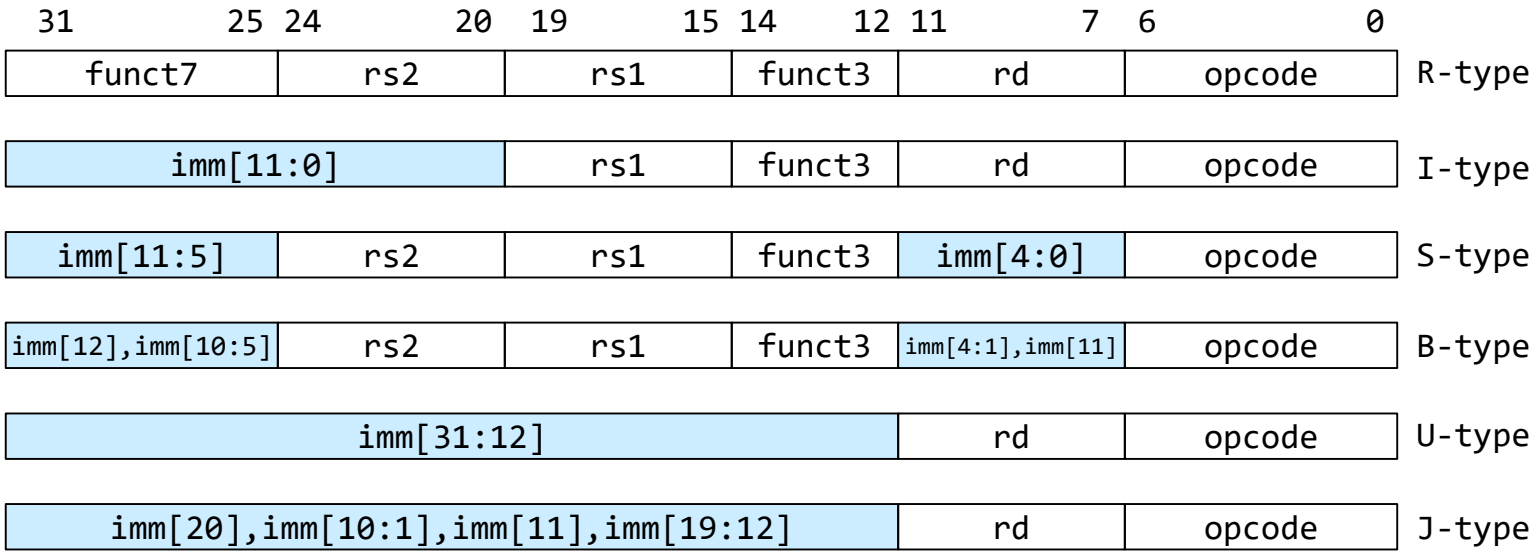
opcode5[2:0] opcode5[4:3]	000	001	010	011	100	101	110
00	LOAD (I-type)	LOAD-FP	custom-0	MISC-MEM (I-type)	OP-IMM (I-type)	AUIPC (U-type)	OP-IMM-32
01	STORE (S-type)	STORE-FP	custom-1	AMO	OP (R-type)	LUI (U-type)	OP-32
10	MADD	MSUB	NMSUB	NMADD	OP-FP	OP-V	custom-2/ rv128
11	BRANCH (B-type)	JALR (I-type)	reserved	JAL (J-type)	SYSTEM (I-type)	reserved	custom-3/ rv128

RISC-V RV32I instruction format

opcode5[2:0] opcode5[4:3]	000	001	010	011	100	101	110
00	LOAD (I-type)	LOAD-FP	custom-0	MISC-MEM (I-type)	OP-IMM (I-type)	AUIPC (U-type)	OP-IMM-32
01	STORE (S-type)	STORE-FP	custom-1	AMO	OP (R-type)	LUI (U-type)	OP-32
10	MADD	MSUB	NMSUB	NMADD	OP-FP	OP-V	custom-2/ rv128
11	BRANCH (B-type)	JALR (I-type)	reserved	JAL (J-type)	SYSTEM (I-type)	reserved	custom-3/ rv128

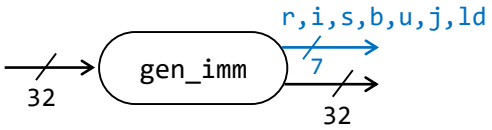
```
module m_get_type(opcode5, r, i, s, b, u, j);
  input  wire [4:0] opcode5;
  output wire r, i, s, b, u, j;
  assign j = (opcode5==5'b11011);
  assign b = (opcode5==5'b11000);
  assign s = (opcode5==5'b01000);
  assign r = (opcode5==5'b01100);
  assign u = (opcode5==5'b01101 || opcode5==5'b00101);
  assign i = ~(j | b | s | r | u);
endmodule
```

RISC-V RV32I instruction format



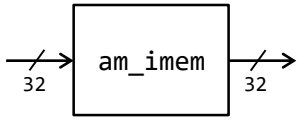

```
module m_get_imm(ir, i, s, b, u, j, imm);
  input wire [31:0] ir;
  input wire i, s, b, u, j;
  output wire [31:0] imm;
  assign imm = (i) ? {{20{ir[31]}}, ir[31:20]} :
               (s) ? {{20{ir[31]}}, ir[31:25], ir[11:7]} :
               (b) ? {{20{ir[31]}}, ir[7], ir[30:25], ir[11:8], 1'b0} :
               (u) ? {ir[31:12], 12'b0} :
               (j) ? {{12{ir[31]}}, ir[19:12], ir[20], ir[30:21], 1'b0} : 0;
endmodule
```

