Course number: CSC.T341

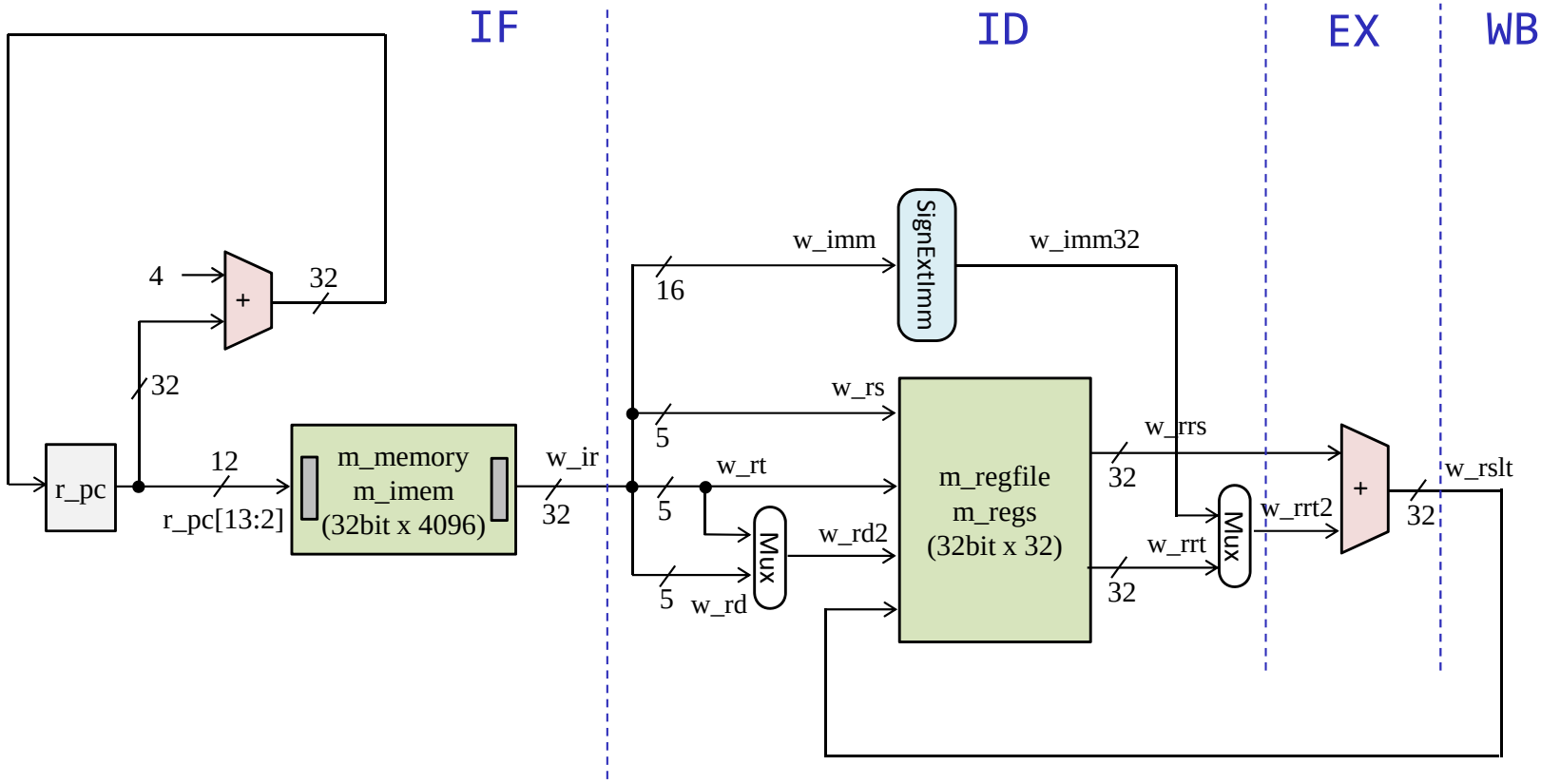
コンピュータ論理設計
Computer Logic Design

10. シングルサイクルプロセッサのデータパス
Datapath for Single 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

Inside module m_proc03

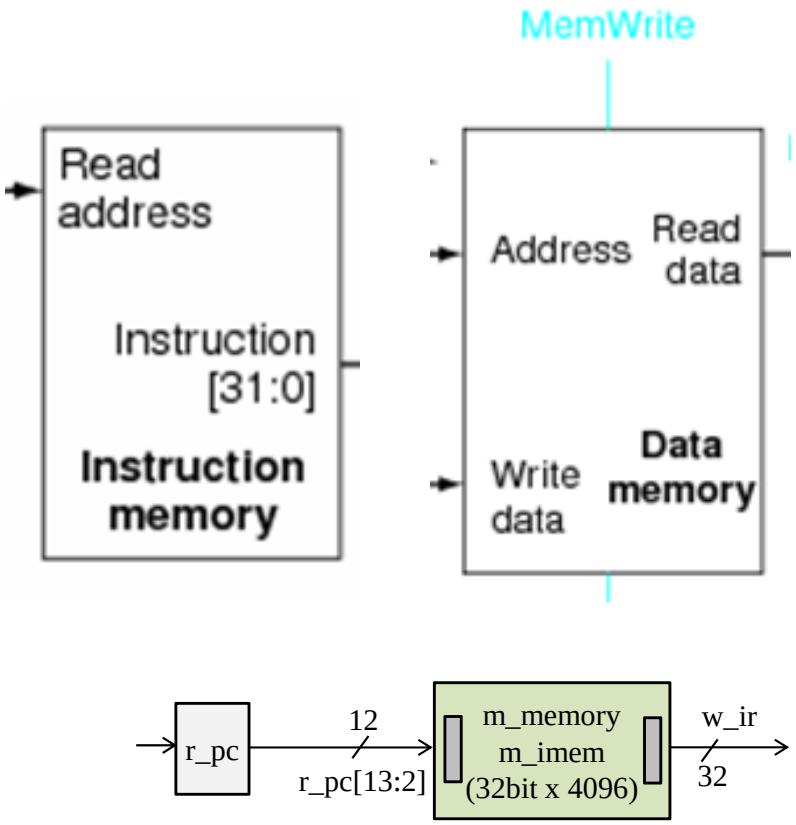
- 命令フェッチ(IF), デコードとオペランドフェッチ(ID), 実行(EX), ライトバック(WB) の処理をおこなう加算命令(add, addi)に対応したプロセッサ



