```
//============================================================================--
//
//  S Y N T H E Z I A B L E    CPU01   C O R E
//
//  www.OpenCores.Org - December 2002
//  This core adheres to the GNU public license
//
// File name      : cpu01.vhd
//
// Purpose        : Implements a 6801 compatible CPU core
//
// Dependencies   : ieee.Std_Logic_1164
//                  ieee.std_logic_unsigned
//
// Author         : John E. Kent
//
//============================================================================----
//
// Revision History:
//
// Date:          Revision         Author
// 22 Sep 2002    0.1              John Kent
//
// 30 Oct 2002    0.2              John Kent
// made NMI edge triggered
//
// 30 Oct 2002    0.3              John Kent
// more corrections to NMI
// added wai_wait_state to prevent stack overflow on wai.
//
//  1 Nov 2002    0.4              John Kent
// removed WAI states and integrated WAI with the interrupt service routine
// replace Data out (do) and Data in (di) register with a single Memory Data (md) reg.
// Added Multiply instruction states.
// run ALU and CC out of CPU module for timing measurements.
//
//  3 Nov 2002    0.5              John Kent
// Memory Data Register was not loaded on Store instructions
// SEV and CLV were not defined in the ALU
// Overflow Flag on NEG was incorrect
//
// 16th Feb 2003  0.6              John Kent
// Rearranged the execution cycle for dual operand instructions
// so that occurs during the following fetch cycle.
// This allows the reduction of one clock cycle from dual operand
// instruction. Note that this also necessitated re-arranging the
// program counter so that it is no longer incremented in the ALU.
// The effective address has also been re-arranged to include a
// separate added. The STD (store accd) now sets the condition codes.
//
// 28th Jun 2003 0.7              John Kent
// Added Hold and Halt signals. Hold is used to steal cycles from the
// CPU or add wait states. Halt puts the CPU in the inactive state
// and is only honoured in the fetch cycle. Both signals are active high.
//
// 24 Aug 2003 1.0               John Kent
// Converted 6800 core to 6801 by removing alu_cpx
// Also added 4 extra interrupt inputs
//
// 16 January 2004 1.1           John Kent (by Michael Hasenfratz)
// Failure to clear carry bit during CLR instructions
// Corrected CLR instructions to set alu_ctrl to alu_clr instead of alu_ld8.
//

module cpu01(
clk,
rst,
rw,
vma,
address,
data_in,
data_out,
hold,
halt,
irq,
nmi,
irq_icf,
irq_ocf,
irq_tof,
irq_sci,
test_alu,
test_cc
);

input clk;
input rst;
output rw;
output vma;
output[15:0] address;
input[7:0] data_in;
output[7:0] data_out;
```

```
input hold;
input halt;
input irq;
input nmi;
input irq_icf;
input irq_ocf;
input irq_tof;
input irq_sci;
output[15:0] test_alu;
output[7:0] test_cc;

wire   clk;
wire   rst;
reg   rw;
reg   vma;
reg  [15:0] address;
wire  [7:0] data_in;
reg  [7:0] data_out;
wire   hold;
wire   halt;
wire   irq;
wire   nmi;
wire   irq_icf;
wire   irq_ocf;
wire   irq_tof;
wire   irq_sci;
reg  [15:0] test_alu;
reg  [7:0] test_cc;


parameter SBIT = 7;
parameter XBIT = 6;
parameter HBIT = 5;
parameter IBIT = 4;
parameter NBIT = 3;
parameter ZBIT = 2;
parameter VBIT = 1;
parameter CBIT = 0;
parameter [5:0]
  reset_state = 0,
  fetch_state = 1,
  decode_state = 2,
  extended_state = 3,
  indexed_state = 4,
  read8_state = 5,
  read16_state = 6,
  immediate16_state = 7,
  write8_state = 8,
  write16_state = 9,
  execute_state = 10,
  halt_state = 11,
  error_state = 12,
  mul_state = 13,
  mulea_state = 14,
  muld_state = 15,
  mul0_state = 16,
  mul1_state = 17,
  mul2_state = 18,
  mul3_state = 19,
  mul4_state = 20,
  mul5_state = 21,
  mul6_state = 22,
  mul7_state = 23,
  jmp_state = 24,
  jsr_state = 25,
  jsr1_state = 26,
  branch_state = 27,
  bsr_state = 28,
  bsr1_state = 29,
  rts_hi_state = 30,
  rts_lo_state = 31,
  int_pcl_state = 32,
  int_pch_state = 33,
  int_ixl_state = 34,
  int_ixh_state = 35,
  int_cc_state = 36,
  int_acca_state = 37,
  int_accb_state = 38,
  int_wai_state = 39,
  int_mask_state = 40,
  rti_state = 41,
  rti_cc_state = 42,
  rti_acca_state = 43,
  rti_accb_state = 44,
  rti_ixl_state = 45,
  rti_ixh_state = 46,
  rti_pcl_state = 47,
  rti_pch_state = 48,
  pula_state = 49,
  psha_state = 50,
  pulb_state = 51,
```

```
  pshb_state = 52,
  pulx_lo_state = 53,
  pulx_hi_state = 54,
  pshx_lo_state = 55,
  pshx_hi_state = 56,
  vect_lo_state = 57,
  vect_hi_state = 58;
parameter [2:0]
  idle_ad = 0,
  fetch_ad = 1,
  read_ad = 2,
  write_ad = 3,
  push_ad = 4,
  pull_ad = 5,
  int_hi_ad = 6,
  int_lo_ad = 7;
parameter [3:0]
  md_lo_dout = 0,
  md_hi_dout = 1,
  acca_dout = 2,
  accb_dout = 3,
  ix_lo_dout = 4,
  ix_hi_dout = 5,
  cc_dout = 6,
  pc_lo_dout = 7,
  pc_hi_dout = 8;
parameter [1:0]
  reset_op = 0,
  fetch_op = 1,
  latch_op = 2;
parameter [2:0]
  reset_acca = 0,
  load_acca = 1,
  load_hi_acca = 2,
  pull_acca = 3,
  latch_acca = 4;
parameter [1:0]
  reset_accb = 0,
  load_accb = 1,
  pull_accb = 2,
  latch_accb = 3;
parameter [1:0]
  reset_cc = 0,
  load_cc = 1,
  pull_cc = 2,
  latch_cc = 3;
parameter [2:0]
  reset_ix = 0,
  load_ix = 1,
  pull_lo_ix = 2,
  pull_hi_ix = 3,
  latch_ix = 4;
parameter [1:0]
  reset_sp = 0,
  latch_sp = 1,
  load_sp = 2;
parameter [2:0]
  reset_pc = 0,
  latch_pc = 1,
  load_ea_pc = 2,
  add_ea_pc = 3,
  pull_lo_pc = 4,
  pull_hi_pc = 5,
  inc_pc = 6;
parameter [2:0]
  reset_md = 0,
  latch_md = 1,
  load_md = 2,
  fetch_first_md = 3,
  fetch_next_md = 4,
  shiftl_md = 5;
parameter [2:0]
  reset_ea = 0,
  latch_ea = 1,
  add_ix_ea = 2,
  load_accb_ea = 3,
  inc_ea = 4,
  fetch_first_ea = 5,
  fetch_next_ea = 6;
parameter [3:0]
  reset_iv = 0,
  latch_iv = 1,
  swi_iv = 2,
  nmi_iv = 3,
  irq_iv = 4,
  icf_iv = 5,
  ocf_iv = 6,
  tof_iv = 7,
  sci_iv = 8;
parameter [1:0]
  reset_nmi = 0,
```

```
  set_nmi = 1,
  latch_nmi = 2;
parameter [2:0]
  acca_left = 0,
  accb_left = 1,
  accd_left = 2,
  md_left = 3,
  ix_left = 4,
  sp_left = 5;
parameter [1:0]
  md_right = 0,
  zero_right = 1,
  plus_one_right = 2,
  accb_right = 3;
parameter [5:0]
  alu_add8 = 0,
  alu_sub8 = 1,
  alu_add16 = 2,
  alu_sub16 = 3,
  alu_adc = 4,
  alu_sbc = 5,
  alu_and = 6,
  alu_ora = 7,
  alu_eor = 8,
  alu_tst = 9,
  alu_inc = 10,
  alu_dec = 11,
  alu_clr = 12,
  alu_neg = 13,
  alu_com = 14,
  alu_inx = 15,
  alu_dex = 16,
  alu_lsr16 = 17,
  alu_lsl16 = 18,
  alu_ror8 = 19,
  alu_rol8 = 20,
  alu_asr8 = 21,
  alu_asl8 = 22,
  alu_lsr8 = 23,
  alu_sei = 24,
  alu_cli = 25,
  alu_sec = 26,
  alu_clc = 27,
  alu_sev = 28,
  alu_clv = 29,
  alu_tpa = 30,
  alu_tap = 31,
  alu_ld8 = 32,
  alu_st8 = 33,
  alu_ld16 = 34,
  alu_st16 = 35,
  alu_nop = 36,
  alu_daa = 37;
reg [7:0] op_code;
reg [7:0] acca;
reg [7:0] accb;
reg [7:0] cc;
reg [7:0] cc_out;
reg [15:0] xreg;
reg [15:0] sp;
reg [15:0] ea;
reg [15:0] pc;
reg [15:0] md;
reg [15:0] left;
reg [15:0] right;
reg [15:0] out_alu;
reg [2:0] iv;
reg  nmi_req;
reg  nmi_ack;
reg [5:0] state;
reg [5:0] next_state;
reg [2:0] pc_ctrl;
reg [2:0] ea_ctrl;
reg [1:0] op_ctrl;
reg [2:0] md_ctrl;
reg [2:0] acca_ctrl;
reg [1:0] accb_ctrl;
reg [2:0] ix_ctrl;
reg [1:0] cc_ctrl;
reg [1:0] sp_ctrl;
reg [3:0] iv_ctrl;
reg [2:0] left_ctrl;
reg [1:0] right_ctrl;
reg [5:0] alu_ctrl;
reg [2:0] addr_ctrl;
reg [3:0] dout_ctrl;
reg [1:0] nmi_ctrl;

  //--------------------------------
  //
  // Address bus multiplexer
```

```
  //
  //---------------------------------

/*
case (4_bit_expression)
    4'b0000 :
    begin
        statement1;
    end
    4'b1010:
    begin
        statement2;
    end
    default :
    begin
        statement3;
    end
endcase */
  always @ (negedge clk) //(clk or addr_ctrl or pc or ea or sp or iv)
    begin
    case(addr_ctrl)
    idle_ad : begin
      address <= 16'b1111111111111111;
      vma <= 1'b0;
      rw <= 1'b1;
    end
    fetch_ad : begin
      address <= pc;
      vma <= 1'b1;
      rw <= 1'b1;
    end
    read_ad : begin
      address <= ea;
      vma <= 1'b1;
      rw <= 1'b1;
    end
    write_ad : begin
      address <= ea;
      vma <= 1'b1;
      rw <= 1'b0;
    end
    push_ad : begin
      address <= sp;
      vma <= 1'b1;
      rw <= 1'b0;
    end
    pull_ad : begin
      address <= sp;
      vma <= 1'b1;
      rw <= 1'b1;
    end
    int_hi_ad : begin
      address <= {12'b 111111111111,iv,1'b0};
      vma <= 1'b1;
      rw <= 1'b1;
    end
    int_lo_ad : begin
      address <= {12'b 111111111111,iv,1'b1};
      vma <= 1'b1;
      rw <= 1'b1;
    end
    default : begin
      address <= 16'b1111111111111111;
      vma <= 1'b0;
      rw <= 1'b1;
    end
    endcase
  end

// endmodule

//--------------------------------
//
// Data Bus output
//
//--------------------------------

  always @ (negedge clk) //(clk or dout_ctrl or md or acca or accb or xreg or pc or cc)
    begin
    case(dout_ctrl)
    md_hi_dout : begin
      // alu output
      data_out <= md[15:8] ;
    end
    md_lo_dout : begin
      data_out <= md[7:0] ;
    end
    acca_dout : begin
      // accumulator a
      data_out <= acca;
    end
```

```
    accb_dout : begin
      // accumulator b
      data_out <= accb;
    end
    ix_lo_dout : begin
      // index reg
      data_out <= xreg[7:0] ;
    end
    ix_hi_dout : begin
      // index reg
      data_out <= xreg[15:8] ;
    end
    cc_dout : begin
      // condition codes
      data_out <= cc;
    end
    pc_lo_dout : begin
      // low order pc
      data_out <= pc[7:0] ;
    end
    pc_hi_dout : begin
      // high order pc
      data_out <= pc[15:8] ;
    end
    default : begin
      data_out <= 8'b 00000000;
    end
    endcase
  end

// endmodule

  //-------------------------------
  //
  // Program Counter Control
  //
  //--------------------------------
  always @ (negedge clk) //(clk or pc_ctrl or pc or out_alu or data_in or ea or hold)
  begin
    reg [15:0] tempof;
    reg [15:0] temppc;

    case(pc_ctrl)
    add_ea_pc : begin
      if(ea[7]  == 1'b0) begin
        tempof = {8'b 00000000,ea[7:0] };
      end
      else begin
        tempof = {8'b 11111111,ea[7:0] };
      end
    end
    inc_pc : begin
      tempof = 16'b0000000000000001;
    end
    default : begin
      tempof = 16'b0000000000000000;
    end
    endcase
    case(pc_ctrl)
    reset_pc : begin
      temppc = 16'b1111111111111110;
    end
    load_ea_pc : begin
      temppc = ea;
    end
    pull_lo_pc : begin
      temppc[7:0]  = data_in;
      temppc[15:8]  = pc[15:8] ;
    end
    pull_hi_pc : begin
      temppc[7:0]  = pc[7:0] ;
      temppc[15:8]  = data_in;
    end
    default : begin
      temppc = pc;
    end
    endcase
  end
    //  if clk'event and clk = '0' then
    //    if hold = '1' then
    //      pc <= pc;
    //    else
    //      pc <= temppc + tempof;
    //   end if;
    //  end if;
  always @ (negedge clk)
  begin
        if(hold == 1'b1) pc <= pc;
        else pc <= temppc + tempof;
  end
// endmodule
```

```
  //--------------------------------
  //
  // Effective Address  Control
  //
  //--------------------------------
  always @ (negedge clk) //(clk or ea_ctrl or ea or out_alu or data_in or accb or xreg or hold)
    begin
    reg [15:0] tempind;
    reg [15:0] tempea;

    case(ea_ctrl)
    add_ix_ea : begin
      tempind = {8'b 00000000,ea[7:0] };
    end
    inc_ea : begin
      tempind = 16'b0000000000000001;
    end
    default : begin
      tempind = 16'b0000000000000000;
    end
    endcase
    case(ea_ctrl)
    reset_ea : begin
      tempea = 16'b0000000000000000;
    end
    load_accb_ea : begin
      tempea = {8'b 00000000,accb[7:0] };
    end
    add_ix_ea : begin
      tempea = xreg;
    end
    fetch_first_ea : begin
      tempea[7:0]  = data_in;
      tempea[15:8]  = 8'b 00000000;
    end
    fetch_next_ea : begin
      tempea[7:0]  = data_in;
      tempea[15:8]  = ea[7:0] ;
    end
    default : begin
      tempea = ea;
    end
    endcase
  end
    //  if clk'event and clk = '0' then
    //    if hold = '1' then
    //      ea <= ea;
    //    else
    //      ea <= tempea + tempind;
    //   end if;
    //  end if;
  always @ (negedge clk)
  begin
        if(hold == 1'b1) ea <= ea;
        else ea <= tempea + tempind;
  end

  //--------------------------------
  //
  // Accumulator A
  //
  //--------------------------------
  always @ (negedge clk) //(negedge clk or negedge acca_ctrl or negedge out_alu or negedge acca or negedge data_
in or negedge hold)
    begin
    if(hold == 1'b1) begin
      acca <= acca;
    end
    else begin
      case(acca_ctrl)
      reset_acca : begin
        acca <= 8'b 00000000;
      end
      load_acca : begin
        acca <= out_alu[7:0] ;
      end
      load_hi_acca : begin
        acca <= out_alu[15:8] ;
      end
      pull_acca : begin
        acca <= data_in;
      end
      default : begin
        //   when latch_acca =>
        acca <= acca;
      end
      endcase
    end
  end
// endmodule
```

```
  //--------------------------------
  //
  // Accumulator B
  //
  //--------------------------------
  always @ (negedge clk) //(negedge clk or negedge accb_ctrl or negedge out_alu or negedge accb or negedge data_
in or negedge hold)
    begin
    if(hold == 1'b1) begin
      accb <= accb;
    end
    else begin
      case(accb_ctrl)
      reset_accb : begin
        accb <= 8'b 00000000;
      end
      load_accb : begin
        accb <= out_alu[7:0] ;
      end
      pull_accb : begin
        accb <= data_in;
      end
      default : begin
        //   when latch_accb =>
        accb <= accb;
      end
      endcase
    end
  end
// endmodule

  //--------------------------------
  //
  // X Index register
  //
  //--------------------------------
  always @ (negedge clk) //(negedge clk or negedge ix_ctrl or negedge out_alu or negedge xreg or negedge data_in
 or negedge hold)
    begin
    if(hold == 1'b1) begin
      xreg <= xreg;
    end
    else begin
      case(ix_ctrl)
      reset_ix : begin
        xreg <= 16'b0000000000000000;
      end
      load_ix : begin
        xreg <= out_alu[15:0] ;
      end
      pull_hi_ix : begin
        xreg[15:8]  <= data_in;
      end
      pull_lo_ix : begin
        xreg[7:0]  <= data_in;
      end
      default : begin
        //   when latch_ix =>
        xreg <= xreg;
      end
      endcase
    end
  end
// endmodule

  //--------------------------------
  //
  // stack pointer
  //
  //--------------------------------
  always @ (negedge clk) //(negedge clk or negedge sp_ctrl or negedge out_alu or negedge hold)
    begin
    if(hold == 1'b1) begin
      sp <= sp;
    end
    else begin
      case(sp_ctrl)
      reset_sp : begin
        sp <= 16'b0000000000000000;
      end
      load_sp : begin
        sp <= out_alu[15:0] ;
      end
      default : begin
        //   when latch_sp =>
        sp <= sp;
      end
      endcase
    end
  end
```

```
// endmodule

  //------------------------------
  //
  // Memory Data
  //
  //------------------------------
  always @ (negedge clk) //(negedge clk or negedge md_ctrl or negedge out_alu or negedge data_in or negedge md o
r negedge hold)
    begin
    if(hold == 1'b1) begin
      md <= md;
    end
    else begin
      case(md_ctrl)
      reset_md : begin
        md <= 16'b0000000000000000;
      end
      load_md : begin
        md <= out_alu[15:0] ;
      end
      fetch_first_md : begin
        md[15:8]  <= 8'b 00000000;
        md[7:0]  <= data_in[7:0];
      end
      fetch_next_md : begin
        md[15:8]  <= md[7:0] ;
        md[7:0]  <= data_in[7:0];
      end
      shiftl_md : begin
        md[15:1]  <= md[14:0] ;
        md[0]  <= 1'b0;
      end
      default : begin
        //   when latch_md =>
        md <= md;
      end
      endcase
    end
  end
// endmodule

  //------------------------------
  //