RISC-V RV32I instruction format



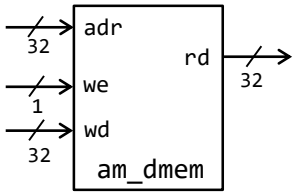

```
module m_gen_imm(w_ir, w_imm, w_r, w_i, w_s, w_b, w_u, w_j, w_ld);
  input  wire [31:0] w_ir;
  output wire [31:0] w_imm;
  output wire w_r, w_i, w_s, w_b, w_u, w_j, w_ld;
  m_get_type m1 (w_ir[6:2], w_r, w_i, w_s, w_b, w_u, w_j);
  m_get_imm m2 (w_ir, w_i, w_s, w_b, w_u, w_j, w_imm);
  assign w_ld = (w_ir[6:2]==5'h0);
endmodule
```

Instruction memory



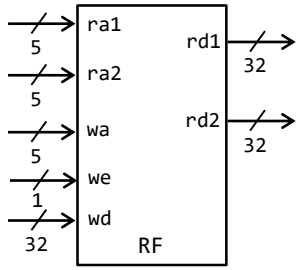

```
module m_am_imem(w_pc, w_insn);
  input  wire [31:0] w_pc;
  output wire [31:0] w_insn;
  reg [31:0] mem [0:63];
  assign w_insn = mem[w_pc[7:2]];
  integer i; initial for (i=0; i<64; i=i+1) mem[i] = 32'd0;
endmodule
```

Data memory



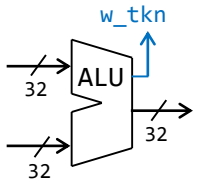

```
module m_am_dmem(w_clk, w_adr, w_we, w_wd, w_rd);
  input  wire w_clk, w_we;
  input  wire [31:0] w_adr, w_wd;
  output wire [31:0] w_rd;
  reg [31:0] mem [0:127];
  assign w_rd = mem[w_adr[7:2]];
  always @(posedge w_clk) if (w_we) mem[w_adr[7:2]] <= w_wd;
  integer i; initial for (i=0; i<64; i=i+1) mem[i] = 32'd0;
endmodule
```

RF (Register File)



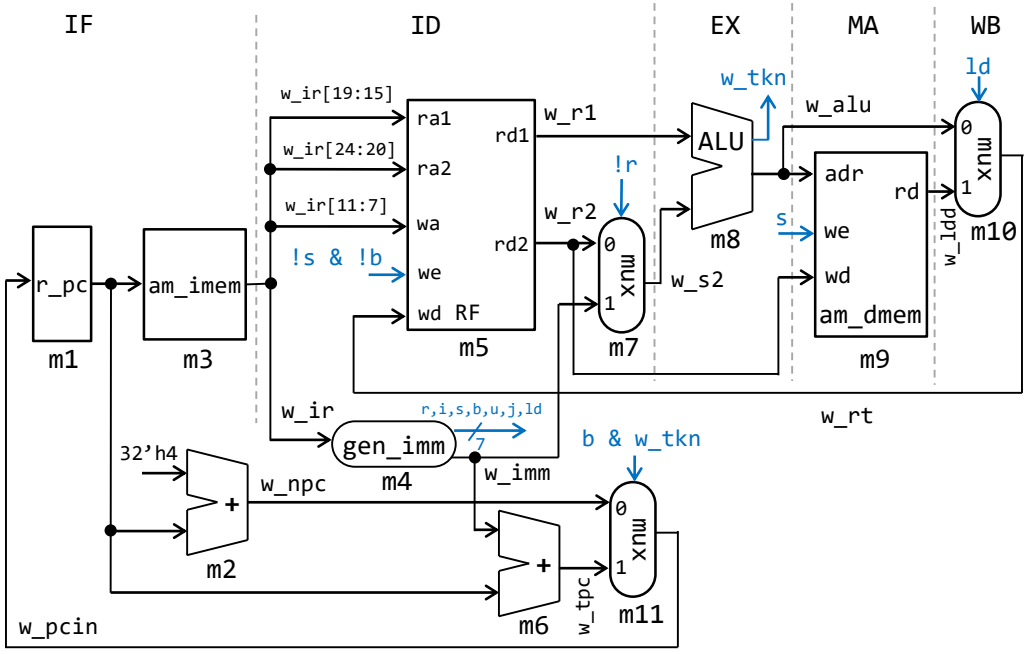

```
module m_RF(w_clk, w_radr1, w_radr2, w_rd1, w_rd2, w_wadr, w_we, w_wd);
  input  wire w_clk, w_we;
  input  wire [ 4:0] w_radr1, w_radr2, w_wadr;
  output wire [31:0] w_rd1, w_rd2;
  input  wire [31:0] w_wd;
  reg [31:0] mem [0:31];
  assign w_rd1 = (w_radr1==5'd0) ? 32'd0 : mem[w_radr1];
  assign w_rd2 = (w_radr2==5'd0) ? 32'd0 : 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
```