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

Computer Memory

- Read-only memory (ROM)
- Random-access memory (RAM)
  - Verilog HDLでは，ビット幅Bでワード数Wのメモリcm_ramを reg [B-1:0] cm_ram [0:W-1] として宣言できる.



code083.v

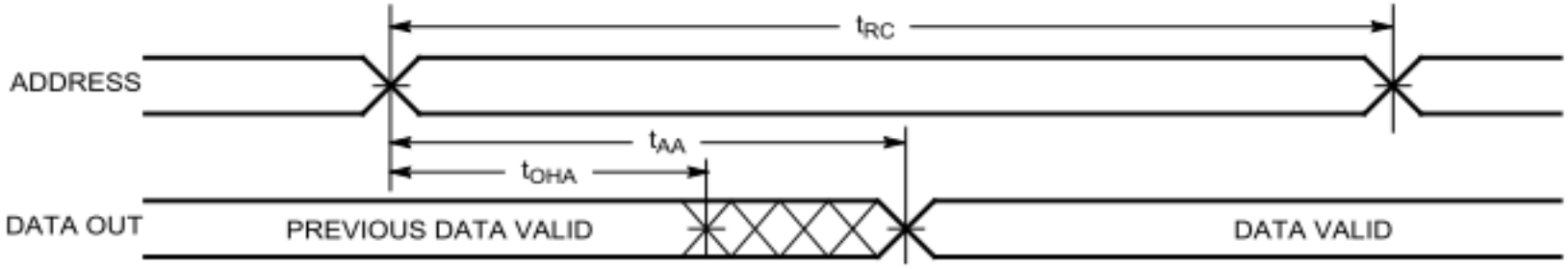
```
module m_memory (w_clk, w_addr, w_we, w_din, r_dout);
  input  wire w_clk, w_we;
  input  wire [11:0] w_addr;
  input  wire [31:0] w_din;
  output reg  [31:0] r_dout;
  reg r_we=0;
  reg [11:0] r_addr=0;
  reg [31:0] r_din=0;
  reg [31:0] cm_ram [0:4095]; // 4K word (4096 x 32bit) memory
  always @(posedge w_clk) begin
    r_addr <= w_addr;
    r_din  <= w_din;
    r_we   <= w_we;
    r_dout <= cm_ram[r_addr];
    if (r_we) cm_ram[r_addr] <= r_din;
  end
  initial begin
    r_dout = 0;
    cm_ram[0] = {6'h0, 5'd0, 5'd0, 5'd0, 5'h0, 6'h20}; // add $0, $0, $0
    cm_ram[1] = {6'h0, 5'd2, 5'd1, 5'd2, 5'h0, 6'h20}; // add $2, $2, $1
    cm_ram[2] = {6'h0, 5'd3, 5'd1, 5'd3, 5'h0, 6'h20}; // add $3, $3, $1
    cm_ram[3] = {6'h0, 5'd4, 5'd4, 5'd4, 5'h0, 6'h20}; // add $4, $4, $4
  end
endmodule
```

非同期式メモリの例

CY7C1049DV33

Switching Waveforms

Figure 3. Read Cycle No. 1[13, 14]



Inside module m_amemory

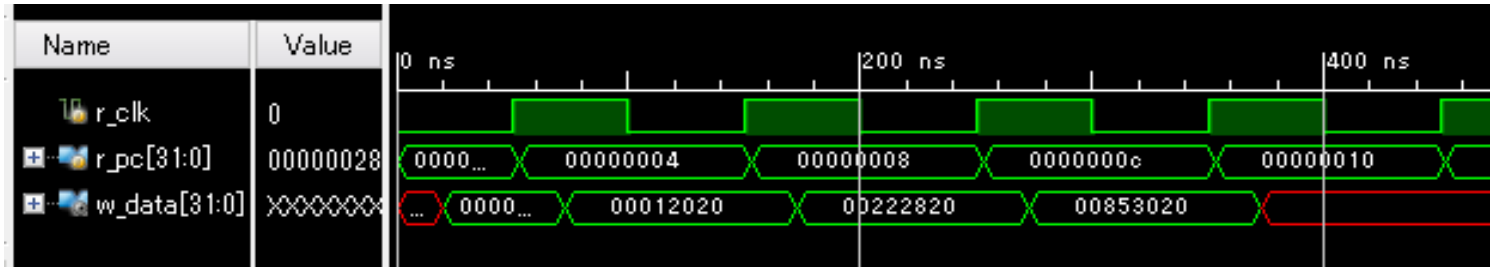
- 非同期式メモリの記述の例を示す．メモリの読み出しの遅延を 20nsec とした．
- main.vを code112.v と code111.v の内容となるように入力して，シミュレーションする．