  // Condition Codes
  //
  //--------------------------------
  always @ (negedge clk) //(negedge clk or negedge cc_ctrl or negedge cc_out or negedge cc or negedge data_in or
 negedge hold)
    begin
    if(hold == 1'b1) begin
      cc <= cc;
    end
    else begin
      case(cc_ctrl)
      reset_cc : begin
        cc <= 8'b 11000000;
      end
      load_cc : begin
        cc <= cc_out;
      end
      pull_cc : begin
        cc <= data_in;
      end
      default : begin
        //  when latch_cc =>
        cc <= cc;
      end
      endcase
    end
  end
// endmodule

  //------------------------------
  //
  // interrupt vector
  //
  //--------------------------------
  always @ (negedge clk) //(negedge clk or negedge iv_ctrl or negedge hold)
    begin
    if(hold == 1'b1) begin
      iv <= iv;
    end
    else begin
      case(iv_ctrl)
      reset_iv : begin
        iv <= 3'b 111;
      end
      nmi_iv : begin
        iv <= 3'b 110;
      end
```

```
      swi_iv : begin
        iv <= 3'b 101;
      end
      irq_iv : begin
        iv <= 3'b 100;
      end
      icf_iv : begin
        iv <= 3'b 011;
      end
      ocf_iv : begin
        iv <= 3'b 010;
      end
      tof_iv : begin
        iv <= 3'b 001;
      end
      sci_iv : begin
        iv <= 3'b 000;
      end
      default : begin
        iv <= iv;
      end
      endcase
    end
  end
// endmodule

//-------------------------------
//
// op code fetch
//
//---------------------------------

  always @ (negedge clk) //(negedge clk or negedge data_in or negedge op_ctrl or negedge op_code or negedge hold
)
    begin
    if(hold == 1'b1) begin
      op_code <= op_code;
    end
    else begin
      case(op_ctrl)
      reset_op : begin
        op_code <= 8'b 00000001;
        // nop
      end
      fetch_op : begin
        op_code <= data_in;
      end
      default : begin
        //   when latch_op =>
        op_code <= op_code;
      end
      endcase
    end
  end
// endmodule

  //-------------------------------
  //
  // Left Mux
  //
  //---------------------------------
  always @ (negedge clk) //(left_ctrl or acca or accb or xreg or sp or pc or ea or md)
    begin
    case(left_ctrl)
    acca_left : begin
      left[15:8]  <= 8'b 00000000;
      left[7:0]  <= acca;
    end
    accb_left : begin
      left[15:8]  <= 8'b 00000000;
      left[7:0]  <= accb;
    end
    accd_left : begin
      left[15:8]  <= acca;
      left[7:0]  <= accb;
    end
    ix_left : begin
      left <= xreg;
    end
    sp_left : begin
      left <= sp;
    end
    default : begin
      //     when md_left =>
      left <= md;
    end
    endcase
  end
// endmodule

  //-------------------------------
```

```
  //
  // Right Mux
  //
  //--------------------------------
  always @ (negedge clk) //(right_ctrl or data_in or md or accb or ea)
    begin
    case(right_ctrl)
    zero_right : begin
      right <= 16'b0000000000000000;
    end
    plus_one_right : begin
      right <= 16'b0000000000000001;
    end
    accb_right : begin
      right <= {8'b 00000000,accb};
    end
    default : begin
      //     when md_right =>
      right <= md;
    end
    endcase
  end
// endmodule

  //------------------------------
  //
  // Arithmetic Logic Unit
  //
  //--------------------------------
  always @ (negedge clk) //(alu_ctrl or cc or left or right or out_alu or cc_out)
    begin
    reg  valid_lo;
  //boolean;
    reg  valid_hi;
  //boolean;
    reg  carry_in;
    reg [7:0] daa_reg;

    case(alu_ctrl)
    alu_adc,alu_sbc,alu_rol8,alu_ror8 : begin
      carry_in = cc[CBIT] ;
    end
    default : begin
      carry_in = 1'b0;
    end
    endcase
    //  valid_lo := left(3 downto 0) <= 9;
    if((left[3:0]  <= 9)) begin
      valid_lo = 1'b1;
    end
    else begin
      valid_lo = 1'b0;
    end
    //  valid_hi := left(7 downto 4) <= 9;
    if((left[7:4]  <= 9)) begin
      valid_hi = 1'b1;
    end
    else begin
      valid_hi = 1'b0;
    end
    if((cc[CBIT]  == 1'b0)) begin
      if((cc[HBIT]  == 1'b1)) begin
        if(valid_hi == 1'b1) begin
          daa_reg = 8'b 00000110;
        end
        else begin
          daa_reg = 8'b 01100110;
        end
      end
      else begin
        if(valid_lo == 1'b1) begin
          if(valid_hi == 1'b1) begin
            daa_reg = 8'b 00000000;
          end
          else begin
            daa_reg = 8'b 01100000;
          end
        end
        else begin
          if((left[7:4]  <= 8)) begin
            daa_reg = 8'b 00000110;
          end
          else begin
            daa_reg = 8'b 01100110;
          end
        end
      end
    end
    else begin
      if((cc[HBIT]  == 1'b1)) begin
        daa_reg = 8'b 01100110;
```

```
      end
      else begin
        if(valid_lo == 1'b1) begin
          daa_reg = 8'b 01100000;
        end
        else begin
          daa_reg = 8'b 01100110;
        end
      end
    end
    case(alu_ctrl)
    alu_add8,alu_inc,alu_add16,alu_inx,alu_adc : begin
      out_alu <= left + right + ({15'b 000000000000000,carry_in});
    end
    alu_sub8,alu_dec,alu_sub16,alu_dex,alu_sbc : begin
      out_alu <= left - right - ({15'b 000000000000000,carry_in});
    end
    alu_and : begin
      out_alu <= left & right;
      // and/bit
    end
    alu_ora : begin
      out_alu <= left | right;
      // or
    end
    alu_eor : begin
      out_alu <= left ^ right;
      // eor/xor
    end
    alu_lsl16,alu_asl8,alu_rol8 : begin
      out_alu <= {left[14:0] ,carry_in};
      // rol8/asl8/lsl16
    end
    alu_lsr16,alu_lsr8 : begin
      out_alu <= {carry_in,left[15:1] };
      // lsr
    end
    alu_ror8 : begin
      out_alu <= {8'b 00000000,carry_in,left[7:1] };
      // ror
    end
    alu_asr8 : begin
      out_alu <= {8'b 00000000,left[7] ,left[7:1] };
      // asr
    end
    alu_neg : begin
      out_alu <= right - left;
      // neg (right=0)
    end
    alu_com : begin
      out_alu <=  ~left;
    end
    alu_clr,alu_ld8,alu_ld16 : begin
      out_alu <= right;
      // clr, ld
    end
    alu_st8,alu_st16 : begin
      out_alu <= left;
    end
    alu_daa : begin
      out_alu <= left + ({8'b 00000000,daa_reg});
    end
    alu_tpa : begin
      out_alu <= {8'b 00000000,cc};
    end
    default : begin
      out_alu <= left;
      // nop
    end
    endcase
    //
    // carry bit
    //
    case(alu_ctrl)
    alu_add8,alu_adc : begin
      cc_out[CBIT]  <= ((left[7]  & right[7] )) | ((left[7]  &  ~out_alu[7] )) | ((right[7]  &  ~out_alu[7] ));
    end
    alu_sub8,alu_sbc : begin
      cc_out[CBIT]  <= ((( ~left[7] )) & right[7] )) | ((( ~left[7] )) & out_alu[7] )) | ((right[7]  & out_alu
[7] ));
    end
    alu_add16 : begin
      cc_out[CBIT]  <= ((left[15]  & right[15] )) | ((left[15]  &  ~out_alu[15] )) | ((right[15]  &  ~out_alu[15
] ));
    end
    alu_sub16 : begin
      cc_out[CBIT]  <= ((( ~left[15] )) & right[15] )) | ((( ~left[15] )) & out_alu[15] )) | ((right[15]  & ou
t_alu[15] ));
    end
    alu_ror8,alu_lsr16,alu_lsr8,alu_asr8 : begin
      cc_out[CBIT]  <= left[0] ;
```

```
    end
    alu_rol8,alu_asl8 : begin
      cc_out[CBIT]  <= left[7] ;
    end
    alu_lsl16 : begin
      cc_out[CBIT]  <= left[15] ;
    end
    alu_com : begin
      cc_out[CBIT]  <= 1'b1;
    end
    alu_neg,alu_clr : begin
      cc_out[CBIT]  <= out_alu[7]  | out_alu[6]  | out_alu[5]  | out_alu[4]  | out_alu[3]  | out_alu[2]  | out_a
lu[1]  | out_alu[0] ;
    end
    alu_daa : begin
      if((daa_reg[7:4]  == 4'b 0110)) begin
        cc_out[CBIT]  <= 1'b1;
      end
      else begin
        cc_out[CBIT]  <= 1'b0;
      end
    end
    alu_sec : begin
      cc_out[CBIT]  <= 1'b1;
    end
    alu_clc : begin
      cc_out[CBIT]  <= 1'b0;
    end
    alu_tap : begin
      cc_out[CBIT]  <= left[CBIT] ;
    end
    default : begin
      cc_out[CBIT]  <= cc[CBIT] ;
    end
    endcase
    //
    // Zero flag
    //
    case(alu_ctrl)
    alu_add8,alu_sub8,alu_adc,alu_sbc,alu_and,alu_ora,alu_eor,alu_inc,alu_dec,alu_neg,alu_com,alu_clr,alu_rol8,a
lu_ror8,alu_asr8,alu_asl8,alu_lsr8,alu_ld8,alu_st8 : begin
      cc_out[ZBIT]  <=  ~((out_alu[7]  | out_alu[6]  | out_alu[5]  | out_alu[4]  | out_alu[3]  | out_alu[2]  | o
ut_alu[1]  | out_alu[0] ));
    end
    alu_add16,alu_sub16,alu_lsl16,alu_lsr16,alu_inx,alu_dex,alu_ld16,alu_st16 : begin
      cc_out[ZBIT]  <=  ~((out_alu[15]  | out_alu[14]  | out_alu[13]  | out_alu[12]  | out_alu[11]  | out_alu[10
]  | out_alu[9]  | out_alu[8]  | out_alu[7]  | out_alu[6]  | out_alu[5]  | out_alu[4]  | out_alu[3]  | out_alu[2
]  | out_alu[1]  | out_alu[0] ));
    end
    alu_tap : begin
      cc_out[ZBIT]  <= left[ZBIT] ;
    end
    default : begin
      cc_out[ZBIT]  <= cc[ZBIT] ;
    end
    endcase
    //
    // negative flag
    //
    case(alu_ctrl)
    alu_add8,alu_sub8,alu_adc,alu_sbc,alu_and,alu_ora,alu_eor,alu_rol8,alu_ror8,alu_asr8,alu_asl8,alu_lsr8,alu_i
nc,alu_dec,alu_neg,alu_com,alu_clr,alu_ld8,alu_st8 : begin
      cc_out[NBIT]  <= out_alu[7] ;
    end
    alu_add16,alu_sub16,alu_lsl16,alu_lsr16,alu_ld16,alu_st16 : begin
      cc_out[NBIT]  <= out_alu[15] ;
    end
    alu_tap : begin
      cc_out[NBIT]  <= left[NBIT] ;
    end
    default : begin
      cc_out[NBIT]  <= cc[NBIT] ;
    end
    endcase
    //
    // Interrupt mask flag
    //
    case(alu_ctrl)
    alu_sei : begin
      cc_out[IBIT]  <= 1'b1;
      // set interrupt mask
    end
    alu_cli : begin
      cc_out[IBIT]  <= 1'b0;
      // clear interrupt mask
    end
    alu_tap : begin
      cc_out[IBIT]  <= left[IBIT] ;
    end
    default : begin
      cc_out[IBIT]  <= cc[IBIT] ;
```

```
      // interrupt mask
    end
    endcase
    //
    // Half Carry flag
    //
    case(alu_ctrl)
    alu_add8,alu_adc : begin
      cc_out[HBIT]  <= ((left[3]  & right[3] )) | ((right[3]  &  ~out_alu[3] )) | ((left[3]  &  ~out_alu[3] ));
    end
    alu_tap : begin
      cc_out[HBIT]  <= left[HBIT] ;
    end
    default : begin
      cc_out[HBIT]  <= cc[HBIT] ;
    end
    endcase
    //
    // Overflow flag
    //
    case(alu_ctrl)
    alu_add8,alu_adc : begin
      cc_out[VBIT]  <= ((left[7]  & right[7]  & (( ~out_alu[7] )))) | ((((  ~left[7] )) & (( ~right[7] )) & out_a
lu[7] ));
    end
    alu_sub8,alu_sbc : begin
      cc_out[VBIT]  <= ((left[7]  & (( ~right[7] )) & (( ~out_alu[7] )))) | ((((  ~left[7] )) & right[7]  & out_a
lu[7] ));
    end
    alu_add16 : begin
      cc_out[VBIT]  <= ((left[15]  & right[15]  & (( ~out_alu[15] )))) | ((((  ~left[15] )) & (( ~right[15] )) &
out_alu[15] ));
    end
    alu_sub16 : begin
      cc_out[VBIT]  <= ((left[15]  & (( ~right[15] )) & (( ~out_alu[15] )))) | ((((  ~left[15] )) & right[15]  &
out_alu[15] ));
    end
    alu_inc : begin
      cc_out[VBIT]  <= ((( ~left[7] )) & left[6]  & left[5]  & left[4]  & left[3]  & left[2]  & left[1]  & left[
0] );
    end
    alu_dec,alu_neg : begin
      cc_out[VBIT]  <= (left[7]  & (( ~left[6] )) & (( ~left[5] )) & (( ~left[4] )) & (( ~left[3] )) & (( ~left[
2] )) & (( ~left[1] )) & (( ~left[0] )));
    end
    alu_asr8 : begin
      cc_out[VBIT]  <= left[0]  ^ left[7] ;
    end
    alu_lsr8,alu_lsr16 : begin
      cc_out[VBIT]  <= left[0] ;
    end
    alu_ror8 : begin
      cc_out[VBIT]  <= left[0]  ^ cc[CBIT] ;
    end
    alu_lsl16 : begin
      cc_out[VBIT]  <= left[15]  ^ left[14] ;
    end
    alu_rol8,alu_asl8 : begin
      cc_out[VBIT]  <= left[7]  ^ left[6] ;
    end
    alu_tap : begin
      cc_out[VBIT]  <= left[VBIT] ;
    end
    alu_and,alu_ora,alu_eor,alu_com,alu_st8,alu_st16,alu_ld8,alu_ld16,alu_clv : begin
      cc_out[VBIT]  <= 1'b0;
    end
    alu_sev : begin
      cc_out[VBIT]  <= 1'b1;
    end
    default : begin
      cc_out[VBIT]  <= cc[VBIT] ;
    end
    endcase
    case(alu_ctrl)
    alu_tap : begin
      cc_out[XBIT]  <= cc[XBIT]  & left[XBIT] ;
      cc_out[SBIT]  <= left[SBIT] ;
    end
    default : begin
      cc_out[XBIT]  <= cc[XBIT]  & left[XBIT] ;
      cc_out[SBIT]  <= cc[SBIT] ;
    end
    endcase
    test_alu <= out_alu;
    test_cc <= cc_out;
  end
// endmodule

  //------------------------------
  //
  // Detect Edge of NMI interrupt
```

```
  //
  //-----------------------------------
  always @ (negedge clk) //(negedge clk or negedge rst or negedge nmi or negedge nmi_ack)
    begin
    if(hold == 1'b1) begin
      nmi_req <= nmi_req;
    end
    else begin
      if(rst == 1'b1) begin
        nmi_req <= 1'b0;
      end
      else begin
        if((nmi == 1'b1) && (nmi_ack == 1'b0)) begin
          nmi_req <= 1'b1;
        end
        else begin
          if((nmi == 1'b0) && (nmi_ack == 1'b1)) begin
            nmi_req <= 1'b0;
          end
          else begin
            nmi_req <= nmi_req;
          end
        end
      end
    end
  end
// endmodule

  //-------------------------------
  //
  // Nmi mux
  //
  //-----------------------------------
  always @ (negedge clk) //(negedge clk or negedge nmi_ctrl or negedge nmi_ack or negedge hold)
    begin
    if(hold == 1'b1) begin
      nmi_ack <= nmi_ack;
    end
    else begin
      case(nmi_ctrl)
      set_nmi : begin
        nmi_ack <= 1'b1;
      end
      reset_nmi : begin
        nmi_ack <= 1'b0;
      end
      default : begin
        //  when latch_nmi =>
        nmi_ack <= nmi_ack;
      end
      endcase
    end
  end
// endmodule

  //-------------------------------
  //
  // state sequencer
  //
  //-----------------------------------
  always @ (negedge clk) //(state or op_code or cc or ea or irq or irq_icf or irq_ocf or irq_tof or irq_sci or n
mi_req or nmi_ack or hold or halt)
    begin
    case(state)
    reset_state : begin
      //  released from reset
      // reset the registers
      op_ctrl <= reset_op;
      acca_ctrl <= reset_acca;
      accb_ctrl <= reset_accb;
      ix_ctrl <= reset_ix;
      sp_ctrl <= reset_sp;
      pc_ctrl <= reset_pc;
      ea_ctrl <= reset_ea;
      md_ctrl <= reset_md;
      iv_ctrl <= reset_iv;
      nmi_ctrl <= reset_nmi;
      // idle the ALU
      left_ctrl <= acca_left;
      right_ctrl <= zero_right;
      alu_ctrl <= alu_nop;
      cc_ctrl <= reset_cc;
      // idle the bus
      dout_ctrl <= md_lo_dout;
      addr_ctrl <= idle_ad;
      next_state <= vect_hi_state;
      //
      // Jump via interrupt vector
      // iv holds interrupt type
      // fetch PC hi from vector location
      //
```

```
    end
    vect_hi_state : begin
      // default the registers
      op_ctrl <= latch_op;
      nmi_ctrl <= latch_nmi;
      acca_ctrl <= latch_acca;
      accb_ctrl <= latch_accb;
      ix_ctrl <= latch_ix;
      sp_ctrl <= latch_sp;
      md_ctrl <= latch_md;
      ea_ctrl <= latch_ea;
      iv_ctrl <= latch_iv;
      // idle the ALU
      left_ctrl <= acca_left;
      right_ctrl <= zero_right;
      alu_ctrl <= alu_nop;
      cc_ctrl <= latch_cc;
      // fetch pc low interrupt vector
      pc_ctrl <= pull_hi_pc;
      addr_ctrl <= int_hi_ad;
      dout_ctrl <= pc_hi_dout;
      next_state <= vect_lo_state;
      //
      // jump via interrupt vector
      // iv holds vector type
      // fetch PC lo from vector location
      //
    end
    vect_lo_state : begin
      // default the registers
      op_ctrl <= latch_op;
      nmi_ctrl <= latch_nmi;
      acca_ctrl <= latch_acca;
      accb_ctrl <= latch_accb;
      ix_ctrl <= latch_ix;
      sp_ctrl <= latch_sp;
      md_ctrl <= latch_md;
      ea_ctrl <= latch_ea;
      iv_ctrl <= latch_iv;
      // idle the ALU
      left_ctrl <= acca_left;
      right_ctrl <= zero_right;
      alu_ctrl <= alu_nop;
      cc_ctrl <= latch_cc;
      // fetch the vector low byte
      pc_ctrl <= pull_lo_pc;
      addr_ctrl <= int_lo_ad;
      dout_ctrl <= pc_lo_dout;
      next_state <= fetch_state;
      //
      // Here to fetch an instruction
      // PC points to opcode
      // Should service interrupt requests at this point
      // either from the timer
      // or from the external input.