ALU (arithmetic and logic unit)



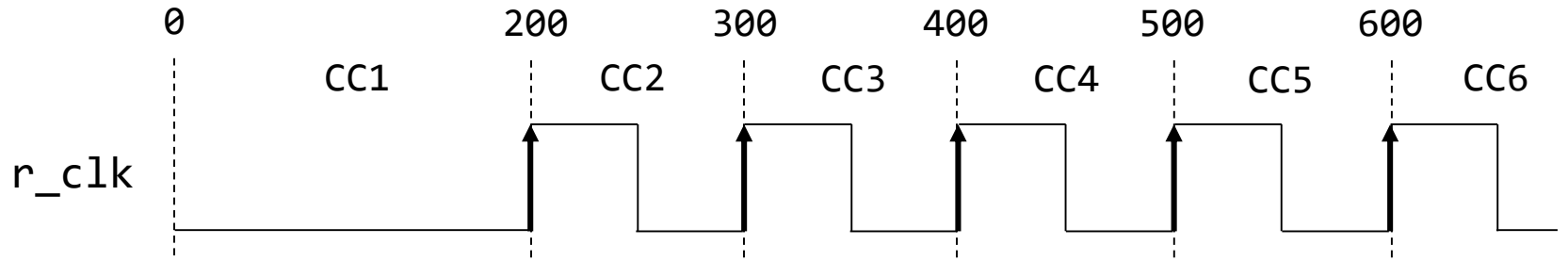

```
module m_alu(w_in1, w_in2, w_out, w_tkn);
  input  wire [31:0] w_in1, w_in2;
  output wire [31:0] w_out;
  output wire w_tkn;
  assign w_out = w_in1 + w_in2;
  assign w_tkn = w_in1 != w_in2;
endmodule
```

proc5s: single-cycle processor




```
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
```

Processor simulation




```
module m_top_wrapper();
  reg r_clk=0; initial #150 forever #50 r_clk = ~r_clk;
  reg [31:0] r_cc=1; always @(posedge r_clk) r_cc <= r_cc + 1;
  initial #1000000 begin $display("time out"); $finish; end
  m_sim m (r_clk, r_cc);
  // initial $dumpvars(0, m);
endmodule
```

```
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_proc5s m (w_clk);
  initial begin
    `define MM m.m3.mem
    `include "asm.txt"
  end
  initial #99 forever #100 $display("CC%02d %h %d %d %d",
    w_cc, m.r_pc, m.w_r1, m.w_s2, m.w_rt);
endmodule
```

Processor simulation

asm.txt

```
`ifdef P1
    `MM[0]={12'd3,5'd0,3'h0,5'd1,7'h13};           // 00   addi x1,x0,3
    `MM[1] ={7'd0,5'd1,5'd1,3'h0,5'd2,7'h33};     // 04   add  x2,x1,x1
    `MM[2]={12'd5,5'd2,3'h0,5'd10,7'h13};         // 08   addi x10,x2,5
    `MM[3]=32'h00050f13;                          // 0c   HALT
`elsif P2
    `MM[0] ={12'd5,5'd0,3'h0,5'd1,7'h13};         // 00   addi x1,x0,5
    `MM[1] ={7'd0,5'd1,5'd1,3'h0,5'd2,7'h33};     // 04   add  x2,x1,x1
    `MM[2] ={12'd1,5'd1,3'h0,5'd1,7'h13};         // 0c L:addi x1,x1,1
    `MM[3] ={~12'd0,5'd2,5'd1,3'h1,5'b11101,7'h63};// 10   bne  x1,x2,L
    `MM[4] ={12'd9,5'd1,3'h0,5'd10,7'h13};        // 14   addi x10,x1,9
    `MM[5] =32'h00050f13;                         // 18   HALT
`else
    `MM[0] ={12'd7,5'd0,3'h0,5'd1,7'h13};         // 00   addi x1,x0,7
    `MM[1] ={7'd0,5'd1,5'd0,3'h2,5'd0,7'h23};     // 04   sw   x1,0(x0)
    `MM[2] ={12'd0,5'd0,3'h2,5'd2,7'h3};          // 08   lw   x2,0(x0)
    `MM[3] ={12'd0,5'd2,3'h0,5'd10,7'h13};        // 0c   addi x10,x2,0
    `MM[4] =32'h00050f13;                         // 10   HALT
`endif
```

```
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_proc5s m (w_clk);
  initial begin
    `define MM m.m3.mem
    `include "asm.txt"
  end
  initial #99 forever #100 $display("CC%02d %h %d %d %d",
    w_cc, m.r_pc, m.w_r1, m.w_s2, m.w_rt);
endmodule
```