code112.v

```
`default_nettype none

module m_top ();
  reg r_clk=0; initial forever #50 r_clk = ~r_clk;
  reg [31:0] r_pc = 0;
  always @(posedge r_clk) r_pc <= #3 r_pc + 4;

  wire [31:0] w_data;
  m_amemory m (r_clk, r_pc[13:2], 0, 0, w_data);

  initial begin
    m.cm_ram[0] = {6'h0, 5'd0, 5'd0, 5'd0, 5'h0, 6'h20}; // add $0, $0, $0
    m.cm_ram[1] = {6'h0, 5'd0, 5'd1, 5'd4, 5'h0, 6'h20}; // add $4, $0, $1
    m.cm_ram[2] = {6'h0, 5'd1, 5'd2, 5'd5, 5'h0, 6'h20}; // add $5, $1, $2
    m.cm_ram[3] = {6'h0, 5'd4, 5'd5, 5'd6, 5'h0, 6'h20}; // add $6, $4, $5
  end

  always@(*) #80 $write("%3d %d %x¥n", $time, r_pc, w_data);
endmodule
```

code111.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];
endmodule
```



プロセッサが命令を処理するための基本的な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.
- MEM (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.

MIPS Memory Access Instructions

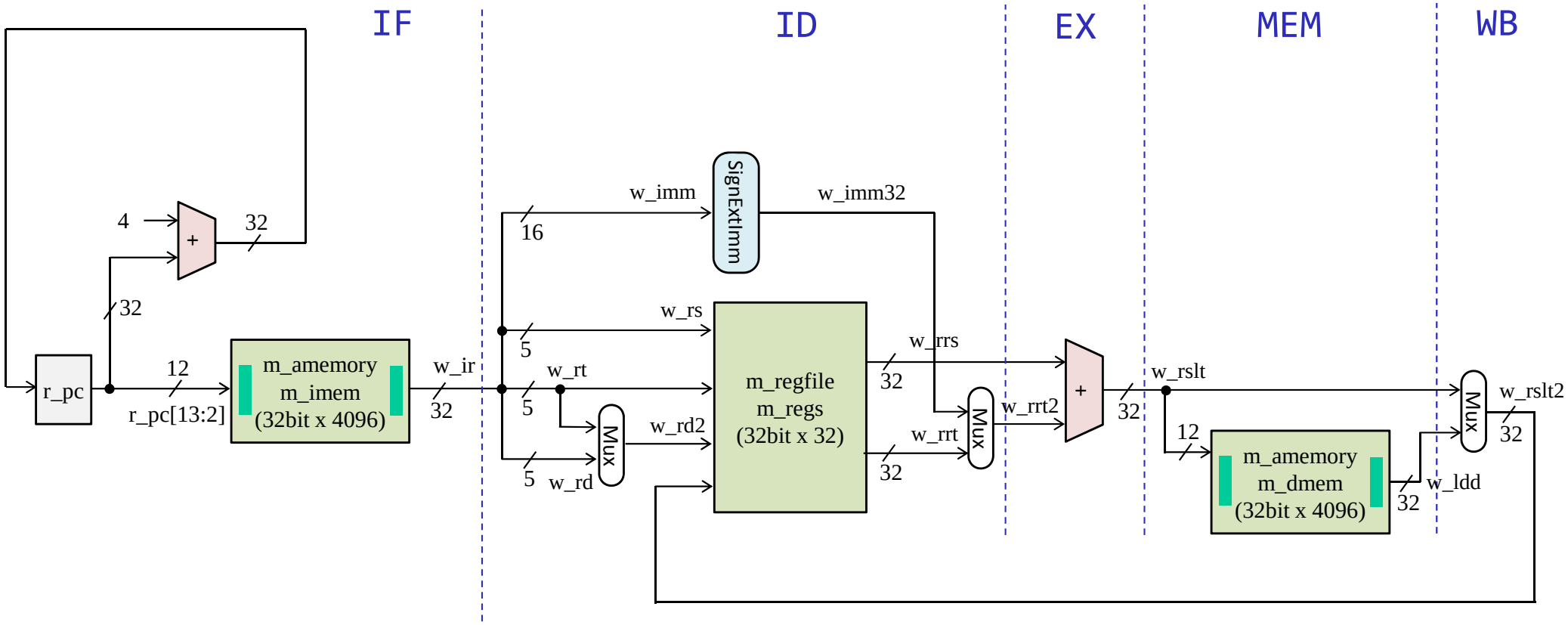
- MIPS has two basic data transfer instructions for accessing memory