      //
      // branch conditional
      // acca single op
      // accb single op
      // indexed single op
      // extended single op
      // idle ALU
    end
    fetch_state : begin
      case(op_code[7:4] )
      4'b 0000,4'b 0001,4'b 0010,4'b 0011,4'b 0100,4'b 0101,4'b 0110,4'b 0111 : begin
        left_ctrl <= acca_left;
        right_ctrl <= zero_right;
        alu_ctrl <= alu_nop;
        cc_ctrl <= latch_cc;
        acca_ctrl <= latch_acca;
        accb_ctrl <= latch_accb;
        ix_ctrl <= latch_ix;
        sp_ctrl <= latch_sp;
        // acca immediate
        // acca direct
        // acca indexed
        // acca extended
      end
      4'b 1000,4'b 1001,4'b 1010,4'b 1011 : begin
        case(op_code[3:0] )
        4'b 0000 : begin
          // suba
          left_ctrl <= acca_left;
          right_ctrl <= md_right;
          alu_ctrl <= alu_sub8;
          cc_ctrl <= load_cc;
          acca_ctrl <= load_acca;
          accb_ctrl <= latch_accb;
          ix_ctrl <= latch_ix;
          sp_ctrl <= latch_sp;
```

```
        end
        4'b 0001 : begin
          // cmpa
          left_ctrl <= acca_left;
          right_ctrl <= md_right;
          alu_ctrl <= alu_sub8;
          cc_ctrl <= load_cc;
          acca_ctrl <= latch_acca;
          accb_ctrl <= latch_accb;
          ix_ctrl <= latch_ix;
          sp_ctrl <= latch_sp;
        end
        4'b 0010 : begin
          // sbca
          left_ctrl <= acca_left;
          right_ctrl <= md_right;
          alu_ctrl <= alu_sbc;
          cc_ctrl <= load_cc;
          acca_ctrl <= load_acca;
          accb_ctrl <= latch_accb;
          ix_ctrl <= latch_ix;
          sp_ctrl <= latch_sp;
        end
        4'b 0011 : begin
          // subd
          left_ctrl <= accd_left;
          right_ctrl <= md_right;
          alu_ctrl <= alu_sub16;
          cc_ctrl <= load_cc;
          acca_ctrl <= load_hi_acca;
          accb_ctrl <= load_accb;
          ix_ctrl <= latch_ix;
          sp_ctrl <= latch_sp;
        end
        4'b 0100 : begin
          // anda
          left_ctrl <= acca_left;
          right_ctrl <= md_right;
          alu_ctrl <= alu_and;
          cc_ctrl <= load_cc;
          acca_ctrl <= load_acca;
          accb_ctrl <= latch_accb;
          ix_ctrl <= latch_ix;
          sp_ctrl <= latch_sp;
        end
        4'b 0101 : begin
          // bita
          left_ctrl <= acca_left;
          right_ctrl <= md_right;
          alu_ctrl <= alu_and;
          cc_ctrl <= load_cc;
          acca_ctrl <= latch_acca;
          accb_ctrl <= latch_accb;
          ix_ctrl <= latch_ix;
          sp_ctrl <= latch_sp;
        end
        4'b 0110 : begin
          // ldaa
          left_ctrl <= acca_left;
          right_ctrl <= md_right;
          alu_ctrl <= alu_ld8;
          cc_ctrl <= load_cc;
          acca_ctrl <= load_acca;
          accb_ctrl <= latch_accb;
          ix_ctrl <= latch_ix;
          sp_ctrl <= latch_sp;
        end
        4'b 0111 : begin
          // staa
          left_ctrl <= acca_left;
          right_ctrl <= md_right;
          alu_ctrl <= alu_st8;
          cc_ctrl <= load_cc;
          acca_ctrl <= latch_acca;
          accb_ctrl <= latch_accb;
          ix_ctrl <= latch_ix;
          sp_ctrl <= latch_sp;
        end
        4'b 1000 : begin
          // eora
          left_ctrl <= acca_left;
          right_ctrl <= md_right;
          alu_ctrl <= alu_eor;
          cc_ctrl <= load_cc;
          acca_ctrl <= load_acca;
          accb_ctrl <= latch_accb;
          ix_ctrl <= latch_ix;
          sp_ctrl <= latch_sp;
        end
        4'b 1001 : begin
          // adca
```

```
          left_ctrl <= acca_left;
          right_ctrl <= md_right;
          alu_ctrl <= alu_adc;
          cc_ctrl <= load_cc;
          acca_ctrl <= load_acca;
          accb_ctrl <= latch_accb;
          ix_ctrl <= latch_ix;
          sp_ctrl <= latch_sp;
        end
        4'b 1010 : begin
          // oraa
          left_ctrl <= acca_left;
          right_ctrl <= md_right;
          alu_ctrl <= alu_ora;
          cc_ctrl <= load_cc;
          acca_ctrl <= load_acca;
          accb_ctrl <= latch_accb;
          ix_ctrl <= latch_ix;
          sp_ctrl <= latch_sp;
        end
        4'b 1011 : begin
          // adda
          left_ctrl <= acca_left;
          right_ctrl <= md_right;
          alu_ctrl <= alu_add8;
          cc_ctrl <= load_cc;
          acca_ctrl <= load_acca;
          accb_ctrl <= latch_accb;
          ix_ctrl <= latch_ix;
          sp_ctrl <= latch_sp;
        end
        4'b 1100 : begin
          // cpx
          left_ctrl <= ix_left;
          right_ctrl <= md_right;
          alu_ctrl <= alu_sub16;
          cc_ctrl <= load_cc;
          acca_ctrl <= latch_acca;
          accb_ctrl <= latch_accb;
          ix_ctrl <= latch_ix;
          sp_ctrl <= latch_sp;
        end
        4'b 1101 : begin
          // bsr / jsr
          left_ctrl <= acca_left;
          right_ctrl <= md_right;
          alu_ctrl <= alu_nop;
          cc_ctrl <= latch_cc;
          acca_ctrl <= latch_acca;
          accb_ctrl <= latch_accb;
          ix_ctrl <= latch_ix;
          sp_ctrl <= latch_sp;
        end
        4'b 1110 : begin
          // lds
          left_ctrl <= sp_left;
          right_ctrl <= md_right;
          alu_ctrl <= alu_ld16;
          cc_ctrl <= load_cc;
          acca_ctrl <= latch_acca;
          accb_ctrl <= latch_accb;
          ix_ctrl <= latch_ix;
          sp_ctrl <= load_sp;
        end
        4'b 1111 : begin
          // sts
          left_ctrl <= sp_left;
          right_ctrl <= md_right;
          alu_ctrl <= alu_st16;
          cc_ctrl <= load_cc;
          acca_ctrl <= latch_acca;
          accb_ctrl <= latch_accb;
          ix_ctrl <= latch_ix;
          sp_ctrl <= latch_sp;
        end
        default : begin
          left_ctrl <= acca_left;
          right_ctrl <= md_right;
          alu_ctrl <= alu_nop;
          cc_ctrl <= latch_cc;
          acca_ctrl <= latch_acca;
          accb_ctrl <= latch_accb;
          ix_ctrl <= latch_ix;
          sp_ctrl <= latch_sp;
        end
        endcase
        // accb immediate
        // accb direct
        // accb indexed
        // accb extended
      end
```

```
      4'b 1100,4'b 1101,4'b 1110,4'b 1111 : begin
        case(op_code[3:0] )
        4'b 0000 : begin
          // subb
          left_ctrl <= accb_left;
          right_ctrl <= md_right;
          alu_ctrl <= alu_sub8;
          cc_ctrl <= load_cc;
          acca_ctrl <= latch_acca;
          accb_ctrl <= load_accb;
          ix_ctrl <= latch_ix;
          sp_ctrl <= latch_sp;
        end
        4'b 0001 : begin
          // cmpb
          left_ctrl <= accb_left;
          right_ctrl <= md_right;
          alu_ctrl <= alu_sub8;
          cc_ctrl <= load_cc;
          acca_ctrl <= latch_acca;
          accb_ctrl <= latch_accb;
          ix_ctrl <= latch_ix;
          sp_ctrl <= latch_sp;
        end
        4'b 0010 : begin
          // sbcb
          left_ctrl <= accb_left;
          right_ctrl <= md_right;
          alu_ctrl <= alu_sbc;
          cc_ctrl <= load_cc;
          acca_ctrl <= latch_acca;
          accb_ctrl <= load_accb;
          ix_ctrl <= latch_ix;
          sp_ctrl <= latch_sp;
        end
        4'b 0011 : begin
          // addd
          left_ctrl <= accd_left;
          right_ctrl <= md_right;
          alu_ctrl <= alu_add16;
          cc_ctrl <= load_cc;
          acca_ctrl <= load_hi_acca;
          accb_ctrl <= load_accb;
          ix_ctrl <= latch_ix;
          sp_ctrl <= latch_sp;
        end
        4'b 0100 : begin
          // andb
          left_ctrl <= accb_left;
          right_ctrl <= md_right;
          alu_ctrl <= alu_and;
          cc_ctrl <= load_cc;
          acca_ctrl <= latch_acca;
          accb_ctrl <= load_accb;
          ix_ctrl <= latch_ix;
          sp_ctrl <= latch_sp;
        end
        4'b 0101 : begin
          // bitb
          left_ctrl <= accb_left;
          right_ctrl <= md_right;
          alu_ctrl <= alu_and;
          cc_ctrl <= load_cc;
          acca_ctrl <= latch_acca;
          accb_ctrl <= latch_accb;
          ix_ctrl <= latch_ix;
          sp_ctrl <= latch_sp;
        end
        4'b 0110 : begin
          // ldab
          left_ctrl <= accb_left;
          right_ctrl <= md_right;
          alu_ctrl <= alu_ld8;
          cc_ctrl <= load_cc;
          acca_ctrl <= latch_acca;
          accb_ctrl <= load_accb;
          ix_ctrl <= latch_ix;
          sp_ctrl <= latch_sp;
        end
        4'b 0111 : begin
          // stab
          left_ctrl <= accb_left;
          right_ctrl <= md_right;
          alu_ctrl <= alu_st8;
          cc_ctrl <= load_cc;
          acca_ctrl <= latch_acca;
          accb_ctrl <= latch_accb;
          ix_ctrl <= latch_ix;
          sp_ctrl <= latch_sp;
        end
        4'b 1000 : begin
```

```
          // eorb
          left_ctrl <= accb_left;
          right_ctrl <= md_right;
          alu_ctrl <= alu_eor;
          cc_ctrl <= load_cc;
          acca_ctrl <= latch_acca;
          accb_ctrl <= load_accb;
          ix_ctrl <= latch_ix;
          sp_ctrl <= latch_sp;
        end
        4'b 1001 : begin
          // adcb
          left_ctrl <= accb_left;
          right_ctrl <= md_right;
          alu_ctrl <= alu_adc;
          cc_ctrl <= load_cc;
          acca_ctrl <= latch_acca;
          accb_ctrl <= load_accb;
          ix_ctrl <= latch_ix;
          sp_ctrl <= latch_sp;
        end
        4'b 1010 : begin
          // orab
          left_ctrl <= accb_left;
          right_ctrl <= md_right;
          alu_ctrl <= alu_ora;
          cc_ctrl <= load_cc;
          acca_ctrl <= latch_acca;
          accb_ctrl <= load_accb;
          ix_ctrl <= latch_ix;
          sp_ctrl <= latch_sp;
        end
        4'b 1011 : begin
          // addb
          left_ctrl <= accb_left;
          right_ctrl <= md_right;
          alu_ctrl <= alu_add8;
          cc_ctrl <= load_cc;
          acca_ctrl <= latch_acca;
          accb_ctrl <= load_accb;
          ix_ctrl <= latch_ix;
          sp_ctrl <= latch_sp;
        end
        4'b 1100 : begin
          // ldd
          left_ctrl <= accd_left;
          right_ctrl <= md_right;
          alu_ctrl <= alu_ld16;
          cc_ctrl <= load_cc;
          acca_ctrl <= load_hi_acca;
          accb_ctrl <= load_accb;
          ix_ctrl <= latch_ix;
          sp_ctrl <= latch_sp;
        end
        4'b 1101 : begin
          // std
          left_ctrl <= accd_left;
          right_ctrl <= md_right;
          alu_ctrl <= alu_st16;
          cc_ctrl <= load_cc;
          acca_ctrl <= latch_acca;
          accb_ctrl <= latch_accb;
          ix_ctrl <= latch_ix;
          sp_ctrl <= latch_sp;
        end
        4'b 1110 : begin
          // ldx
          left_ctrl <= ix_left;
          right_ctrl <= md_right;
          alu_ctrl <= alu_ld16;
          cc_ctrl <= load_cc;
          acca_ctrl <= latch_acca;
          accb_ctrl <= latch_accb;
          ix_ctrl <= load_ix;
          sp_ctrl <= latch_sp;
        end
        4'b 1111 : begin
          // stx
          left_ctrl <= ix_left;
          right_ctrl <= md_right;
          alu_ctrl <= alu_st16;
          cc_ctrl <= load_cc;
          acca_ctrl <= latch_acca;
          accb_ctrl <= latch_accb;
          ix_ctrl <= latch_ix;
          sp_ctrl <= latch_sp;
        end
        default : begin
          left_ctrl <= accb_left;
          right_ctrl <= md_right;
          alu_ctrl <= alu_nop;
```

```
          cc_ctrl <= latch_cc;
          acca_ctrl <= latch_acca;
          accb_ctrl <= latch_accb;
          ix_ctrl <= latch_ix;
          sp_ctrl <= latch_sp;
        end
        endcase
      end
      default : begin
        left_ctrl <= accd_left;
        right_ctrl <= md_right;
        alu_ctrl <= alu_nop;
        cc_ctrl <= latch_cc;
        acca_ctrl <= latch_acca;
        accb_ctrl <= latch_accb;
        ix_ctrl <= latch_ix;
        sp_ctrl <= latch_sp;
      end
      endcase
      md_ctrl <= latch_md;
      // fetch the op code
      op_ctrl <= fetch_op;
      ea_ctrl <= reset_ea;
      addr_ctrl <= fetch_ad;
      dout_ctrl <= md_lo_dout;
      iv_ctrl <= latch_iv;
      if(halt == 1'b1) begin
        pc_ctrl <= latch_pc;
        nmi_ctrl <= latch_nmi;
        next_state <= halt_state;
        // service non maskable interrupts
      end
      else if((nmi_req == 1'b1) && (nmi_ack == 1'b0)) begin
        pc_ctrl <= latch_pc;
        nmi_ctrl <= set_nmi;
        next_state <= int_pcl_state;
        // service maskable interrupts
      end
      else begin
        //
        // nmi request is not cleared until nmi input goes low
        //
        if((nmi_req == 1'b0) && (nmi_ack == 1'b1)) begin
          nmi_ctrl <= reset_nmi;
        end
        else begin
          nmi_ctrl <= latch_nmi;
        end
        //
        // IRQ is level sensitive
        //
        if(((irq == 1'b1) || (irq_icf == 1'b1) || (irq_ocf == 1'b1) || (irq_tof == 1'b1) || (irq_sci == 1'b1)) &
& (cc[IBIT]  == 1'b0)) begin
          pc_ctrl <= latch_pc;
          next_state <= int_pcl_state;
        end
        else begin
          // Advance the PC to fetch next instruction byte
          pc_ctrl <= inc_pc;
          next_state <= decode_state;
        end
      end
      //
      // Here to decode instruction
      // and fetch next byte of intruction
      // whether it be necessary or not
      //
    end
    decode_state : begin
      // fetch first byte of address or immediate data
      ea_ctrl <= fetch_first_ea;
      addr_ctrl <= fetch_ad;
      dout_ctrl <= md_lo_dout;
      op_ctrl <= latch_op;
      nmi_ctrl <= latch_nmi;
      iv_ctrl <= latch_iv;
      case(op_code[7:4] )
      4'b 0000 : begin
        md_ctrl <= fetch_first_md;
        sp_ctrl <= latch_sp;
        pc_ctrl <= latch_pc;
        case(op_code[3:0] )
        4'b 0001 : begin
          // nop
          left_ctrl <= accd_left;
          right_ctrl <= zero_right;
          alu_ctrl <= alu_nop;
          cc_ctrl <= latch_cc;
          acca_ctrl <= latch_acca;
          accb_ctrl <= latch_accb;
          ix_ctrl <= latch_ix;
```

```
        end
        4'b 0100 : begin
          // lsrd
          left_ctrl <= accd_left;