Processor simulation

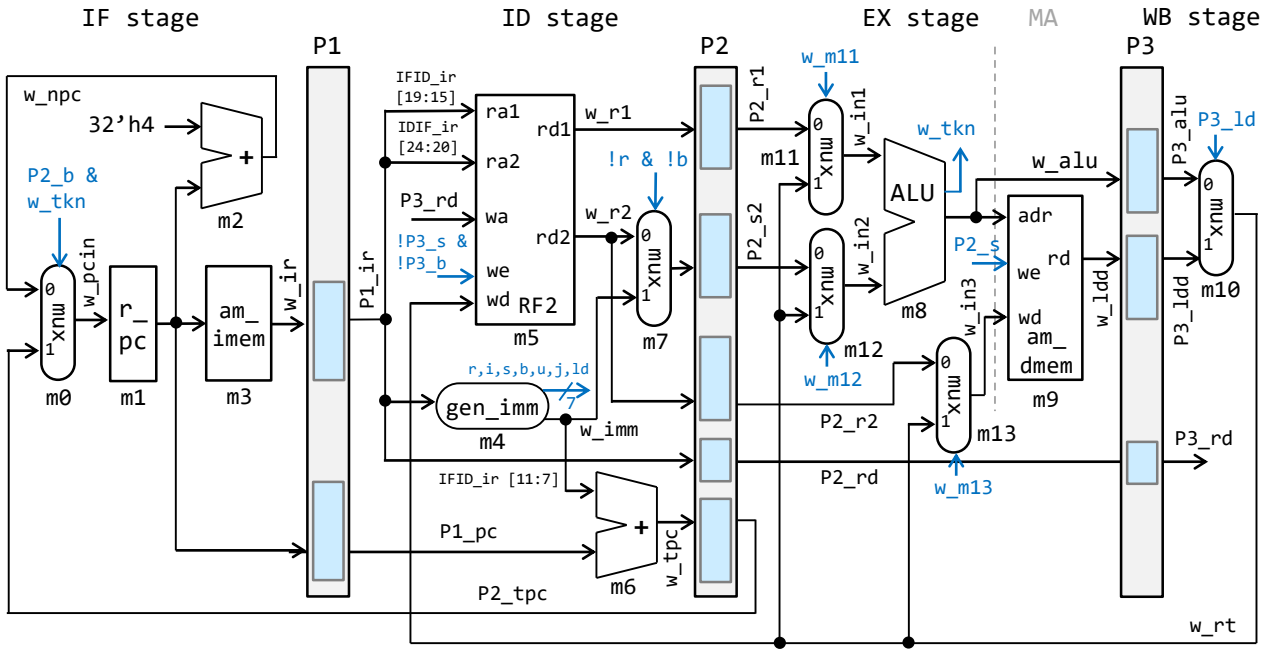
```
`ifdef P1
    `MM[0]={12'd3,5'd0,3'h0,5'd1,7'h13};           // 00   addi x1,x0,3
    `MM[1] ={7'd0,5'd1,5'd1,3'h0,5'd2,7'h33};      // 04   add  x2,x1,x1
    `MM[2]={12'd5,5'd2,3'h0,5'd10,7'h13};          // 08   addi x10,x2,5
    `MM[3]=32'h00050f13;                           // 0c   HALT
`elsif P2
    `MM[0] ={12'd5,5'd0,3'h0,5'd1,7'h13};          // 00   addi x1,x0,5
    `MM[1] ={7'd0,5'd1,5'd1,3'h0,5'd2,7'h33};      // 04   add  x2,x1,x1
    `MM[2] ={12'd1,5'd1,3'h0,5'd1,7'h13};          // 0c L:addi x1,x1,1
    `MM[3] ={~12'd0,5'd2,5'd1,3'h1,5'b11101,7'h63};// 10   bne  x1,x2,L
    `MM[4] ={12'd9,5'd1,3'h0,5'd10,7'h13};         // 14   addi x10,x1,9
    `MM[5] =32'h00050f13;                          // 18   HALT
`else
    `MM[0] ={12'd7,5'd0,3'h0,5'd1,7'h13};          // 00   addi x1,x0,7
    `MM[1] ={7'd0,5'd1,5'd0,3'h2,5'd0,7'h23};      // 04   sw   x1,0(x0)
    `MM[2] ={12'd0,5'd0,3'h2,5'd2,7'h3};           // 08   lw   x2,0(x0)
    `MM[3] ={12'd0,5'd2,3'h0,5'd10,7'h13};         // 0c   addi x10,x2,0
    `MM[4] =32'h00050f13;                          // 10   HALT
`endif
```

```
$ iverilog proc5s.v
$ ./a.out

CC01 00000000          0          7          7
CC02 00000004          0          0          0
CC03 00000008          0          0          7
CC04 0000000c          7          0          7
CC05 00000010          7          0          7

$ iverilog -DP1 proc5s.v
$ ./a.out

CC01 00000000          0          3          3
CC02 00000004          3          3          6
CC03 00000008          6          5         11
CC04 0000000c         11          0         11
```