```
lw  $t0, 4($s3)  # load word from memory
sw  $t0, 8($s3)  # store word to memory
```

Add	add	R	R[rd]=R[rs]+R[rt]
Add Immediate	addi	I	R[rt]=R[rs]+SignExtImm
Add Imm. Unsigned	addiu	I	R[rt]=R[rs]+SignExtImm
Add Unsigned	addu	R	R[rd]=R[rs]+R[rt]
Subtract	sub	R	R[rd]=R[rs]-R[rt]
Subtract Unsigned	subu	R	R[rd]=R[rs]-R[rt]
Load Word	lw	I	R[rt]=M[R[rs]+SignExtImm]
Load Immediate	li	P	R[rd]=immediate
Load Address	la	P	R[rd]=immediate
Store Byte	sb	I	M[R[rs]+SignExtImm] (7:0)=R[rt](7:0)
Store Halfword	sh	I	M[R[rs]+SignExtImm] (15:0)=R[rt](15:0)
Store Word	sw	I	M[R[rs]+SignExtImm]=R[rt]

Inside module m_proc04

- 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命令に対応したプロセッサ



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

Inside module m_proc04

- 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命令に対応したプロセッサ
- main.vを code{093, 111, 114, 113}.v となるように編集して，シミュレーションする.
  - 加算，レジスタの更新，メモリ読み出しにシミュレーションのための遅延を挿入

code114.v

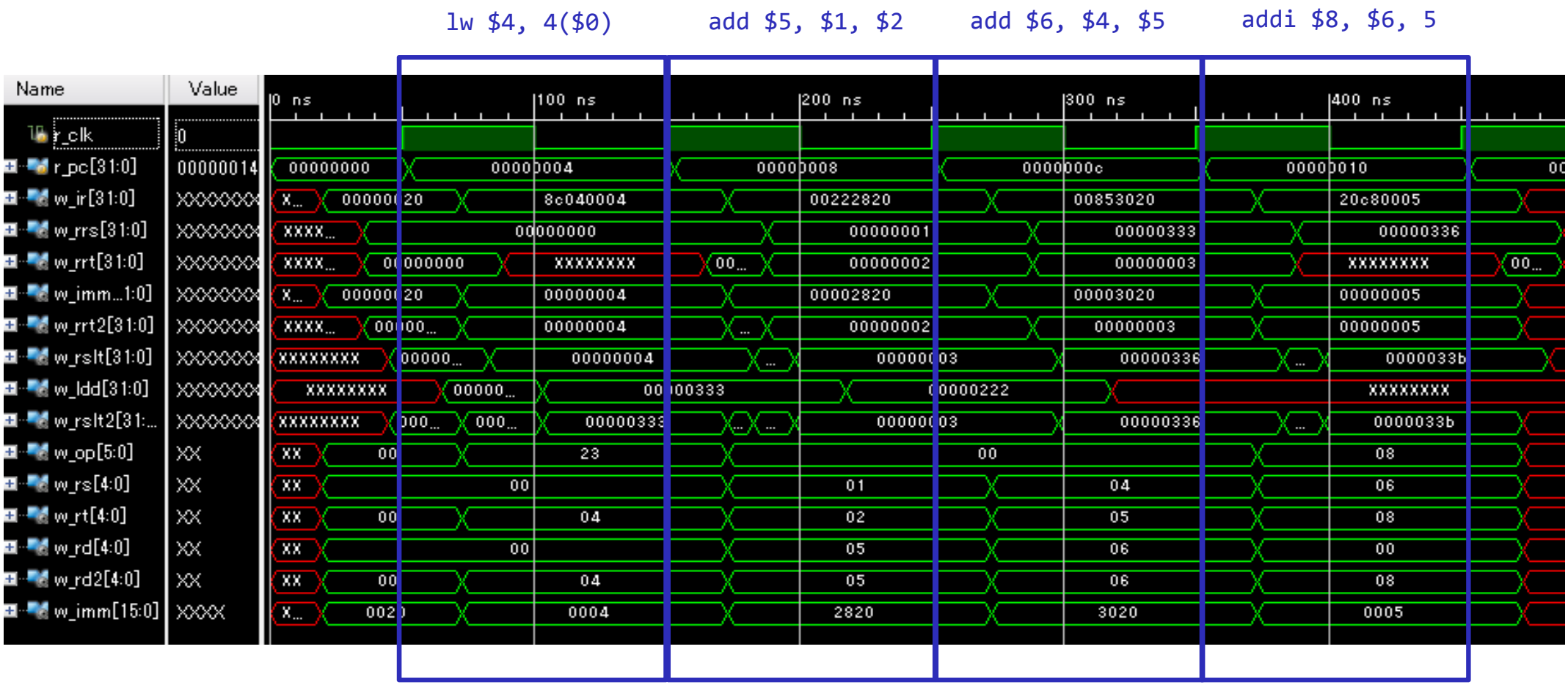
```
module m_top ();
  reg r_clk=0; initial forever #50 r_clk = ~r_clk;
  wire [15:0] w_led;
  m_proc04 p (r_clk, 0, 1, w_led);
 always@(*) #80 $write("%4d %x %x %x %x %x¥n", $time,
                        p.r_pc, p.w_ir, p.w_rrs, p.w_rrt2, p.w_rslt2);
  initial begin
    p.m_imem.cm_ram[0] = {6'h0, 5'd0, 5'd0, 5'd0, 5'h0, 6'h20}; // add  $0, $0, $0
    p.m_imem.cm_ram[1] = {6'h23,5'd0, 5'd4, 16'h4};             // lw   $4, 4($0)
    p.m_imem.cm_ram[2] = {6'h0, 5'd1, 5'd2, 5'd5, 5'h0, 6'h20}; // add  $5, $1, $2
    p.m_imem.cm_ram[3] = {6'h0, 5'd4, 5'd5, 5'd6, 5'h0, 6'h20}; // add  $6, $4, $5
    p.m_imem.cm_ram[4] = {6'h8, 5'd6, 5'd8, 16'h5};             // addi $8, $6, 5
    p.m_dmem.cm_ram[0] = 32'h222;
    p.m_dmem.cm_ram[1] = 32'h333;
  end
  initial #550 $finish;
endmodule
```

code113.v