          right_ctrl <= zero_right;
          alu_ctrl <= alu_lsr16;
          cc_ctrl <= load_cc;
          acca_ctrl <= load_hi_acca;
          accb_ctrl <= load_accb;
          ix_ctrl <= latch_ix;
        end
        4'b 0101 : begin
          // lsld
          left_ctrl <= accd_left;
          right_ctrl <= zero_right;
          alu_ctrl <= alu_lsl16;
          cc_ctrl <= load_cc;
          acca_ctrl <= load_hi_acca;
          accb_ctrl <= load_accb;
          ix_ctrl <= latch_ix;
        end
        4'b 0110 : begin
          // tap
          left_ctrl <= acca_left;
          right_ctrl <= zero_right;
          alu_ctrl <= alu_tap;
          cc_ctrl <= load_cc;
          acca_ctrl <= latch_acca;
          accb_ctrl <= latch_accb;
          ix_ctrl <= latch_ix;
        end
        4'b 0111 : begin
          // tpa
          left_ctrl <= acca_left;
          right_ctrl <= zero_right;
          alu_ctrl <= alu_tpa;
          cc_ctrl <= latch_cc;
          acca_ctrl <= load_acca;
          accb_ctrl <= latch_accb;
          ix_ctrl <= latch_ix;
        end
        4'b 1000 : begin
          // inx
          left_ctrl <= ix_left;
          right_ctrl <= plus_one_right;
          alu_ctrl <= alu_inx;
          cc_ctrl <= load_cc;
          acca_ctrl <= latch_acca;
          accb_ctrl <= latch_accb;
          ix_ctrl <= load_ix;
        end
        4'b 1001 : begin
          // dex
          left_ctrl <= ix_left;
          right_ctrl <= plus_one_right;
          alu_ctrl <= alu_dex;
          cc_ctrl <= load_cc;
          acca_ctrl <= latch_acca;
          accb_ctrl <= latch_accb;
          ix_ctrl <= load_ix;
        end
        4'b 1010 : begin
          // clv
          left_ctrl <= acca_left;
          right_ctrl <= zero_right;
          alu_ctrl <= alu_clv;
          cc_ctrl <= load_cc;
          acca_ctrl <= latch_acca;
          accb_ctrl <= latch_accb;
          ix_ctrl <= latch_ix;
        end
        4'b 1011 : begin
          // sev
          left_ctrl <= acca_left;
          right_ctrl <= zero_right;
          alu_ctrl <= alu_sev;
          cc_ctrl <= load_cc;
          acca_ctrl <= latch_acca;
          accb_ctrl <= latch_accb;
          ix_ctrl <= latch_ix;
        end
        4'b 1100 : begin
          // clc
          left_ctrl <= acca_left;
          right_ctrl <= zero_right;
          alu_ctrl <= alu_clc;
          cc_ctrl <= load_cc;
          acca_ctrl <= latch_acca;
          accb_ctrl <= latch_accb;
          ix_ctrl <= latch_ix;
        end
```

```
        4'b 1101 : begin
          // sec
          left_ctrl <= acca_left;
          right_ctrl <= zero_right;
          alu_ctrl <= alu_sec;
          cc_ctrl <= load_cc;
          acca_ctrl <= latch_acca;
          accb_ctrl <= latch_accb;
          ix_ctrl <= latch_ix;
        end
        4'b 1110 : begin
          // cli
          left_ctrl <= acca_left;
          right_ctrl <= zero_right;
          alu_ctrl <= alu_cli;
          cc_ctrl <= load_cc;
          acca_ctrl <= latch_acca;
          accb_ctrl <= latch_accb;
          ix_ctrl <= latch_ix;
        end
        4'b 1111 : begin
          // sei
          left_ctrl <= acca_left;
          right_ctrl <= zero_right;
          alu_ctrl <= alu_sei;
          cc_ctrl <= load_cc;
          acca_ctrl <= latch_acca;
          accb_ctrl <= latch_accb;
          ix_ctrl <= latch_ix;
        end
        default : begin
          left_ctrl <= acca_left;
          right_ctrl <= zero_right;
          alu_ctrl <= alu_nop;
          cc_ctrl <= latch_cc;
          acca_ctrl <= latch_acca;
          accb_ctrl <= latch_accb;
          ix_ctrl <= latch_ix;
        end
        endcase
        next_state <= fetch_state;
        // acca / accb inherent instructions
      end
      4'b 0001 : begin
        md_ctrl <= fetch_first_md;
        ix_ctrl <= latch_ix;
        sp_ctrl <= latch_sp;
        pc_ctrl <= latch_pc;
        left_ctrl <= acca_left;
        right_ctrl <= accb_right;
        case(op_code[3:0] )
        4'b 0000 : begin
          // sba
          alu_ctrl <= alu_sub8;
          cc_ctrl <= load_cc;
          acca_ctrl <= load_acca;
          accb_ctrl <= latch_accb;
        end
        4'b 0001 : begin
          // cba
          alu_ctrl <= alu_sub8;
          cc_ctrl <= load_cc;
          acca_ctrl <= latch_acca;
          accb_ctrl <= latch_accb;
        end
        4'b 0110 : begin
          // tab
          alu_ctrl <= alu_st8;
          cc_ctrl <= load_cc;
          acca_ctrl <= latch_acca;
          accb_ctrl <= load_accb;
        end
        4'b 0111 : begin
          // tba
          alu_ctrl <= alu_ld8;
          cc_ctrl <= load_cc;
          acca_ctrl <= load_acca;
          accb_ctrl <= latch_accb;
        end
        4'b 1001 : begin
          // daa
          alu_ctrl <= alu_daa;
          cc_ctrl <= load_cc;
          acca_ctrl <= load_acca;
          accb_ctrl <= latch_accb;
        end
        4'b 1011 : begin
          // aba
          alu_ctrl <= alu_add8;
          cc_ctrl <= load_cc;
          acca_ctrl <= load_acca;
```

```
          accb_ctrl <= latch_accb;
        end
        default : begin
          alu_ctrl <= alu_nop;
          cc_ctrl <= latch_cc;
          acca_ctrl <= latch_acca;
          accb_ctrl <= latch_accb;
        end
        endcase
        next_state <= fetch_state;
      end
      4'b 0010 : begin
        // branch conditional
        md_ctrl <= fetch_first_md;
        acca_ctrl <= latch_acca;
        accb_ctrl <= latch_accb;
        ix_ctrl <= latch_ix;
        sp_ctrl <= latch_sp;
        left_ctrl <= acca_left;
        right_ctrl <= zero_right;
        alu_ctrl <= alu_nop;
        cc_ctrl <= latch_cc;
        // increment the pc
        pc_ctrl <= inc_pc;
        case(op_code[3:0] )
        4'b 0000 : begin
          // bra
          next_state <= branch_state;
        end
        4'b 0001 : begin
          // brn
          next_state <= fetch_state;
        end
        4'b 0010 : begin
          // bhi
          if(((cc[CBIT]  | cc[ZBIT] )) == 1'b0) begin
            next_state <= branch_state;
          end
          else begin
            next_state <= fetch_state;
          end
        end
        4'b 0011 : begin
          // bls
          if(((cc[CBIT]  | cc[ZBIT] )) == 1'b1) begin
            next_state <= branch_state;
          end
          else begin
            next_state <= fetch_state;
          end
        end
        4'b 0100 : begin
          // bcc/bhs
          if(cc[CBIT]  == 1'b0) begin
            next_state <= branch_state;
          end
          else begin
            next_state <= fetch_state;
          end
        end
        4'b 0101 : begin
          // bcs/blo
          if(cc[CBIT]  == 1'b1) begin
            next_state <= branch_state;
          end
          else begin
            next_state <= fetch_state;
          end
        end
        4'b 0110 : begin
          // bne
          if(cc[ZBIT]  == 1'b0) begin
            next_state <= branch_state;
          end
          else begin
            next_state <= fetch_state;
          end
        end
        4'b 0111 : begin
          // beq
          if(cc[ZBIT]  == 1'b1) begin
            next_state <= branch_state;
          end
          else begin
            next_state <= fetch_state;
          end
        end
        4'b 1000 : begin
          // bvc
          if(cc[VBIT]  == 1'b0) begin
            next_state <= branch_state;
```

```
          end
          else begin
            next_state <= fetch_state;
          end
        end
        4'b 1001 : begin
          // bvs
          if(cc[VBIT]  == 1'b1) begin
            next_state <= branch_state;
          end
          else begin
            next_state <= fetch_state;
          end
        end
        4'b 1010 : begin
          // bpl
          if(cc[NBIT]  == 1'b0) begin
            next_state <= branch_state;
          end
          else begin
            next_state <= fetch_state;
          end
        end
        4'b 1011 : begin
          // bmi
          if(cc[NBIT]  == 1'b1) begin
            next_state <= branch_state;
          end
          else begin
            next_state <= fetch_state;
          end
        end
        4'b 1100 : begin
          // bge
          if(((cc[NBIT]  ^ cc[VBIT] )) == 1'b0) begin
            next_state <= branch_state;
          end
          else begin
            next_state <= fetch_state;
          end
        end
        4'b 1101 : begin
          // blt
          if(((cc[NBIT]  ^ cc[VBIT] )) == 1'b1) begin
            next_state <= branch_state;
          end
          else begin
            next_state <= fetch_state;
          end
        end
        4'b 1110 : begin
          // bgt
          if(((cc[ZBIT]  | ((cc[NBIT]  ^ cc[VBIT] )))) == 1'b0) begin
            next_state <= branch_state;
          end
          else begin
            next_state <= fetch_state;
          end
        end
        4'b 1111 : begin
          // ble
          if(((cc[ZBIT]  | ((cc[NBIT]  ^ cc[VBIT] )))) == 1'b1) begin
            next_state <= branch_state;
          end
          else begin
            next_state <= fetch_state;
          end
        end
        default : begin
          next_state <= fetch_state;
        end
        endcase
        //
        // Single byte stack operators
        // Do not advance PC
        //
      end
      4'b 0011 : begin
        md_ctrl <= fetch_first_md;
        acca_ctrl <= latch_acca;
        accb_ctrl <= latch_accb;
        pc_ctrl <= latch_pc;
        case(op_code[3:0] )
        4'b 0000 : begin
          // tsx
          left_ctrl <= sp_left;
          right_ctrl <= plus_one_right;
          alu_ctrl <= alu_add16;
          cc_ctrl <= latch_cc;
          ix_ctrl <= load_ix;
          sp_ctrl <= latch_sp;
```

```
      next_state <= fetch_state;
    end
    4'b 0001 : begin
      // ins
      left_ctrl <= sp_left;
      right_ctrl <= plus_one_right;
      alu_ctrl <= alu_add16;
      cc_ctrl <= latch_cc;
      ix_ctrl <= latch_ix;
      sp_ctrl <= load_sp;
      next_state <= fetch_state;
    end
    4'b 0010 : begin
      // pula
      left_ctrl <= sp_left;
      right_ctrl <= plus_one_right;
      alu_ctrl <= alu_add16;
      cc_ctrl <= latch_cc;
      ix_ctrl <= latch_ix;
      sp_ctrl <= load_sp;
      next_state <= pula_state;
    end
    4'b 0011 : begin
      // pulb
      left_ctrl <= sp_left;
      right_ctrl <= plus_one_right;
      alu_ctrl <= alu_add16;
      cc_ctrl <= latch_cc;
      ix_ctrl <= latch_ix;
      sp_ctrl <= load_sp;
      next_state <= pulb_state;
    end
    4'b 0100 : begin
      // des
      // decrement sp
      left_ctrl <= sp_left;
      right_ctrl <= plus_one_right;
      alu_ctrl <= alu_sub16;
      cc_ctrl <= latch_cc;
      ix_ctrl <= latch_ix;
      sp_ctrl <= load_sp;
      next_state <= fetch_state;
    end
    4'b 0101 : begin
      // txs
      left_ctrl <= ix_left;
      right_ctrl <= plus_one_right;
      alu_ctrl <= alu_sub16;
      cc_ctrl <= latch_cc;
      ix_ctrl <= latch_ix;
      sp_ctrl <= load_sp;
      next_state <= fetch_state;
    end
    4'b 0110 : begin
      // psha
      left_ctrl <= sp_left;
      right_ctrl <= zero_right;
      alu_ctrl <= alu_nop;
      cc_ctrl <= latch_cc;
      ix_ctrl <= latch_ix;
      sp_ctrl <= latch_sp;
      next_state <= psha_state;
    end
    4'b 0111 : begin
      // pshb
      left_ctrl <= sp_left;
      right_ctrl <= zero_right;
      alu_ctrl <= alu_nop;
      cc_ctrl <= latch_cc;
      ix_ctrl <= latch_ix;
      sp_ctrl <= latch_sp;
      next_state <= pshb_state;
    end
    4'b 1000 : begin
      // pulx
      left_ctrl <= sp_left;
      right_ctrl <= plus_one_right;
      alu_ctrl <= alu_add16;
      cc_ctrl <= latch_cc;
      ix_ctrl <= latch_ix;
      sp_ctrl <= load_sp;
      next_state <= pulx_hi_state;
    end
    4'b 1001 : begin
      // rts
      left_ctrl <= sp_left;
      right_ctrl <= plus_one_right;
      alu_ctrl <= alu_add16;
      cc_ctrl <= latch_cc;
      ix_ctrl <= latch_ix;
      sp_ctrl <= load_sp;
```

```
          next_state <= rts_hi_state;
        end
        4'b 1010 : begin
          // abx
          left_ctrl <= ix_left;
          right_ctrl <= accb_right;
          alu_ctrl <= alu_add16;
          cc_ctrl <= latch_cc;
          ix_ctrl <= load_ix;
          sp_ctrl <= latch_sp;
          next_state <= fetch_state;
        end
        4'b 1011 : begin
          // rti
          left_ctrl <= sp_left;
          right_ctrl <= plus_one_right;
          alu_ctrl <= alu_add16;
          cc_ctrl <= latch_cc;
          ix_ctrl <= latch_ix;
          sp_ctrl <= load_sp;
          next_state <= rti_cc_state;
        end
        4'b 1100 : begin
          // pshx
          left_ctrl <= sp_left;
          right_ctrl <= zero_right;
          alu_ctrl <= alu_nop;
          cc_ctrl <= latch_cc;
          ix_ctrl <= latch_ix;
          sp_ctrl <= latch_sp;
          next_state <= pshx_lo_state;
        end
        4'b 1101 : begin
          // mul
          left_ctrl <= acca_left;
          right_ctrl <= accb_right;
          alu_ctrl <= alu_add16;
          cc_ctrl <= latch_cc;
          ix_ctrl <= latch_ix;
          sp_ctrl <= latch_sp;
          next_state <= mul_state;
        end
        4'b 1110 : begin
          // wai
          left_ctrl <= sp_left;
          right_ctrl <= zero_right;
          alu_ctrl <= alu_nop;
          cc_ctrl <= latch_cc;
          ix_ctrl <= latch_ix;
          sp_ctrl <= latch_sp;
          next_state <= int_pcl_state;
        end
        4'b 1111 : begin
          // swi
          left_ctrl <= sp_left;
          right_ctrl <= zero_right;
          alu_ctrl <= alu_nop;
          cc_ctrl <= latch_cc;
          ix_ctrl <= latch_ix;
          sp_ctrl <= latch_sp;
          next_state <= int_pcl_state;
        end
        default : begin
          left_ctrl <= sp_left;
          right_ctrl <= zero_right;
          alu_ctrl <= alu_nop;
          cc_ctrl <= latch_cc;
          ix_ctrl <= latch_ix;
          sp_ctrl <= latch_sp;
          next_state <= fetch_state;
        end
        endcase
        //
        // Accumulator A Single operand
        // source = Acc A dest = Acc A
        // Do not advance PC