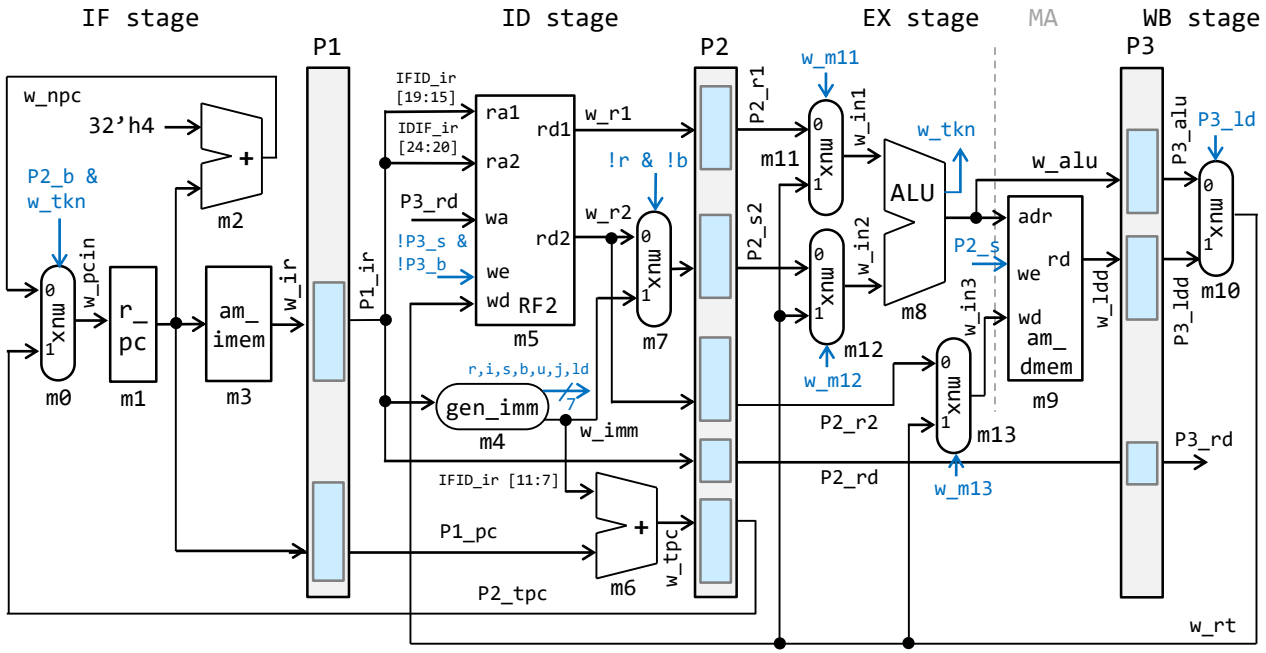
プロセッサが命令を処理するための5つのステップ

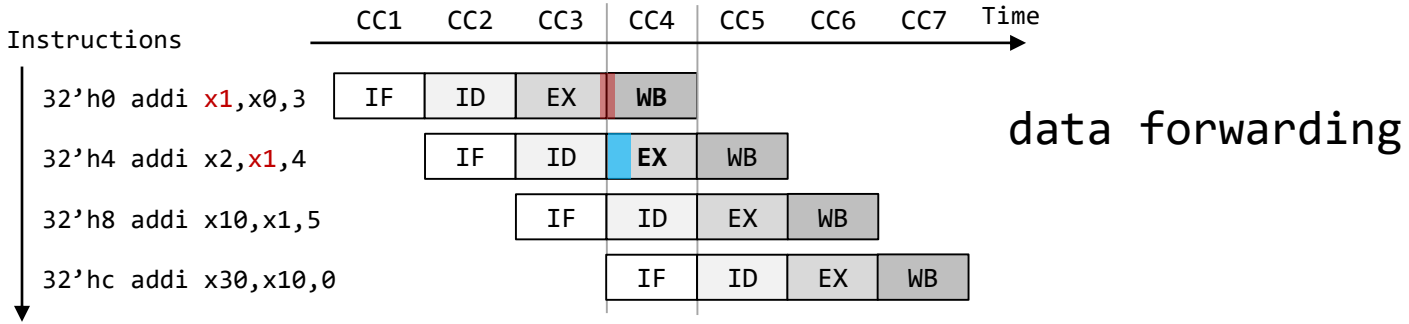
- IF (Instruction Fetch)
  メモリから命令をフェッチする.
- ID (Instruction Decode)
  命令をデコード(解読)しながら, レジスタの値を読み出す.
- EX (Execution)
  命令操作の実行またはアドレスの生成を行う.
- MA (Memory Access)
  必要であれば, データ・メモリ中のオペランドにアクセスする.
- WB (Write Back)
  必要であれば, 結果をレジスタに書き込む.