```
module m_proc04 (w_clk, w_btnu, w_btnd, w_led);
  input  wire w_clk, w_btnu, w_btnd;
  output wire [15:0] w_led;

  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;

  always @(posedge w_clk) r_pc <= #3 r_pc + 4;
  m_amemory m_imem (w_clk, r_pc[13:2], 0, 0, w_ir);

  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  [4:0] w_rd2 = (w_op!=0) ? w_rt : w_rd;
  wire [15:0] w_imm = w_ir[15:0];
  m_regfile m_regs (w_clk, w_rs, w_rt, w_rd2, 1, 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;

  m_amemory m_dmem (w_clk, w_rslt[13:2], 0, 0, w_ldd);
  assign w_rslt2 = (w_op>6'h19 && w_op<6'h28) ? w_ldd : w_rslt;

  assign w_led = (w_btnu | w_btnd) ? w_rslt[31:16] : w_rslt[15:0];
endmodule
```

Inside module m_proc04

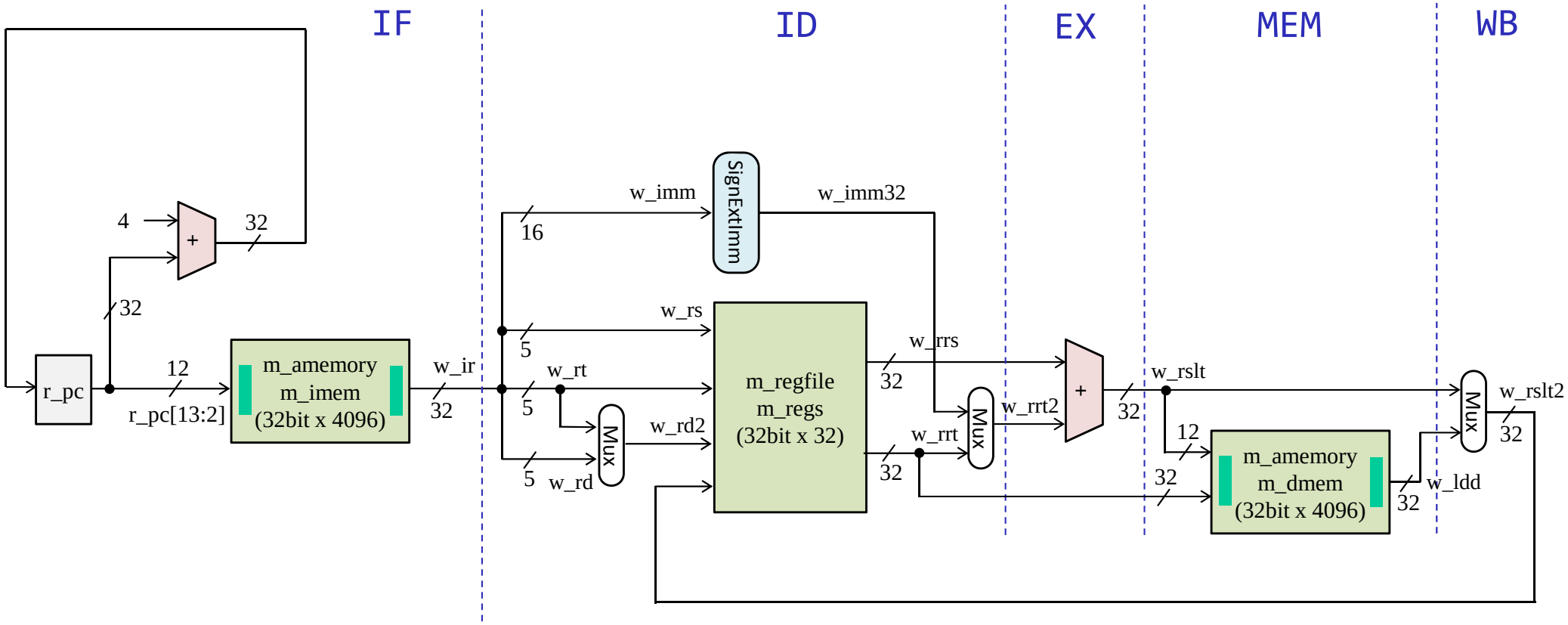
- code114.v と code113.v のシミュレーション結果の波形



Vivado Demo

Inside module m_proc05

- 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命令に対応したプロセッサ



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

Inside module m_proc05

- 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命令に対応したプロセッサ
- main.vを code{093, 111, 116, 115}.v となるように編集して，シミュレーションする.