        //
      end
      4'b 0100 : begin
        // acca single op
        md_ctrl <= fetch_first_md;
        accb_ctrl <= latch_accb;
        pc_ctrl <= latch_pc;
        ix_ctrl <= latch_ix;
        sp_ctrl <= latch_sp;
        left_ctrl <= acca_left;
        case(op_code[3:0] )
        4'b 0000 : begin
          // neg
          right_ctrl <= zero_right;
          alu_ctrl <= alu_neg;
```

```
          acca_ctrl <= load_acca;
          cc_ctrl <= load_cc;
        end
        4'b 0011 : begin
          // com
          right_ctrl <= zero_right;
          alu_ctrl <= alu_com;
          acca_ctrl <= load_acca;
          cc_ctrl <= load_cc;
        end
        4'b 0100 : begin
          // lsr
          right_ctrl <= zero_right;
          alu_ctrl <= alu_lsr8;
          acca_ctrl <= load_acca;
          cc_ctrl <= load_cc;
        end
        4'b 0110 : begin
          // ror
          right_ctrl <= zero_right;
          alu_ctrl <= alu_ror8;
          acca_ctrl <= load_acca;
          cc_ctrl <= load_cc;
        end
        4'b 0111 : begin
          // asr
          right_ctrl <= zero_right;
          alu_ctrl <= alu_asr8;
          acca_ctrl <= load_acca;
          cc_ctrl <= load_cc;
        end
        4'b 1000 : begin
          // asl
          right_ctrl <= zero_right;
          alu_ctrl <= alu_asl8;
          acca_ctrl <= load_acca;
          cc_ctrl <= load_cc;
        end
        4'b 1001 : begin
          // rol
          right_ctrl <= zero_right;
          alu_ctrl <= alu_rol8;
          acca_ctrl <= load_acca;
          cc_ctrl <= load_cc;
        end
        4'b 1010 : begin
          // dec
          right_ctrl <= plus_one_right;
          alu_ctrl <= alu_dec;
          acca_ctrl <= load_acca;
          cc_ctrl <= load_cc;
        end
        4'b 1011 : begin
          // undefined
          right_ctrl <= zero_right;
          alu_ctrl <= alu_nop;
          acca_ctrl <= latch_acca;
          cc_ctrl <= latch_cc;
        end
        4'b 1100 : begin
          // inc
          right_ctrl <= plus_one_right;
          alu_ctrl <= alu_inc;
          acca_ctrl <= load_acca;
          cc_ctrl <= load_cc;
        end
        4'b 1101 : begin
          // tst
          right_ctrl <= zero_right;
          alu_ctrl <= alu_st8;
          acca_ctrl <= latch_acca;
          cc_ctrl <= load_cc;
        end
        4'b 1110 : begin
          // jmp
          right_ctrl <= zero_right;
          alu_ctrl <= alu_nop;
          acca_ctrl <= latch_acca;
          cc_ctrl <= latch_cc;
        end
        4'b 1111 : begin
          // clr
          right_ctrl <= zero_right;
          alu_ctrl <= alu_clr;
          acca_ctrl <= load_acca;
          cc_ctrl <= load_cc;
        end
        default : begin
          right_ctrl <= zero_right;
          alu_ctrl <= alu_nop;
          acca_ctrl <= latch_acca;
```

```
          cc_ctrl <= latch_cc;
        end
        endcase
        next_state <= fetch_state;
        //
        // single operand acc b
        // Do not advance PC
        //
      end
      4'b 0101 : begin
        md_ctrl <= fetch_first_md;
        acca_ctrl <= latch_acca;
        pc_ctrl <= latch_pc;
        ix_ctrl <= latch_ix;
        sp_ctrl <= latch_sp;
        left_ctrl <= accb_left;
        case(op_code[3:0] )
        4'b 0000 : begin
          // neg
          right_ctrl <= zero_right;
          alu_ctrl <= alu_neg;
          accb_ctrl <= load_accb;
          cc_ctrl <= load_cc;
        end
        4'b 0011 : begin
          // com
          right_ctrl <= zero_right;
          alu_ctrl <= alu_com;
          accb_ctrl <= load_accb;
          cc_ctrl <= load_cc;
        end
        4'b 0100 : begin
          // lsr
          right_ctrl <= zero_right;
          alu_ctrl <= alu_lsr8;
          accb_ctrl <= load_accb;
          cc_ctrl <= load_cc;
        end
        4'b 0110 : begin
          // ror
          right_ctrl <= zero_right;
          alu_ctrl <= alu_ror8;
          accb_ctrl <= load_accb;
          cc_ctrl <= load_cc;
        end
        4'b 0111 : begin
          // asr
          right_ctrl <= zero_right;
          alu_ctrl <= alu_asr8;
          accb_ctrl <= load_accb;
          cc_ctrl <= load_cc;
        end
        4'b 1000 : begin
          // asl
          right_ctrl <= zero_right;
          alu_ctrl <= alu_asl8;
          accb_ctrl <= load_accb;
          cc_ctrl <= load_cc;
        end
        4'b 1001 : begin
          // rol
          right_ctrl <= zero_right;
          alu_ctrl <= alu_rol8;
          accb_ctrl <= load_accb;
          cc_ctrl <= load_cc;
        end
        4'b 1010 : begin
          // dec
          right_ctrl <= plus_one_right;
          alu_ctrl <= alu_dec;
          accb_ctrl <= load_accb;
          cc_ctrl <= load_cc;
        end
        4'b 1011 : begin
          // undefined
          right_ctrl <= zero_right;
          alu_ctrl <= alu_nop;
          accb_ctrl <= latch_accb;
          cc_ctrl <= latch_cc;
        end
        4'b 1100 : begin
          // inc
          right_ctrl <= plus_one_right;
          alu_ctrl <= alu_inc;
          accb_ctrl <= load_accb;
          cc_ctrl <= load_cc;
        end
        4'b 1101 : begin
          // tst
          right_ctrl <= zero_right;
          alu_ctrl <= alu_st8;
```

```
          accb_ctrl <= latch_accb;
          cc_ctrl <= load_cc;
        end
        4'b 1110 : begin
          // jmp
          right_ctrl <= zero_right;
          alu_ctrl <= alu_nop;
          accb_ctrl <= latch_accb;
          cc_ctrl <= latch_cc;
        end
        4'b 1111 : begin
          // clr
          right_ctrl <= zero_right;
          alu_ctrl <= alu_clr;
          accb_ctrl <= load_accb;
          cc_ctrl <= load_cc;
        end
        default : begin
          right_ctrl <= zero_right;
          alu_ctrl <= alu_nop;
          accb_ctrl <= latch_accb;
          cc_ctrl <= latch_cc;
        end
        endcase
        next_state <= fetch_state;
        //
        // Single operand indexed
        // Two byte instruction so advance PC
        // EA should hold index offset
        //
      end
      4'b 0110 : begin
        // indexed single op
        md_ctrl <= fetch_first_md;
        acca_ctrl <= latch_acca;
        accb_ctrl <= latch_accb;
        ix_ctrl <= latch_ix;
        sp_ctrl <= latch_sp;
        // increment the pc
        left_ctrl <= acca_left;
        right_ctrl <= zero_right;
        alu_ctrl <= alu_nop;
        cc_ctrl <= latch_cc;
        pc_ctrl <= inc_pc;
        next_state <= indexed_state;
        //
        // Single operand extended addressing
        // three byte instruction so advance the PC
        // Low order EA holds high order address
        //
      end
      4'b 0111 : begin
        // extended single op
        md_ctrl <= fetch_first_md;
        acca_ctrl <= latch_acca;
        accb_ctrl <= latch_accb;
        ix_ctrl <= latch_ix;
        sp_ctrl <= latch_sp;
        // increment the pc
        left_ctrl <= acca_left;
        right_ctrl <= zero_right;
        alu_ctrl <= alu_nop;
        cc_ctrl <= latch_cc;
        pc_ctrl <= inc_pc;
        next_state <= extended_state;
      end
      4'b 1000 : begin
        // acca immediate
        md_ctrl <= fetch_first_md;
        acca_ctrl <= latch_acca;
        accb_ctrl <= latch_accb;
        ix_ctrl <= latch_ix;
        sp_ctrl <= latch_sp;
        // increment the pc
        left_ctrl <= acca_left;
        right_ctrl <= zero_right;
        alu_ctrl <= alu_nop;
        cc_ctrl <= latch_cc;
        pc_ctrl <= inc_pc;
        // subdd #
        // cpx #
        // lds #
        // bsr
        case(op_code[3:0] )
        4'b 0011,4'b 1100,4'b 1110 : begin
          next_state <= immediate16_state;
        end
        4'b 1101 : begin
          next_state <= bsr_state;
        end
        default : begin
```

```
          next_state <= fetch_state;
        end
        endcase
        // acca direct
      end
      4'b 1001 : begin
        acca_ctrl <= latch_acca;
        accb_ctrl <= latch_accb;
        ix_ctrl <= latch_ix;
        sp_ctrl <= latch_sp;
        // increment the pc
        pc_ctrl <= inc_pc;
        // staa direct
        case(op_code[3:0] )
        4'b 0111 : begin
          left_ctrl <= acca_left;
          right_ctrl <= zero_right;
          alu_ctrl <= alu_st8;
          cc_ctrl <= latch_cc;
          md_ctrl <= load_md;
          next_state <= write8_state;
          // sts direct
        end
        4'b 1111 : begin
          left_ctrl <= sp_left;
          right_ctrl <= zero_right;
          alu_ctrl <= alu_st16;
          cc_ctrl <= latch_cc;
          md_ctrl <= load_md;
          next_state <= write16_state;
          // jsr direct
        end
        4'b 1101 : begin
          left_ctrl <= acca_left;
          right_ctrl <= zero_right;
          alu_ctrl <= alu_nop;
          cc_ctrl <= latch_cc;
          md_ctrl <= fetch_first_md;
          next_state <= jsr_state;
        end
        default : begin
          left_ctrl <= acca_left;
          right_ctrl <= zero_right;
          alu_ctrl <= alu_nop;
          cc_ctrl <= latch_cc;
          md_ctrl <= fetch_first_md;
          next_state <= read8_state;
        end
        endcase
        // acca indexed
      end
      4'b 1010 : begin
        md_ctrl <= fetch_first_md;
        acca_ctrl <= latch_acca;
        accb_ctrl <= latch_accb;
        ix_ctrl <= latch_ix;
        sp_ctrl <= latch_sp;
        // increment the pc
        left_ctrl <= acca_left;
        right_ctrl <= zero_right;
        alu_ctrl <= alu_nop;
        cc_ctrl <= latch_cc;
        pc_ctrl <= inc_pc;
        next_state <= indexed_state;
        // acca extended
      end
      4'b 1011 : begin
        md_ctrl <= fetch_first_md;
        acca_ctrl <= latch_acca;
        accb_ctrl <= latch_accb;
        ix_ctrl <= latch_ix;
        sp_ctrl <= latch_sp;
        // increment the pc
        left_ctrl <= acca_left;
        right_ctrl <= zero_right;
        alu_ctrl <= alu_nop;
        cc_ctrl <= latch_cc;
        pc_ctrl <= inc_pc;
        next_state <= extended_state;
        // accb immediate
      end
      4'b 1100 : begin
        md_ctrl <= fetch_first_md;
        acca_ctrl <= latch_acca;
        accb_ctrl <= latch_accb;
        ix_ctrl <= latch_ix;
        sp_ctrl <= latch_sp;
        // increment the pc
        left_ctrl <= acca_left;
        right_ctrl <= zero_right;
        alu_ctrl <= alu_nop;
```

```
        cc_ctrl <= latch_cc;
        pc_ctrl <= inc_pc;
        // addd #
        // ldd #
        // ldx #
        case(op_code[3:0] )
        4'b 0011,4'b 1100,4'b 1110 : begin
          next_state <= immediate16_state;
        end
        default : begin
          next_state <= fetch_state;
        end
        endcase
        // accb direct
      end
      4'b 1101 : begin
        acca_ctrl <= latch_acca;
        accb_ctrl <= latch_accb;
        ix_ctrl <= latch_ix;
        sp_ctrl <= latch_sp;
        // increment the pc
        pc_ctrl <= inc_pc;
        // stab direct
        case(op_code[3:0] )
        4'b 0111 : begin
          left_ctrl <= accb_left;
          right_ctrl <= zero_right;
          alu_ctrl <= alu_st8;
          cc_ctrl <= latch_cc;
          md_ctrl <= load_md;
          next_state <= write8_state;
          // std direct
        end
        4'b 1101 : begin
          left_ctrl <= accd_left;
          right_ctrl <= zero_right;
          alu_ctrl <= alu_st16;
          cc_ctrl <= latch_cc;
          md_ctrl <= load_md;
          next_state <= write16_state;
          // stx direct
        end
        4'b 1111 : begin
          left_ctrl <= ix_left;
          right_ctrl <= zero_right;
          alu_ctrl <= alu_st16;
          cc_ctrl <= latch_cc;
          md_ctrl <= load_md;
          next_state <= write16_state;
        end
        default : begin
          left_ctrl <= acca_left;
          right_ctrl <= zero_right;
          alu_ctrl <= alu_nop;
          cc_ctrl <= latch_cc;
          md_ctrl <= fetch_first_md;
          next_state <= read8_state;
        end
        endcase
        // accb indexed
      end
      4'b 1110 : begin
        md_ctrl <= fetch_first_md;
        acca_ctrl <= latch_acca;
        accb_ctrl <= latch_accb;
        ix_ctrl <= latch_ix;
        sp_ctrl <= latch_sp;
        // increment the pc
        left_ctrl <= acca_left;
        right_ctrl <= zero_right;
        alu_ctrl <= alu_nop;
        cc_ctrl <= latch_cc;
        pc_ctrl <= inc_pc;
        next_state <= indexed_state;
        // accb extended
      end
      4'b 1111 : begin
        md_ctrl <= fetch_first_md;
        acca_ctrl <= latch_acca;
        accb_ctrl <= latch_accb;
        ix_ctrl <= latch_ix;
        sp_ctrl <= latch_sp;
        // increment the pc
        left_ctrl <= acca_left;
        right_ctrl <= zero_right;
        alu_ctrl <= alu_nop;
        cc_ctrl <= latch_cc;
        pc_ctrl <= inc_pc;
        next_state <= extended_state;
      end
      default : begin
```

```
        md_ctrl <= fetch_first_md;
        acca_ctrl <= latch_acca;
        accb_ctrl <= latch_accb;
        ix_ctrl <= latch_ix;
        sp_ctrl <= latch_sp;
        // idle the pc
        left_ctrl <= acca_left;
        right_ctrl <= zero_right;
        alu_ctrl <= alu_nop;
        cc_ctrl <= latch_cc;
        pc_ctrl <= latch_pc;
        next_state <= fetch_state;
      end
      endcase
    end
    immediate16_state : begin
      acca_ctrl <= latch_acca;
      accb_ctrl <= latch_accb;
      ix_ctrl <= latch_ix;
      sp_ctrl <= latch_sp;
      op_ctrl <= latch_op;
      iv_ctrl <= latch_iv;
      nmi_ctrl <= latch_nmi;
      ea_ctrl <= latch_ea;
      // increment pc
      left_ctrl <= acca_left;
      right_ctrl <= zero_right;
      alu_ctrl <= alu_nop;
      cc_ctrl <= latch_cc;
      pc_ctrl <= inc_pc;
      // fetch next immediate byte
      md_ctrl <= fetch_next_md;
      addr_ctrl <= fetch_ad;
      dout_ctrl <= md_lo_dout;
      next_state <= fetch_state;
      //
      // ea holds 8 bit index offet
      // calculate the effective memory address
      // using the alu
      //
    end
    indexed_state : begin
      acca_ctrl <= latch_acca;
      accb_ctrl <= latch_accb;
      ix_ctrl <= latch_ix;
      sp_ctrl <= latch_sp;
      pc_ctrl <= latch_pc;