proc8: 4-stage pipelining processor



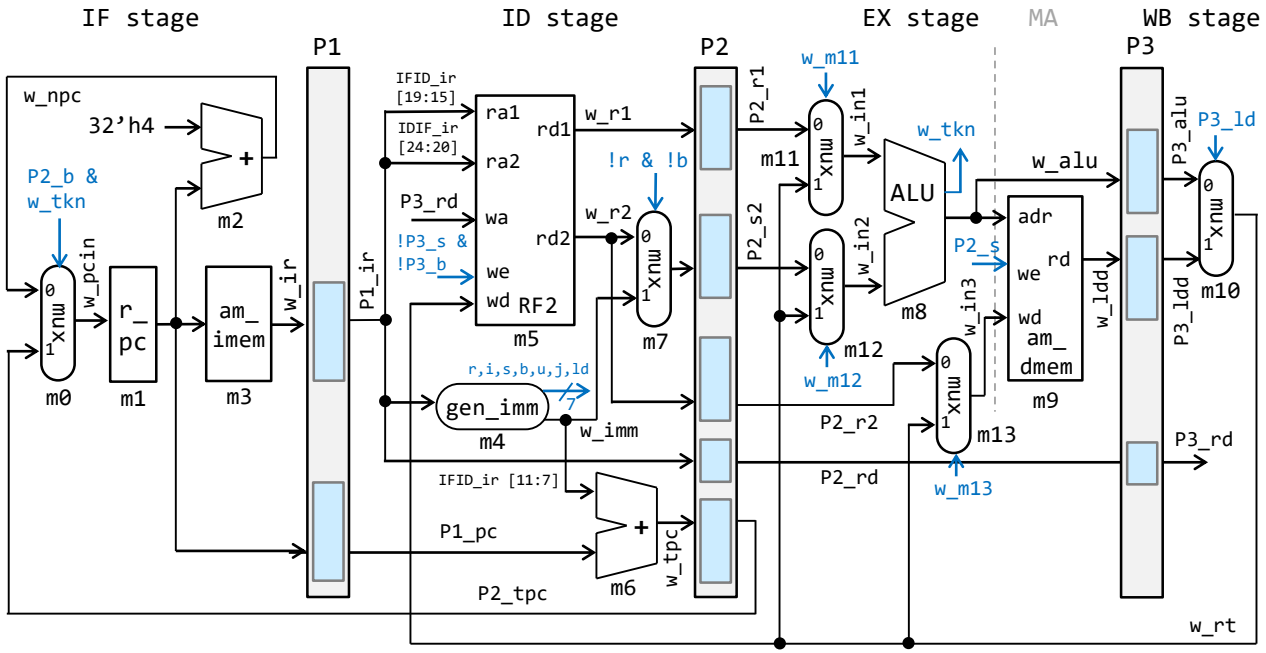
proc8: 4-stage pipelining processor

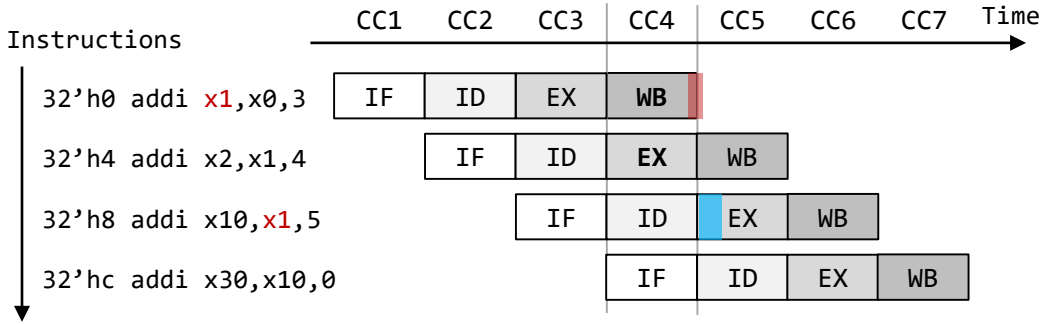




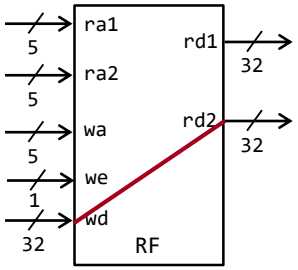
data forwarding

proc8: 4-stage pipelining processor



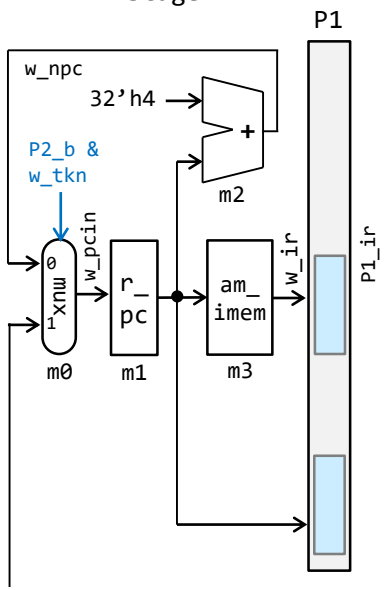


RF2 (Register File) with bypassing

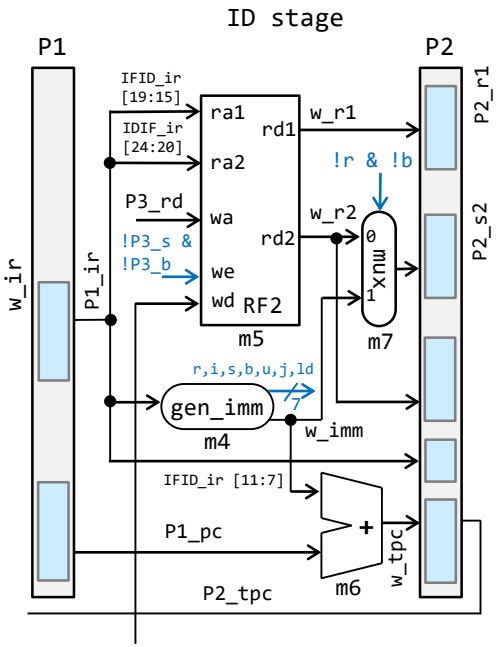



```
module m_RF2(w_clk, w_radr1, w_radr2, w_rd1, w_rd2, w_wadr, w_we, w_wd);
  input  wire w_clk, w_we;
  input  wire [4:0] w_radr1, w_radr2, w_wadr;
  output wire [31:0] w_rd1, w_rd2;
  input  wire [31:0] w_wd;
  reg [31:0] mem [0:31];
  wire w_bp1 = (w_we & w_radr1==w_wadr);
  wire w_bp2 = (w_we & 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
```