code116.v

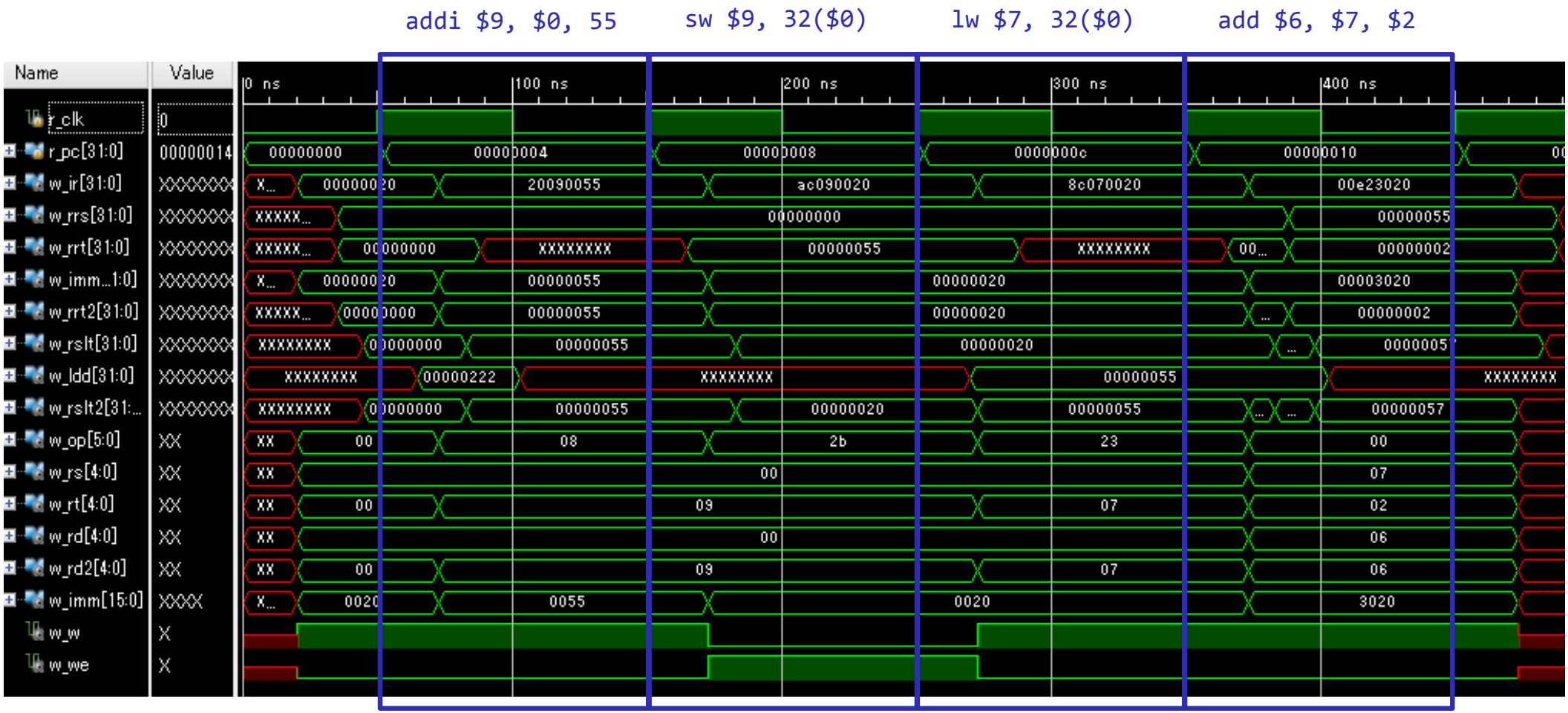
```
module m_top ();
  reg r_clk=0; initial forever #50 r_clk = ~r_clk;
  wire [15:0] w_led;
  m_proc05 p (r_clk, 0, 1, w_led);
 always@(*) #80 $write("%4d %x %x %x %x %x¥n", $time,
                        p.r_pc, p.w_ir, p.w_rrs, p.w_rrt2, p.w_rslt2);
  initial begin
    p.m_imem.cm_ram[0] = {6'h0, 5'd0, 5'd0, 5'd0, 5'h0, 6'h20}; // add  $0, $0, $0
    p.m_imem.cm_ram[1] = {6'h8, 5'd0, 5'd9, 16'h55};            // addi $9, $0, 55
    p.m_imem.cm_ram[2] = {6'h2b,5'd0, 5'd9, 16'd32};            // sw   $9, 32($0)
    p.m_imem.cm_ram[3] = {6'h23,5'd0, 5'd7, 16'd32};            // lw   $7, 32($0)
    p.m_imem.cm_ram[4] = {6'h0, 5'd7, 5'd2, 5'd6, 5'h0, 6'h20}; // add  $6, $7, $2
    p.m_dmem.cm_ram[0] = 32'h222;
    p.m_dmem.cm_ram[1] = 32'h333;
  end
  initial #550 $finish;
endmodule
```

code115.v

```
module m_proc05 (w_clk, w_btnu, w_btnd, w_led);
  input  wire w_clk, w_btnu, w_btnd;
  output wire [15:0] w_led;

  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;

  always @(posedge w_clk) r_pc <= #3 r_pc + 4;
  m_amemory m_imem (w_clk, r_pc[13:2], 0, 0, w_ir);