      iv_ctrl <= latch_iv;
      op_ctrl <= latch_op;
      nmi_ctrl <= latch_nmi;
      // calculate effective address from index reg
      // index offest is not sign extended
      ea_ctrl <= add_ix_ea;
      // idle the bus
      addr_ctrl <= idle_ad;
      dout_ctrl <= md_lo_dout;
      // work out next state
      // single op indexed
      case(op_code[7:4] )
      4'b 0110 : begin
        md_ctrl <= latch_md;
        left_ctrl <= acca_left;
        right_ctrl <= zero_right;
        alu_ctrl <= alu_nop;
        cc_ctrl <= latch_cc;
        // undefined
        // jmp
        // acca indexed
        case(op_code[3:0] )
        4'b 1011 : begin
          next_state <= fetch_state;
        end
        4'b 1110 : begin
          next_state <= jmp_state;
        end
        default : begin
          next_state <= read8_state;
        end
        endcase
        // staa
        // jsr
        // sts
      end
      4'b 1010 : begin
        case(op_code[3:0] )
        4'b 0111 : begin
          left_ctrl <= acca_left;
          right_ctrl <= zero_right;
          alu_ctrl <= alu_st8;
          cc_ctrl <= latch_cc;
          md_ctrl <= load_md;
```

```
          next_state <= write8_state;
        end
        4'b 1101 : begin
          left_ctrl <= acca_left;
          right_ctrl <= zero_right;
          alu_ctrl <= alu_nop;
          cc_ctrl <= latch_cc;
          md_ctrl <= latch_md;
          next_state <= jsr_state;
        end
        4'b 1111 : begin
          left_ctrl <= sp_left;
          right_ctrl <= zero_right;
          alu_ctrl <= alu_st16;
          cc_ctrl <= latch_cc;
          md_ctrl <= load_md;
          next_state <= write16_state;
        end
        default : begin
          left_ctrl <= acca_left;
          right_ctrl <= zero_right;
          alu_ctrl <= alu_nop;
          cc_ctrl <= latch_cc;
          md_ctrl <= latch_md;
          next_state <= read8_state;
        end
        endcase
        // accb indexed
        // stab direct
        // std direct
        // stx direct
      end
      4'b 1110 : begin
        case(op_code[3:0] )
        4'b 0111 : begin
          left_ctrl <= accb_left;
          right_ctrl <= zero_right;
          alu_ctrl <= alu_st8;
          cc_ctrl <= latch_cc;
          md_ctrl <= load_md;
          next_state <= write8_state;
        end
        4'b 1101 : begin
          left_ctrl <= accd_left;
          right_ctrl <= zero_right;
          alu_ctrl <= alu_st16;
          cc_ctrl <= latch_cc;
          md_ctrl <= load_md;
          next_state <= write16_state;
        end
        4'b 1111 : begin
          left_ctrl <= ix_left;
          right_ctrl <= zero_right;
          alu_ctrl <= alu_st16;
          cc_ctrl <= latch_cc;
          md_ctrl <= load_md;
          next_state <= write16_state;
        end
        default : begin
          left_ctrl <= acca_left;
          right_ctrl <= zero_right;
          alu_ctrl <= alu_nop;
          cc_ctrl <= latch_cc;
          md_ctrl <= latch_md;
          next_state <= read8_state;
        end
        endcase
      end
      default : begin
        md_ctrl <= latch_md;
        left_ctrl <= acca_left;
        right_ctrl <= zero_right;
        alu_ctrl <= alu_nop;
        cc_ctrl <= latch_cc;
        next_state <= fetch_state;
      end
      endcase
      //
      // ea holds the low byte of the absolute address
      // Move ea low byte into ea high byte
      // load new ea low byte to for absolute 16 bit address
      // advance the program counter
      //
    end
    extended_state : begin
      // fetch ea low byte
      acca_ctrl <= latch_acca;
      accb_ctrl <= latch_accb;
      ix_ctrl <= latch_ix;
      sp_ctrl <= latch_sp;
      iv_ctrl <= latch_iv;
```

```
      op_ctrl <= latch_op;
      nmi_ctrl <= latch_nmi;
      // increment pc
      pc_ctrl <= inc_pc;
      // fetch next effective address bytes
      ea_ctrl <= fetch_next_ea;
      addr_ctrl <= fetch_ad;
      dout_ctrl <= md_lo_dout;
      // work out the next state
      // single op extended
      // undefined
      // jmp
      case(op_code[7:4] )
      4'b 0111 : begin
        md_ctrl <= latch_md;
        left_ctrl <= acca_left;
        right_ctrl <= zero_right;
        alu_ctrl <= alu_nop;
        cc_ctrl <= latch_cc;
        case(op_code[3:0] )
        4'b 1011 : begin
          next_state <= fetch_state;
        end
        4'b 1110 : begin
          next_state <= jmp_state;
        end
        default : begin
          next_state <= read8_state;
        end
        endcase
        // acca extended
        // staa
        // jsr
        // sts
      end
      4'b 1011 : begin
        case(op_code[3:0] )
        4'b 0111 : begin
          left_ctrl <= acca_left;
          right_ctrl <= zero_right;
          alu_ctrl <= alu_st8;
          cc_ctrl <= latch_cc;
          md_ctrl <= load_md;
          next_state <= write8_state;
        end
        4'b 1101 : begin
          left_ctrl <= acca_left;
          right_ctrl <= zero_right;
          alu_ctrl <= alu_nop;
          cc_ctrl <= latch_cc;
          md_ctrl <= latch_md;
          next_state <= jsr_state;
        end
        4'b 1111 : begin
          left_ctrl <= sp_left;
          right_ctrl <= zero_right;
          alu_ctrl <= alu_st16;
          cc_ctrl <= latch_cc;
          md_ctrl <= load_md;
          next_state <= write16_state;
        end
        default : begin
          left_ctrl <= acca_left;
          right_ctrl <= zero_right;
          alu_ctrl <= alu_nop;
          cc_ctrl <= latch_cc;
          md_ctrl <= latch_md;
          next_state <= read8_state;
        end
        endcase
        // accb extended
        // stab
        // std
        // stx
      end
      4'b 1111 : begin
        case(op_code[3:0] )
        4'b 0111 : begin
          left_ctrl <= accb_left;
          right_ctrl <= zero_right;
          alu_ctrl <= alu_st8;
          cc_ctrl <= latch_cc;
          md_ctrl <= load_md;
          next_state <= write8_state;
        end
        4'b 1101 : begin
          left_ctrl <= accd_left;
          right_ctrl <= zero_right;
          alu_ctrl <= alu_st16;
          cc_ctrl <= latch_cc;
          md_ctrl <= load_md;
```

```
          next_state <= write16_state;
        end
        4'b 1111 : begin
          left_ctrl <= ix_left;
          right_ctrl <= zero_right;
          alu_ctrl <= alu_st16;
          cc_ctrl <= latch_cc;
          md_ctrl <= load_md;
          next_state <= write16_state;
        end
        default : begin
          left_ctrl <= acca_left;
          right_ctrl <= zero_right;
          alu_ctrl <= alu_nop;
          cc_ctrl <= latch_cc;
          md_ctrl <= latch_md;
          next_state <= read8_state;
        end
        endcase
      end
      default : begin
        md_ctrl <= latch_md;
        left_ctrl <= acca_left;
        right_ctrl <= zero_right;
        alu_ctrl <= alu_nop;
        cc_ctrl <= latch_cc;
        next_state <= fetch_state;
      end
      endcase
      //
      // here if ea holds low byte (direct page)
      // can enter here from extended addressing
      // read memory location
      // note that reads may be 8 or 16 bits
      //
    end
    read8_state : begin
      // read data
      acca_ctrl <= latch_acca;
      accb_ctrl <= latch_accb;
      ix_ctrl <= latch_ix;
      sp_ctrl <= latch_sp;
      pc_ctrl <= latch_pc;
      iv_ctrl <= latch_iv;
      op_ctrl <= latch_op;
      nmi_ctrl <= latch_nmi;
      //
      addr_ctrl <= read_ad;
      dout_ctrl <= md_lo_dout;
      // single operand
      // acca
      case(op_code[7:4] )
      4'b 0110,4'b 0111 : begin
        left_ctrl <= acca_left;
        right_ctrl <= zero_right;
        alu_ctrl <= alu_nop;
        cc_ctrl <= latch_cc;
        md_ctrl <= fetch_first_md;
        ea_ctrl <= latch_ea;
        next_state <= execute_state;
        // subd
        // lds
        // cpx
      end
      4'b 1001,4'b 1010,4'b 1011 : begin
        case(op_code[3:0] )
        4'b 0011,4'b 1110,4'b 1100 : begin
          left_ctrl <= acca_left;
          right_ctrl <= zero_right;
          alu_ctrl <= alu_nop;
          cc_ctrl <= latch_cc;
          md_ctrl <= fetch_first_md;
          // increment the effective address in case of 16 bit load
          ea_ctrl <= inc_ea;
          next_state <= read16_state;
        end
        default : begin
          left_ctrl <= acca_left;
          right_ctrl <= zero_right;
          alu_ctrl <= alu_nop;
          cc_ctrl <= latch_cc;
          md_ctrl <= fetch_first_md;
          ea_ctrl <= latch_ea;
          next_state <= fetch_state;
        end
        endcase
        // accb
        // addd
        // ldd
        // ldx
      end
```

```
      4'b 1101,4'b 1110,4'b 1111 : begin
        case(op_code[3:0] )
        4'b 0011,4'b 1100,4'b 1110 : begin
          left_ctrl <= acca_left;
          right_ctrl <= zero_right;
          alu_ctrl <= alu_nop;
          cc_ctrl <= latch_cc;
          md_ctrl <= fetch_first_md;
          // increment the effective address in case of 16 bit load
          ea_ctrl <= inc_ea;
          next_state <= read16_state;
        end
        default : begin
          left_ctrl <= acca_left;
          right_ctrl <= zero_right;
          alu_ctrl <= alu_nop;
          cc_ctrl <= latch_cc;
          md_ctrl <= fetch_first_md;
          ea_ctrl <= latch_ea;
          next_state <= execute_state;
        end
        endcase
      end
      default : begin
        left_ctrl <= acca_left;
        right_ctrl <= zero_right;
        alu_ctrl <= alu_nop;
        cc_ctrl <= latch_cc;
        md_ctrl <= fetch_first_md;
        ea_ctrl <= latch_ea;
        next_state <= fetch_state;
      end
      endcase
      // read second data byte from ea
      // default
    end
    read16_state : begin
      acca_ctrl <= latch_acca;
      accb_ctrl <= latch_accb;
      ix_ctrl <= latch_ix;
      sp_ctrl <= latch_sp;
      pc_ctrl <= latch_pc;
      iv_ctrl <= latch_iv;
      op_ctrl <= latch_op;
      nmi_ctrl <= latch_nmi;
      left_ctrl <= acca_left;
      right_ctrl <= zero_right;
      alu_ctrl <= alu_nop;
      cc_ctrl <= latch_cc;
      // idle the effective address
      ea_ctrl <= latch_ea;
      // read the low byte of the 16 bit data
      md_ctrl <= fetch_next_md;
      addr_ctrl <= read_ad;
      dout_ctrl <= md_lo_dout;
      next_state <= fetch_state;
      //
      // 16 bit Write state
      // write high byte of ALU output.
      // EA hold address of memory to write to
      // Advance the effective address in ALU
      //
    end
    write16_state : begin
      // default
      acca_ctrl <= latch_acca;
      accb_ctrl <= latch_accb;
      ix_ctrl <= latch_ix;
      sp_ctrl <= latch_sp;
      pc_ctrl <= latch_pc;
      md_ctrl <= latch_md;
      iv_ctrl <= latch_iv;
      op_ctrl <= latch_op;
      nmi_ctrl <= latch_nmi;
      // increment the effective address
      left_ctrl <= acca_left;
      right_ctrl <= zero_right;
      alu_ctrl <= alu_nop;
      cc_ctrl <= latch_cc;
      ea_ctrl <= inc_ea;
      // write the ALU hi byte to ea
      addr_ctrl <= write_ad;
      dout_ctrl <= md_hi_dout;
      next_state <= write8_state;
      //
      // 8 bit write
      // Write low 8 bits of ALU output
      //
    end
    write8_state : begin
      // default registers
```

```
      acca_ctrl <= latch_acca;
      accb_ctrl <= latch_accb;
      ix_ctrl <= latch_ix;
      sp_ctrl <= latch_sp;
      pc_ctrl <= latch_pc;
      md_ctrl <= latch_md;
      iv_ctrl <= latch_iv;
      op_ctrl <= latch_op;
      nmi_ctrl <= latch_nmi;
      ea_ctrl <= latch_ea;
      // idle the ALU
      left_ctrl <= acca_left;
      right_ctrl <= zero_right;
      alu_ctrl <= alu_nop;
      cc_ctrl <= latch_cc;
      // write ALU low byte output
      addr_ctrl <= write_ad;
      dout_ctrl <= md_lo_dout;
      next_state <= fetch_state;
    end
    jmp_state : begin
      acca_ctrl <= latch_acca;
      accb_ctrl <= latch_accb;
      ix_ctrl <= latch_ix;
      sp_ctrl <= latch_sp;
      md_ctrl <= latch_md;
      iv_ctrl <= latch_iv;
      op_ctrl <= latch_op;
      nmi_ctrl <= latch_nmi;
      ea_ctrl <= latch_ea;
      // load PC with effective address
      left_ctrl <= acca_left;
      right_ctrl <= zero_right;
      alu_ctrl <= alu_nop;
      cc_ctrl <= latch_cc;
      pc_ctrl <= load_ea_pc;
      // idle the bus
      addr_ctrl <= idle_ad;
      dout_ctrl <= md_lo_dout;
      next_state <= fetch_state;
    end
    jsr_state : begin
      // JSR
      acca_ctrl <= latch_acca;
      accb_ctrl <= latch_accb;
      ix_ctrl <= latch_ix;
      sp_ctrl <= latch_sp;
      pc_ctrl <= latch_pc;
      md_ctrl <= latch_md;
      iv_ctrl <= latch_iv;
      op_ctrl <= latch_op;
      nmi_ctrl <= latch_nmi;
      ea_ctrl <= latch_ea;
      // decrement sp
      left_ctrl <= sp_left;
      right_ctrl <= plus_one_right;
      alu_ctrl <= alu_sub16;
      cc_ctrl <= latch_cc;
      sp_ctrl <= load_sp;
      // write pc low
      addr_ctrl <= push_ad;
      dout_ctrl <= pc_lo_dout;
      next_state <= jsr1_state;
    end
    jsr1_state : begin
      // JSR
      acca_ctrl <= latch_acca;
      accb_ctrl <= latch_accb;
      ix_ctrl <= latch_ix;
      pc_ctrl <= latch_pc;
      md_ctrl <= latch_md;
      iv_ctrl <= latch_iv;
      op_ctrl <= latch_op;
      nmi_ctrl <= latch_nmi;
      ea_ctrl <= latch_ea;
      // decrement sp
      left_ctrl <= sp_left;
      right_ctrl <= plus_one_right;
      alu_ctrl <= alu_sub16;
      cc_ctrl <= latch_cc;
      sp_ctrl <= load_sp;
      // write pc hi
      addr_ctrl <= push_ad;
      dout_ctrl <= pc_hi_dout;
      next_state <= jmp_state;
    end
    branch_state : begin
      // Bcc
      // default registers
      acca_ctrl <= latch_acca;
      accb_ctrl <= latch_accb;
```

```
      ix_ctrl <= latch_ix;
      sp_ctrl <= latch_sp;
      md_ctrl <= latch_md;
      iv_ctrl <= latch_iv;
      op_ctrl <= latch_op;
      nmi_ctrl <= latch_nmi;
      ea_ctrl <= latch_ea;
      // calculate signed branch
      left_ctrl <= acca_left;
      right_ctrl <= zero_right;
      alu_ctrl <= alu_nop;
      cc_ctrl <= latch_cc;
      pc_ctrl <= add_ea_pc;
      // idle the bus
      addr_ctrl <= idle_ad;
      dout_ctrl <= md_lo_dout;
      next_state <= fetch_state;