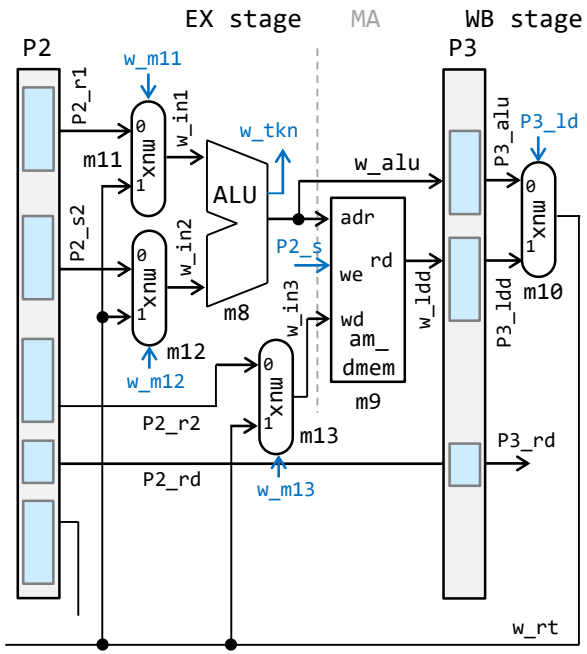
IF stage




```
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
```



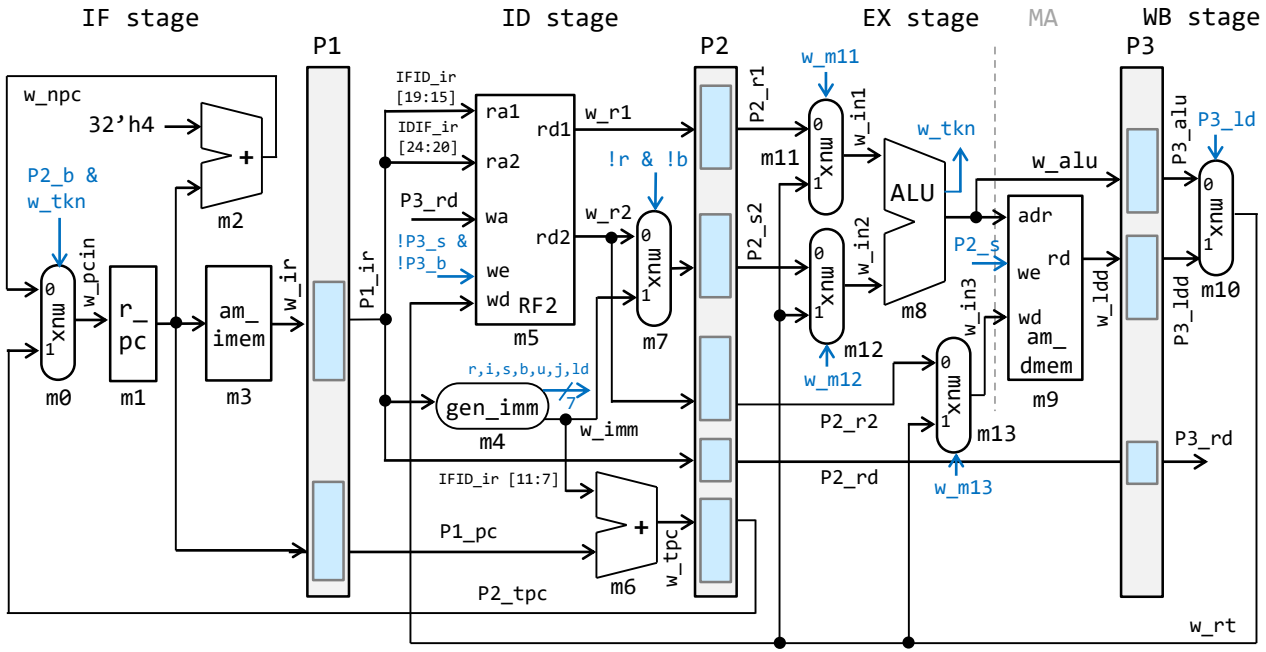

```
module m_proc8s(w_clk);
  input wire w_clk;
  reg [31:0] P1_ir=32'h13, P1_pc=0, P2_pc=0, P3_pc=0;
  reg [31:0] P2_r1=0, P2_s2=0, P2_r2=0, P2_tpc=0;
  reg [31:0] P3_alu=0, P3_ldd=0;
  reg P2_r=0, P2_s=0, P2_b=0, P2_ld=0, P3_s=0, P3_b=0, P3_ld=0;
  reg [4:0] P2_rd=0, P2_rs1=0, P2_rs2=0, P3_rd=0;
  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
```