  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  [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;

  assign w_led = (w_btnu | w_btnd) ? w_rslt[31:16] : w_rslt[15:0];
endmodule
```

Inside module m_proc05

- code116.v と code115.v のシミュレーション結果の波形



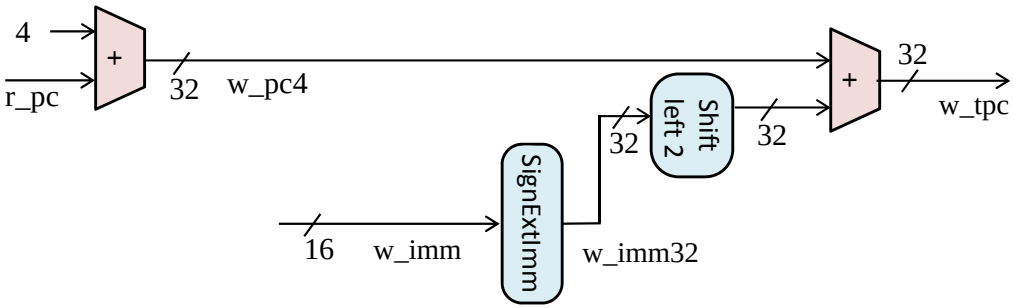
MIPS Control Flow Instructions

- MIPS conditional branch instructions:

```
bne $s0, $s1, Lbl   # go to Lbl if $s0≠$s1
beq $s0, $s1, Lbl   # go to Lbl if $s0=$s1
```

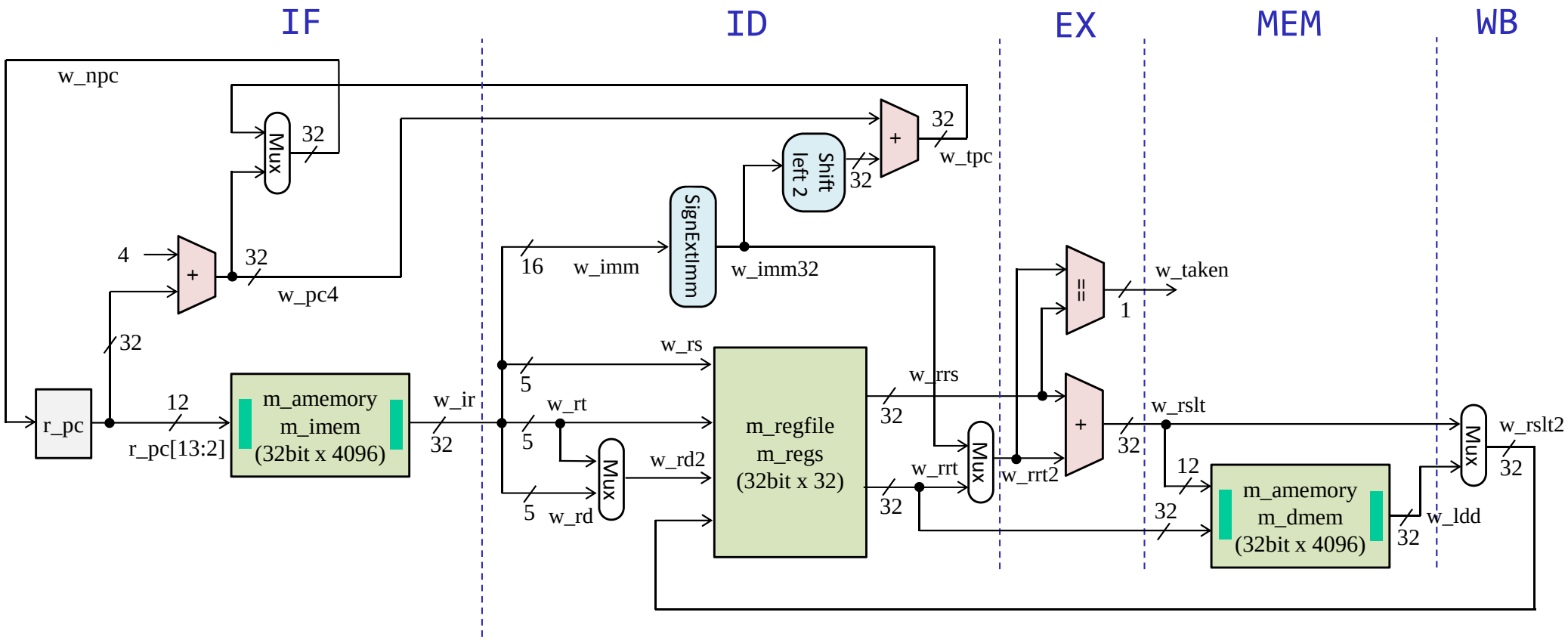
Branch On Equal	beq	I	if(R[rs]==R[rt]) PC=PC+4+BranchAddr
Branch On Not Equal	bne	I	if(R[rs]!=R[rt]) PC=PC+4+BranchAddr

(4) BranchAddr = {14{immediate[15]},immediate,2'b0 }



Inside module m_proc06

- 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



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

Inside module m_proc06

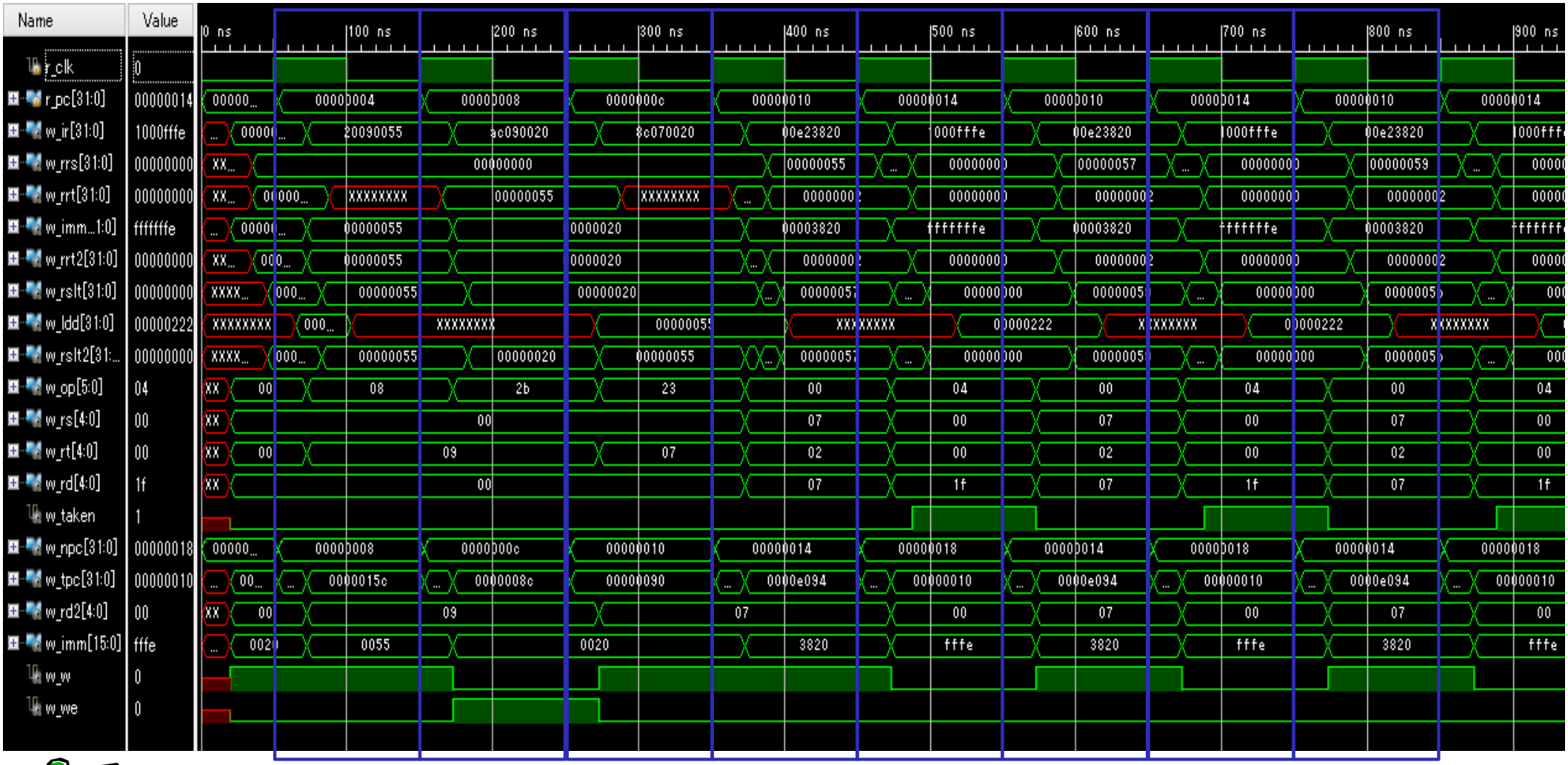
- main.vを code{093, 111, 118, 117}.v となるように編集して，シミュレーションする.

code118.v