    end
    bsr_state : begin
      // BSR
      // default
      acca_ctrl <= latch_acca;
      accb_ctrl <= latch_accb;
      ix_ctrl <= latch_ix;
      pc_ctrl <= latch_pc;
      md_ctrl <= latch_md;
      iv_ctrl <= latch_iv;
      op_ctrl <= latch_op;
      nmi_ctrl <= latch_nmi;
      ea_ctrl <= latch_ea;
      // decrement sp
      left_ctrl <= sp_left;
      right_ctrl <= plus_one_right;
      alu_ctrl <= alu_sub16;
      cc_ctrl <= latch_cc;
      sp_ctrl <= load_sp;
      // write pc low
      addr_ctrl <= push_ad;
      dout_ctrl <= pc_lo_dout;
      next_state <= bsr1_state;
    end
    bsr1_state : begin
      // BSR
      // default registers
      acca_ctrl <= latch_acca;
      accb_ctrl <= latch_accb;
      ix_ctrl <= latch_ix;
      pc_ctrl <= latch_pc;
      md_ctrl <= latch_md;
      iv_ctrl <= latch_iv;
      op_ctrl <= latch_op;
      nmi_ctrl <= latch_nmi;
      ea_ctrl <= latch_ea;
      // decrement sp
      left_ctrl <= sp_left;
      right_ctrl <= plus_one_right;
      alu_ctrl <= alu_sub16;
      cc_ctrl <= latch_cc;
      sp_ctrl <= load_sp;
      // write pc hi
      addr_ctrl <= push_ad;
      dout_ctrl <= pc_hi_dout;
      next_state <= branch_state;
    end
    rts_hi_state : begin
      // RTS
      // default
      acca_ctrl <= latch_acca;
      accb_ctrl <= latch_accb;
      ix_ctrl <= latch_ix;
      pc_ctrl <= latch_pc;
      md_ctrl <= latch_md;
      iv_ctrl <= latch_iv;
      op_ctrl <= latch_op;
      nmi_ctrl <= latch_nmi;
      ea_ctrl <= latch_ea;
      // increment the sp
      left_ctrl <= sp_left;
      right_ctrl <= plus_one_right;
      alu_ctrl <= alu_add16;
      cc_ctrl <= latch_cc;
      sp_ctrl <= load_sp;
      // read pc hi
      pc_ctrl <= pull_hi_pc;
      addr_ctrl <= pull_ad;
      dout_ctrl <= pc_hi_dout;
      next_state <= rts_lo_state;
    end
    rts_lo_state : begin
      // RTS1
      // default
```

```
      acca_ctrl <= latch_acca;
      accb_ctrl <= latch_accb;
      ix_ctrl <= latch_ix;
      sp_ctrl <= latch_sp;
      md_ctrl <= latch_md;
      iv_ctrl <= latch_iv;
      op_ctrl <= latch_op;
      nmi_ctrl <= latch_nmi;
      ea_ctrl <= latch_ea;
      // idle the ALU
      left_ctrl <= acca_left;
      right_ctrl <= zero_right;
      alu_ctrl <= alu_nop;
      cc_ctrl <= latch_cc;
      // read pc low
      pc_ctrl <= pull_lo_pc;
      addr_ctrl <= pull_ad;
      dout_ctrl <= pc_lo_dout;
      next_state <= fetch_state;
    end
    mul_state : begin
      // default
      acca_ctrl <= latch_acca;
      accb_ctrl <= latch_accb;
      ix_ctrl <= latch_ix;
      sp_ctrl <= latch_sp;
      pc_ctrl <= latch_pc;
      iv_ctrl <= latch_iv;
      op_ctrl <= latch_op;
      nmi_ctrl <= latch_nmi;
      ea_ctrl <= latch_ea;
      // move acca to md
      left_ctrl <= acca_left;
      right_ctrl <= zero_right;
      alu_ctrl <= alu_st16;
      cc_ctrl <= latch_cc;
      md_ctrl <= load_md;
      // idle bus
      addr_ctrl <= idle_ad;
      dout_ctrl <= md_lo_dout;
      next_state <= mulea_state;
    end
    mulea_state : begin
      // default
      acca_ctrl <= latch_acca;
      accb_ctrl <= latch_accb;
      ix_ctrl <= latch_ix;
      sp_ctrl <= latch_sp;
      pc_ctrl <= latch_pc;
      iv_ctrl <= latch_iv;
      op_ctrl <= latch_op;
      nmi_ctrl <= latch_nmi;
      md_ctrl <= latch_md;
      // idle ALU
      left_ctrl <= acca_left;
      right_ctrl <= zero_right;
      alu_ctrl <= alu_nop;
      cc_ctrl <= latch_cc;
      // move accb to ea
      ea_ctrl <= load_accb_ea;
      // idle bus
      addr_ctrl <= idle_ad;
      dout_ctrl <= md_lo_dout;
      next_state <= muld_state;
    end
    muld_state : begin
      // default
      ix_ctrl <= latch_ix;
      sp_ctrl <= latch_sp;
      pc_ctrl <= latch_pc;
      iv_ctrl <= latch_iv;
      op_ctrl <= latch_op;
      nmi_ctrl <= latch_nmi;
      ea_ctrl <= latch_ea;
      md_ctrl <= latch_md;
      // clear accd
      left_ctrl <= acca_left;
      right_ctrl <= zero_right;
      alu_ctrl <= alu_ld8;
      cc_ctrl <= latch_cc;
      acca_ctrl <= load_hi_acca;
      accb_ctrl <= load_accb;
      // idle bus
      addr_ctrl <= idle_ad;
      dout_ctrl <= md_lo_dout;
      next_state <= mul0_state;
    end
    mul0_state : begin
      // default
      ix_ctrl <= latch_ix;
      sp_ctrl <= latch_sp;
```

```
      pc_ctrl <= latch_pc;
      iv_ctrl <= latch_iv;
      op_ctrl <= latch_op;
      nmi_ctrl <= latch_nmi;
      ea_ctrl <= latch_ea;
      // if bit 0 of ea set, add accd to md
      left_ctrl <= accd_left;
      right_ctrl <= md_right;
      alu_ctrl <= alu_add16;
      if(ea[0]  == 1'b1) begin
        cc_ctrl <= load_cc;
        acca_ctrl <= load_hi_acca;
        accb_ctrl <= load_accb;
      end
      else begin
        cc_ctrl <= latch_cc;
        acca_ctrl <= latch_acca;
        accb_ctrl <= latch_accb;
      end
      md_ctrl <= shiftl_md;
      // idle bus
      addr_ctrl <= idle_ad;
      dout_ctrl <= md_lo_dout;
      next_state <= mul1_state;
    end
    mul1_state : begin
      // default
      ix_ctrl <= latch_ix;
      sp_ctrl <= latch_sp;
      pc_ctrl <= latch_pc;
      iv_ctrl <= latch_iv;
      op_ctrl <= latch_op;
      nmi_ctrl <= latch_nmi;
      ea_ctrl <= latch_ea;
      // if bit 1 of ea set, add accd to md
      left_ctrl <= accd_left;
      right_ctrl <= md_right;
      alu_ctrl <= alu_add16;
      if(ea[1]  == 1'b1) begin
        cc_ctrl <= load_cc;
        acca_ctrl <= load_hi_acca;
        accb_ctrl <= load_accb;
      end
      else begin
        cc_ctrl <= latch_cc;
        acca_ctrl <= latch_acca;
        accb_ctrl <= latch_accb;
      end
      md_ctrl <= shiftl_md;
      // idle bus
      addr_ctrl <= idle_ad;
      dout_ctrl <= md_lo_dout;
      next_state <= mul2_state;
    end
    mul2_state : begin
      // default
      ix_ctrl <= latch_ix;
      sp_ctrl <= latch_sp;
      pc_ctrl <= latch_pc;
      iv_ctrl <= latch_iv;
      op_ctrl <= latch_op;
      nmi_ctrl <= latch_nmi;
      ea_ctrl <= latch_ea;
      // if bit 2 of ea set, add accd to md
      left_ctrl <= accd_left;
      right_ctrl <= md_right;
      alu_ctrl <= alu_add16;
      if(ea[2]  == 1'b1) begin
        cc_ctrl <= load_cc;
        acca_ctrl <= load_hi_acca;
        accb_ctrl <= load_accb;
      end
      else begin
        cc_ctrl <= latch_cc;
        acca_ctrl <= latch_acca;
        accb_ctrl <= latch_accb;
      end
      md_ctrl <= shiftl_md;
      // idle bus
      addr_ctrl <= idle_ad;
      dout_ctrl <= md_lo_dout;
      next_state <= mul3_state;
    end
    mul3_state : begin
      // default
      ix_ctrl <= latch_ix;
      sp_ctrl <= latch_sp;
      pc_ctrl <= latch_pc;
      iv_ctrl <= latch_iv;
      op_ctrl <= latch_op;
      nmi_ctrl <= latch_nmi;
```

```
      ea_ctrl <= latch_ea;
      // if bit 3 of ea set, add accd to md
      left_ctrl <= accd_left;
      right_ctrl <= md_right;
      alu_ctrl <= alu_add16;
      if(ea[3]  == 1'b1) begin
        cc_ctrl <= load_cc;
        acca_ctrl <= load_hi_acca;
        accb_ctrl <= load_accb;
      end
      else begin
        cc_ctrl <= latch_cc;
        acca_ctrl <= latch_acca;
        accb_ctrl <= latch_accb;
      end
      md_ctrl <= shiftl_md;
      // idle bus
      addr_ctrl <= idle_ad;
      dout_ctrl <= md_lo_dout;
      next_state <= mul4_state;
    end
    mul4_state : begin
      // default
      ix_ctrl <= latch_ix;
      sp_ctrl <= latch_sp;
      pc_ctrl <= latch_pc;
      iv_ctrl <= latch_iv;
      op_ctrl <= latch_op;
      nmi_ctrl <= latch_nmi;
      ea_ctrl <= latch_ea;
      // if bit 4 of ea set, add accd to md
      left_ctrl <= accd_left;
      right_ctrl <= md_right;
      alu_ctrl <= alu_add16;
      if(ea[4]  == 1'b1) begin
        cc_ctrl <= load_cc;
        acca_ctrl <= load_hi_acca;
        accb_ctrl <= load_accb;
      end
      else begin
        cc_ctrl <= latch_cc;
        acca_ctrl <= latch_acca;
        accb_ctrl <= latch_accb;
      end
      md_ctrl <= shiftl_md;
      // idle bus
      addr_ctrl <= idle_ad;
      dout_ctrl <= md_lo_dout;
      next_state <= mul5_state;
    end
    mul5_state : begin
      // default
      ix_ctrl <= latch_ix;
      sp_ctrl <= latch_sp;
      pc_ctrl <= latch_pc;
      iv_ctrl <= latch_iv;
      op_ctrl <= latch_op;
      nmi_ctrl <= latch_nmi;
      ea_ctrl <= latch_ea;
      // if bit 5 of ea set, add accd to md
      left_ctrl <= accd_left;
      right_ctrl <= md_right;
      alu_ctrl <= alu_add16;
      if(ea[5]  == 1'b1) begin
        cc_ctrl <= load_cc;
        acca_ctrl <= load_hi_acca;
        accb_ctrl <= load_accb;
      end
      else begin
        cc_ctrl <= latch_cc;
        acca_ctrl <= latch_acca;
        accb_ctrl <= latch_accb;
      end
      md_ctrl <= shiftl_md;
      // idle bus
      addr_ctrl <= idle_ad;
      dout_ctrl <= md_lo_dout;
      next_state <= mul6_state;
    end
    mul6_state : begin
      // default
      ix_ctrl <= latch_ix;
      sp_ctrl <= latch_sp;
      pc_ctrl <= latch_pc;
      iv_ctrl <= latch_iv;
      op_ctrl <= latch_op;
      nmi_ctrl <= latch_nmi;
      ea_ctrl <= latch_ea;
      // if bit 6 of ea set, add accd to md
      left_ctrl <= accd_left;
      right_ctrl <= md_right;
```

```
      alu_ctrl <= alu_add16;
      if(ea[6]  == 1'b1) begin
        cc_ctrl <= load_cc;
        acca_ctrl <= load_hi_acca;
        accb_ctrl <= load_accb;
      end
      else begin
        cc_ctrl <= latch_cc;
        acca_ctrl <= latch_acca;
        accb_ctrl <= latch_accb;
      end
      md_ctrl <= shiftl_md;
      // idle bus
      addr_ctrl <= idle_ad;
      dout_ctrl <= md_lo_dout;
      next_state <= mul7_state;
    end
    mul7_state : begin
      // default
      ix_ctrl <= latch_ix;
      sp_ctrl <= latch_sp;
      pc_ctrl <= latch_pc;
      iv_ctrl <= latch_iv;
      op_ctrl <= latch_op;
      nmi_ctrl <= latch_nmi;
      ea_ctrl <= latch_ea;
      // if bit 7 of ea set, add accd to md
      left_ctrl <= accd_left;
      right_ctrl <= md_right;
      alu_ctrl <= alu_add16;
      if(ea[7]  == 1'b1) begin
        cc_ctrl <= load_cc;
        acca_ctrl <= load_hi_acca;
        accb_ctrl <= load_accb;
      end
      else begin
        cc_ctrl <= latch_cc;
        acca_ctrl <= latch_acca;
        accb_ctrl <= latch_accb;
      end
      md_ctrl <= shiftl_md;
      // idle bus
      addr_ctrl <= idle_ad;
      dout_ctrl <= md_lo_dout;
      next_state <= fetch_state;
    end
    execute_state : begin
      // execute single operand instruction
      // default
      op_ctrl <= latch_op;
      nmi_ctrl <= latch_nmi;
      // indexed single op
      // extended single op
      case(op_code[7:4] )
      4'b 0110,4'b 0111 : begin
        acca_ctrl <= latch_acca;
        accb_ctrl <= latch_accb;
        ix_ctrl <= latch_ix;
        sp_ctrl <= latch_sp;
        pc_ctrl <= latch_pc;
        iv_ctrl <= latch_iv;
        ea_ctrl <= latch_ea;
        // idle the bus
        addr_ctrl <= idle_ad;
        dout_ctrl <= md_lo_dout;
        case(op_code[3:0] )
        4'b 0000 : begin
          // neg
          left_ctrl <= md_left;
          right_ctrl <= zero_right;
          alu_ctrl <= alu_neg;
          cc_ctrl <= load_cc;
          md_ctrl <= load_md;
          next_state <= write8_state;
        end
        4'b 0011 : begin
          // com
          left_ctrl <= md_left;
          right_ctrl <= zero_right;
          alu_ctrl <= alu_com;
          cc_ctrl <= load_cc;
          md_ctrl <= load_md;
          next_state <= write8_state;
        end
        4'b 0100 : begin
          // lsr
          left_ctrl <= md_left;
          right_ctrl <= zero_right;
          alu_ctrl <= alu_lsr8;
          cc_ctrl <= load_cc;
          md_ctrl <= load_md;
```

```
          next_state <= write8_state;
        end
        4'b 0110 : begin
          // ror
          left_ctrl <= md_left;
          right_ctrl <= zero_right;
          alu_ctrl <= alu_ror8;
          cc_ctrl <= load_cc;
          md_ctrl <= load_md;
          next_state <= write8_state;
        end
        4'b 0111 : begin
          // asr
          left_ctrl <= md_left;
          right_ctrl <= zero_right;
          alu_ctrl <= alu_asr8;
          cc_ctrl <= load_cc;
          md_ctrl <= load_md;
          next_state <= write8_state;
        end
        4'b 1000 : begin
          // asl
          left_ctrl <= md_left;
          right_ctrl <= zero_right;
          alu_ctrl <= alu_asl8;
          cc_ctrl <= load_cc;
          md_ctrl <= load_md;
          next_state <= write8_state;
        end
        4'b 1001 : begin
          // rol
          left_ctrl <= md_left;
          right_ctrl <= zero_right;
          alu_ctrl <= alu_rol8;
          cc_ctrl <= load_cc;
          md_ctrl <= load_md;
          next_state <= write8_state;
        end
        4'b 1010 : begin
          // dec
          left_ctrl <= md_left;
          right_ctrl <= plus_one_right;
          alu_ctrl <= alu_dec;
          cc_ctrl <= load_cc;
          md_ctrl <= load_md;
          next_state <= write8_state;
        end
        4'b 1011 : begin
          // undefined
          left_ctrl <= md_left;
          right_ctrl <= zero_right;
          alu_ctrl <= alu_nop;
          cc_ctrl <= latch_cc;
          md_ctrl <= latch_md;
          next_state <= fetch_state;
        end
        4'b 1100 : begin
          // inc
          left_ctrl <= md_left;
          right_ctrl <= plus_one_right;
          alu_ctrl <= alu_inc;
          cc_ctrl <= load_cc;
          md_ctrl <= load_md;
          next_state <= write8_state;
        end
        4'b 1101 : begin
          // tst
          left_ctrl <= md_left;
          right_ctrl <= zero_right;
          alu_ctrl <= alu_st8;
          cc_ctrl <= load_cc;
          md_ctrl <= latch_md;
          next_state <= fetch_state;
        end
        4'b 1110 : begin
          // jmp
          left_ctrl <= md_left;
          right_ctrl <= zero_right;
          alu_ctrl <= alu_nop;
          cc_ctrl <= latch_cc;
          md_ctrl <= latch_md;
          next_state <= fetch_state;
        end
        4'b 1111 : begin
          // clr
          left_ctrl <= md_left;
          right_ctrl <= zero_right;
          alu_ctrl <= alu_clr;
          cc_ctrl <= load_cc;
          md_ctrl <= load_md;