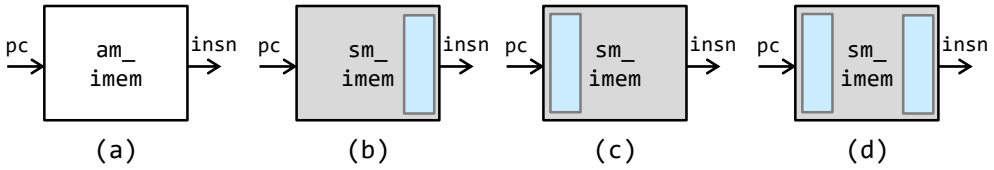

```
module m_proc8s(w_clk);
  input wire w_clk;
  reg [31:0] P1_ir=32'h13, P1_pc=0, P2_pc=0, P3_pc=0;
  reg [31:0] P2_r1=0, P2_s2=0, P2_r2=0, P2_tpc=0;
  reg [31:0] P3_alu=0, P3_ldd=0;
  reg P2_r=0, P2_s=0, P2_b=0, P2_ld=0, P3_s=0, P3_b=0, P3_ld=0;
  reg [4:0] P2_rd=0, P2_rs1=0, P2_rs2=0, P3_rd=0;
  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
```

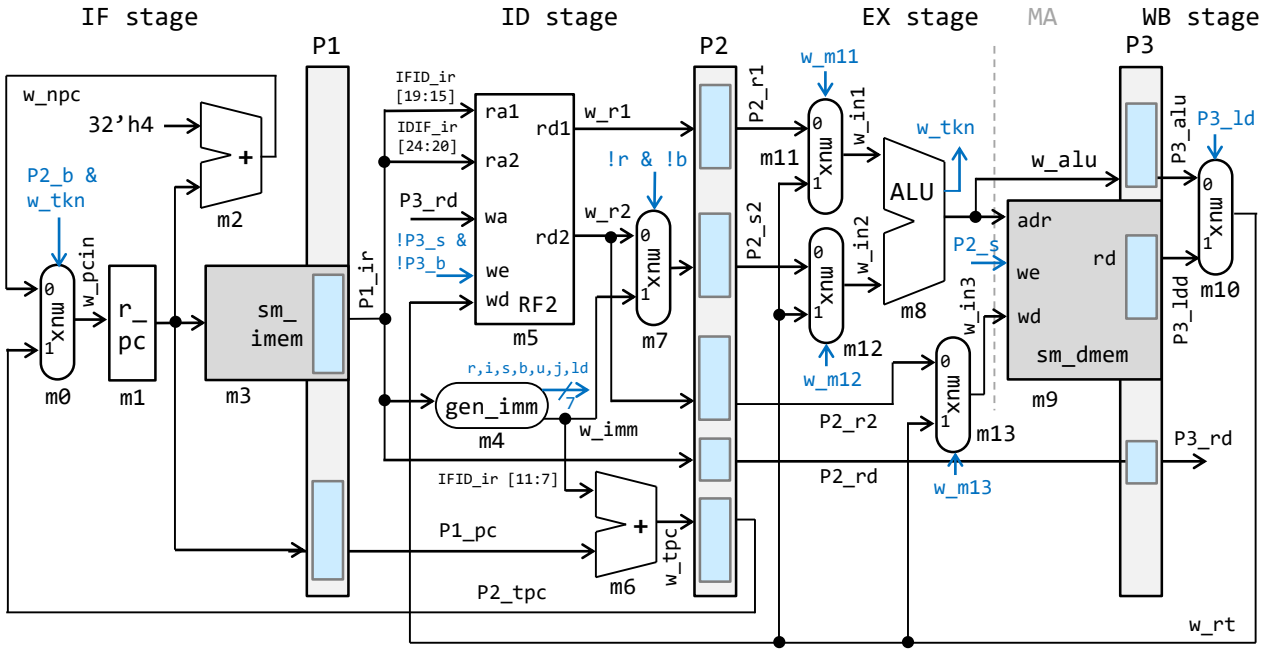
```
module m_proc8s(w_clk);
  input wire w_clk;
  reg [31:0] P1_ir=32'h13, P1_pc=0, P2_pc=0, P3_pc=0;
  reg [31:0] P2_r1=0, P2_s2=0, P2_r2=0, P2_tpc=0;
  reg [31:0] P3_alu=0, P3_ldd=0;
  reg P2_r=0, P2_s=0, P2_b=0, P2_ld=0, P3_s=0, P3_b=0, P3_ld=0;
  reg [4:0] P2_rd=0, P2_rs1=0, P2_rs2=0, P3_rd=0;
  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
```

Cache1

```
module m_cache1(w_clk, w_adr, w_hit, w_dout, w_wadr, w_we, w_wd);
  input  wire w_clk, w_we;
  input  wire [31:0] w_adr;
  input  wire [4:0] w_wadr;
  input  wire [57:0] w_wd;
  output wire w_hit;
  output wire [31:0] w_dout;
  wire w_v;
  wire [24:0] w_tag;
  reg [57:0] mem [0:31];
  always @(posedge w_clk) if (w_we) mem[w_wadr] <= w_wd;
  assign {w_v, w_tag, w_dout} = mem[w_adr[6:2]];
  assign w_hit = w_v & (w_tag==w_adr[31:7]);
  integer i; initial for (i=0; i<32; i=i+1) mem[i] = 0;
endmodule
```