```
module m_top ();
  reg r_clk=0; initial forever #50 r_clk = ~r_clk;
  wire [15:0] w_led;
  m_proc06 p (r_clk, 0, 1, w_led);
 always@(*) #80 $write("%4d %x %x %x %x %x¥n", $time,
                        p.r_pc, p.w_ir, p.w_rrs, p.w_rrt2, p.w_rslt2);
  initial begin
    p.m_imem.cm_ram[0] = {6'h0, 5'd0, 5'd0, 5'd0, 5'h0, 6'h20}; //     add  $0, $0, $0
    p.m_imem.cm_ram[1] = {6'h8, 5'd0, 5'd9, 16'h55};            //     addi $9, $0, 55
    p.m_imem.cm_ram[2] = {6'h2b,5'd0, 5'd9, 16'd32};            //     sw   $9, 32($0)
    p.m_imem.cm_ram[3] = {6'h23,5'd0, 5'd7, 16'd32};            //     lw   $7, 32($0)
    p.m_imem.cm_ram[4] = {6'h0, 5'd7, 5'd2, 5'd7, 5'h0, 6'h20}; // L1: add  $7, $7, $2
    p.m_imem.cm_ram[5] = {6'h4, 5'd0, 5'd0, 16'hfffe};          //     beq  $0, $0, L1
    p.m_dmem.cm_ram[0] = 32'h222;
    p.m_dmem.cm_ram[1] = 32'h333;
  end
  initial #1000 $finish;
endmodule
```

code117.v

```
module m_proc06 (w_clk, w_btnu, w_btnd, w_led);
  input  wire w_clk, w_btnu, w_btnd;
  output wire [15:0] w_led;

  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==6'h4 && 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_taken) ? w_tpc : w_npc;
  m_amemory 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;

  assign w_led = (w_btnu | w_btnd) ? w_rslt[31:16] : w_rslt[15:0];
endmodule
```

Inside module m_proc06

- main.vを code{093, 111, 118, 117}.v となるように編集して，シミュレーションする．