          next_state <= write8_state;
```

```
        end
        default : begin
          left_ctrl <= md_left;
          right_ctrl <= zero_right;
          alu_ctrl <= alu_nop;
          cc_ctrl <= latch_cc;
          md_ctrl <= latch_md;
          next_state <= fetch_state;
        end
        endcase
      end
      default : begin
        left_ctrl <= accd_left;
        right_ctrl <= md_right;
        alu_ctrl <= alu_nop;
        cc_ctrl <= latch_cc;
        acca_ctrl <= latch_acca;
        accb_ctrl <= latch_accb;
        ix_ctrl <= latch_ix;
        sp_ctrl <= latch_sp;
        pc_ctrl <= latch_pc;
        md_ctrl <= latch_md;
        iv_ctrl <= latch_iv;
        ea_ctrl <= latch_ea;
        // idle the bus
        addr_ctrl <= idle_ad;
        dout_ctrl <= md_lo_dout;
        next_state <= fetch_state;
      end
      endcase
    end
    psha_state : begin
      // default registers
      acca_ctrl <= latch_acca;
      accb_ctrl <= latch_accb;
      ix_ctrl <= latch_ix;
      pc_ctrl <= latch_pc;
      md_ctrl <= latch_md;
      iv_ctrl <= latch_iv;
      op_ctrl <= latch_op;
      nmi_ctrl <= latch_nmi;
      ea_ctrl <= latch_ea;
      // decrement sp
      left_ctrl <= sp_left;
      right_ctrl <= plus_one_right;
      alu_ctrl <= alu_sub16;
      cc_ctrl <= latch_cc;
      sp_ctrl <= load_sp;
      // write acca
      addr_ctrl <= push_ad;
      dout_ctrl <= acca_dout;
      next_state <= fetch_state;
    end
    pula_state : begin
      // default registers
      acca_ctrl <= latch_acca;
      accb_ctrl <= latch_accb;
      ix_ctrl <= latch_ix;
      pc_ctrl <= latch_pc;
      md_ctrl <= latch_md;
      iv_ctrl <= latch_iv;
      op_ctrl <= latch_op;
      nmi_ctrl <= latch_nmi;
      ea_ctrl <= latch_ea;
      // idle sp
      left_ctrl <= sp_left;
      right_ctrl <= zero_right;
      alu_ctrl <= alu_nop;
      cc_ctrl <= latch_cc;
      sp_ctrl <= latch_sp;
      // read acca
      acca_ctrl <= pull_acca;
      addr_ctrl <= pull_ad;
      dout_ctrl <= acca_dout;
      next_state <= fetch_state;
    end
    pshb_state : begin
      // default registers
      acca_ctrl <= latch_acca;
      accb_ctrl <= latch_accb;
      ix_ctrl <= latch_ix;
      pc_ctrl <= latch_pc;
      md_ctrl <= latch_md;
      iv_ctrl <= latch_iv;
      op_ctrl <= latch_op;
      nmi_ctrl <= latch_nmi;
      ea_ctrl <= latch_ea;
      // decrement sp
      left_ctrl <= sp_left;
      right_ctrl <= plus_one_right;
      alu_ctrl <= alu_sub16;
```

```
      cc_ctrl <= latch_cc;
      sp_ctrl <= load_sp;
      // write accb
      addr_ctrl <= push_ad;
      dout_ctrl <= accb_dout;
      next_state <= fetch_state;
    end
    pulb_state : begin
      // default
      acca_ctrl <= latch_acca;
      accb_ctrl <= latch_accb;
      ix_ctrl <= latch_ix;
      pc_ctrl <= latch_pc;
      md_ctrl <= latch_md;
      iv_ctrl <= latch_iv;
      op_ctrl <= latch_op;
      nmi_ctrl <= latch_nmi;
      ea_ctrl <= latch_ea;
      // idle sp
      left_ctrl <= sp_left;
      right_ctrl <= zero_right;
      alu_ctrl <= alu_nop;
      cc_ctrl <= latch_cc;
      sp_ctrl <= latch_sp;
      // read accb
      accb_ctrl <= pull_accb;
      addr_ctrl <= pull_ad;
      dout_ctrl <= accb_dout;
      next_state <= fetch_state;
    end
    pshx_lo_state : begin
      // default
      acca_ctrl <= latch_acca;
      accb_ctrl <= latch_accb;
      ix_ctrl <= latch_ix;
      sp_ctrl <= latch_sp;
      pc_ctrl <= latch_pc;
      md_ctrl <= latch_md;
      iv_ctrl <= latch_iv;
      op_ctrl <= latch_op;
      nmi_ctrl <= latch_nmi;
      ea_ctrl <= latch_ea;
      // decrement sp
      left_ctrl <= sp_left;
      right_ctrl <= plus_one_right;
      alu_ctrl <= alu_sub16;
      cc_ctrl <= latch_cc;
      sp_ctrl <= load_sp;
      // write ix low
      addr_ctrl <= push_ad;
      dout_ctrl <= ix_lo_dout;
      next_state <= pshx_hi_state;
    end
    pshx_hi_state : begin
      // default registers
      acca_ctrl <= latch_acca;
      accb_ctrl <= latch_accb;
      ix_ctrl <= latch_ix;
      pc_ctrl <= latch_pc;
      md_ctrl <= latch_md;
      iv_ctrl <= latch_iv;
      op_ctrl <= latch_op;
      nmi_ctrl <= latch_nmi;
      ea_ctrl <= latch_ea;
      // decrement sp
      left_ctrl <= sp_left;
      right_ctrl <= plus_one_right;
      alu_ctrl <= alu_sub16;
      cc_ctrl <= latch_cc;
      sp_ctrl <= load_sp;
      // write ix hi
      addr_ctrl <= push_ad;
      dout_ctrl <= ix_hi_dout;
      next_state <= fetch_state;
    end
    pulx_hi_state : begin
      // default
      acca_ctrl <= latch_acca;
      accb_ctrl <= latch_accb;
      pc_ctrl <= latch_pc;
      md_ctrl <= latch_md;
      iv_ctrl <= latch_iv;
      op_ctrl <= latch_op;
      nmi_ctrl <= latch_nmi;
      ea_ctrl <= latch_ea;
      // increment sp
      left_ctrl <= sp_left;
      right_ctrl <= plus_one_right;
      alu_ctrl <= alu_add16;
      cc_ctrl <= latch_cc;
      sp_ctrl <= load_sp;
```

```
      // pull ix hi
      ix_ctrl <= pull_hi_ix;
      addr_ctrl <= pull_ad;
      dout_ctrl <= ix_hi_dout;
      next_state <= pulx_lo_state;
    end
    pulx_lo_state : begin
      // default
      acca_ctrl <= latch_acca;
      accb_ctrl <= latch_accb;
      pc_ctrl <= latch_pc;
      md_ctrl <= latch_md;
      iv_ctrl <= latch_iv;
      op_ctrl <= latch_op;
      nmi_ctrl <= latch_nmi;
      ea_ctrl <= latch_ea;
      // idle sp
      left_ctrl <= sp_left;
      right_ctrl <= zero_right;
      alu_ctrl <= alu_nop;
      cc_ctrl <= latch_cc;
      sp_ctrl <= latch_sp;
      // read ix low
      ix_ctrl <= pull_lo_ix;
      addr_ctrl <= pull_ad;
      dout_ctrl <= ix_lo_dout;
      next_state <= fetch_state;
      //
      // return from interrupt
      // enter here from bogus interrupts
      //
    end
    rti_state : begin
      // default registers
      acca_ctrl <= latch_acca;
      accb_ctrl <= latch_accb;
      ix_ctrl <= latch_ix;
      pc_ctrl <= latch_pc;
      md_ctrl <= latch_md;
      iv_ctrl <= latch_iv;
      op_ctrl <= latch_op;
      nmi_ctrl <= latch_nmi;
      ea_ctrl <= latch_ea;
      // increment sp
      left_ctrl <= sp_left;
      right_ctrl <= plus_one_right;
      alu_ctrl <= alu_add16;
      sp_ctrl <= load_sp;
      // idle address bus
      cc_ctrl <= latch_cc;
      addr_ctrl <= idle_ad;
      dout_ctrl <= cc_dout;
      next_state <= rti_cc_state;
    end
    rti_cc_state : begin
      // default registers
      acca_ctrl <= latch_acca;
      accb_ctrl <= latch_accb;
      ix_ctrl <= latch_ix;
      pc_ctrl <= latch_pc;
      md_ctrl <= latch_md;
      iv_ctrl <= latch_iv;
      op_ctrl <= latch_op;
      nmi_ctrl <= latch_nmi;
      ea_ctrl <= latch_ea;
      // increment sp
      left_ctrl <= sp_left;
      right_ctrl <= plus_one_right;
      alu_ctrl <= alu_add16;
      sp_ctrl <= load_sp;
      // read cc
      cc_ctrl <= pull_cc;
      addr_ctrl <= pull_ad;
      dout_ctrl <= cc_dout;
      next_state <= rti_accb_state;
    end
    rti_accb_state : begin
      // default registers
      acca_ctrl <= latch_acca;
      ix_ctrl <= latch_ix;
      pc_ctrl <= latch_pc;
      md_ctrl <= latch_md;
      iv_ctrl <= latch_iv;
      op_ctrl <= latch_op;
      nmi_ctrl <= latch_nmi;
      ea_ctrl <= latch_ea;
      // increment sp
      left_ctrl <= sp_left;
      right_ctrl <= plus_one_right;
      alu_ctrl <= alu_add16;
      cc_ctrl <= latch_cc;
```

```
      sp_ctrl <= load_sp;
      // read accb
      accb_ctrl <= pull_accb;
      addr_ctrl <= pull_ad;
      dout_ctrl <= accb_dout;
      next_state <= rti_acca_state;
    end
    rti_acca_state : begin
      // default registers
      accb_ctrl <= latch_accb;
      ix_ctrl <= latch_ix;
      pc_ctrl <= latch_pc;
      md_ctrl <= latch_md;
      iv_ctrl <= latch_iv;
      op_ctrl <= latch_op;
      nmi_ctrl <= latch_nmi;
      ea_ctrl <= latch_ea;
      // increment sp
      left_ctrl <= sp_left;
      right_ctrl <= plus_one_right;
      alu_ctrl <= alu_add16;
      cc_ctrl <= latch_cc;
      sp_ctrl <= load_sp;
      // read acca
      acca_ctrl <= pull_acca;
      addr_ctrl <= pull_ad;
      dout_ctrl <= acca_dout;
      next_state <= rti_ixh_state;
    end
    rti_ixh_state : begin
      // default
      acca_ctrl <= latch_acca;
      accb_ctrl <= latch_accb;
      pc_ctrl <= latch_pc;
      md_ctrl <= latch_md;
      iv_ctrl <= latch_iv;
      op_ctrl <= latch_op;
      nmi_ctrl <= latch_nmi;
      ea_ctrl <= latch_ea;
      // increment sp
      left_ctrl <= sp_left;
      right_ctrl <= plus_one_right;
      alu_ctrl <= alu_add16;
      cc_ctrl <= latch_cc;
      sp_ctrl <= load_sp;
      // read ix hi
      ix_ctrl <= pull_hi_ix;
      addr_ctrl <= pull_ad;
      dout_ctrl <= ix_hi_dout;
      next_state <= rti_ixl_state;
    end
    rti_ixl_state : begin
      // default
      acca_ctrl <= latch_acca;
      accb_ctrl <= latch_accb;
      pc_ctrl <= latch_pc;
      md_ctrl <= latch_md;
      iv_ctrl <= latch_iv;
      op_ctrl <= latch_op;
      nmi_ctrl <= latch_nmi;
      ea_ctrl <= latch_ea;
      // increment sp
      left_ctrl <= sp_left;
      right_ctrl <= plus_one_right;
      alu_ctrl <= alu_add16;
      cc_ctrl <= latch_cc;
      sp_ctrl <= load_sp;
      // read ix low
      ix_ctrl <= pull_lo_ix;
      addr_ctrl <= pull_ad;
      dout_ctrl <= ix_lo_dout;
      next_state <= rti_pch_state;
    end
    rti_pch_state : begin
      // default
      acca_ctrl <= latch_acca;
      accb_ctrl <= latch_accb;
      ix_ctrl <= latch_ix;
      pc_ctrl <= latch_pc;
      md_ctrl <= latch_md;
      iv_ctrl <= latch_iv;
      op_ctrl <= latch_op;
      nmi_ctrl <= latch_nmi;
      ea_ctrl <= latch_ea;
      // increment sp
      left_ctrl <= sp_left;
      right_ctrl <= plus_one_right;
      alu_ctrl <= alu_add16;
      cc_ctrl <= latch_cc;
      sp_ctrl <= load_sp;
      // pull pc hi
```

```
      pc_ctrl <= pull_hi_pc;
      addr_ctrl <= pull_ad;
      dout_ctrl <= pc_hi_dout;
      next_state <= rti_pcl_state;
    end
    rti_pcl_state : begin
      // default
      acca_ctrl <= latch_acca;
      accb_ctrl <= latch_accb;
      ix_ctrl <= latch_ix;
      md_ctrl <= latch_md;
      iv_ctrl <= latch_iv;
      op_ctrl <= latch_op;
      nmi_ctrl <= latch_nmi;
      ea_ctrl <= latch_ea;
      // idle sp
      left_ctrl <= sp_left;
      right_ctrl <= zero_right;
      alu_ctrl <= alu_nop;
      cc_ctrl <= latch_cc;
      sp_ctrl <= latch_sp;
      // pull pc low
      pc_ctrl <= pull_lo_pc;
      addr_ctrl <= pull_ad;
      dout_ctrl <= pc_lo_dout;
      next_state <= fetch_state;
      //
      // here on interrupt
      // iv register hold interrupt type
      //
    end
    int_pcl_state : begin
      // default
      acca_ctrl <= latch_acca;
      accb_ctrl <= latch_accb;
      ix_ctrl <= latch_ix;
      pc_ctrl <= latch_pc;
      md_ctrl <= latch_md;
      iv_ctrl <= latch_iv;
      op_ctrl <= latch_op;
      nmi_ctrl <= latch_nmi;
      ea_ctrl <= latch_ea;
      // decrement sp
      left_ctrl <= sp_left;
      right_ctrl <= plus_one_right;
      alu_ctrl <= alu_sub16;
      cc_ctrl <= latch_cc;
      sp_ctrl <= load_sp;
      // write pc low
      addr_ctrl <= push_ad;
      dout_ctrl <= pc_lo_dout;
      next_state <= int_pch_state;
    end
    int_pch_state : begin
      // default
      acca_ctrl <= latch_acca;
      accb_ctrl <= latch_accb;
      ix_ctrl <= latch_ix;
      pc_ctrl <= latch_pc;
      md_ctrl <= latch_md;
      iv_ctrl <= latch_iv;
      op_ctrl <= latch_op;
      nmi_ctrl <= latch_nmi;
      ea_ctrl <= latch_ea;
      // decrement sp
      left_ctrl <= sp_left;
      right_ctrl <= plus_one_right;
      alu_ctrl <= alu_sub16;
      cc_ctrl <= latch_cc;
      sp_ctrl <= load_sp;
      // write pc hi
      addr_ctrl <= push_ad;
      dout_ctrl <= pc_hi_dout;
      next_state <= int_ixl_state;
    end
    int_ixl_state : begin
      // default
      acca_ctrl <= latch_acca;
      accb_ctrl <= latch_accb;
      ix_ctrl <= latch_ix;
      pc_ctrl <= latch_pc;
      md_ctrl <= latch_md;
      iv_ctrl <= latch_iv;
      op_ctrl <= latch_op;
      nmi_ctrl <= latch_nmi;
      ea_ctrl <= latch_ea;
      // decrement sp
      left_ctrl <= sp_left;
      right_ctrl <= plus_one_right;
      alu_ctrl <= alu_sub16;
      cc_ctrl <= latch_cc;
```

```
      sp_ctrl <= load_sp;
      // write ix low
      addr_ctrl <= push_ad;
      dout_ctrl <= ix_lo_dout;
      next_state <= int_ixh_state;
    end
    int_ixh_state : begin
      // default
      acca_ctrl <= latch_acca;
      accb_ctrl <= latch_accb;
      ix_ctrl <= latch_ix;
      pc_ctrl <= latch_pc;
      md_ctrl <= latch_md;
      iv_ctrl <= latch_iv;
      op_ctrl <= latch_op;
      nmi_ctrl <= latch_nmi;
      ea_ctrl <= latch_ea;
      // decrement sp
      left_ctrl <= sp_left;
      right_ctrl <= plus_one_right;
      alu_ctrl <= alu_sub16;
      cc_ctrl <= latch_cc;
      sp_ctrl <= load_sp;
      // write ix hi
      addr_ctrl <= push_ad;
      dout_ctrl <= ix_hi_dout;
      next_state <= int_acca_state;
    end
    int_acca_state : begin
      // default
      acca_ctrl <= latch_acca;
      accb_ctrl <= latch_accb;
      ix_ctrl <= latch_ix;
      pc_ctrl <= latch_pc;
      md_ctrl <= latch_md;
      iv_ctrl <= latch_iv;
      op_ctrl <= latch_op;
      nmi_ctrl <= latch_nmi;
      ea_ctrl <= latch_ea;
      // decrement sp
      left_ctrl <= sp_left;
      right_ctrl <= plus_one_right;
      alu_ctrl <= alu_sub16;
      cc_ctrl <= latch_cc;
      sp_ctrl <= load_sp;
      // write acca
      addr_ctrl <= push_ad;
      dout_ctrl <= acca_dout;
      next_state <= int_accb_state;
    end
    int_accb_state : begin
      // default
      acca_ctrl <= latch_acca;
      accb_ctrl <= latch_accb;
      ix_ctrl <= latch_ix;
      pc_ctrl <= latch_pc;
      md_ctrl <= latch_md;
      iv_ctrl <= latch_iv;
      op_ctrl <= latch_op;
      nmi_ctrl <= latch_nmi;
      ea_ctrl <= latch_ea;
      // decrement sp
      left_ctrl <= sp_left;
      right_ctrl <= plus_one_right;
      alu_ctrl <= alu_sub16;
      cc_ctrl <= latch_cc;
      sp_ctrl <= load_sp;
      // write accb
      addr_ctrl <= push_ad;
      dout_ctrl <= accb_dout;
      next_state <= int_cc_state;
    end
    int_cc_state : begin
      // default
      acca_ctrl <= latch_acca;
      accb_ctrl <= latch_accb;
      ix_ctrl <= latch_ix;
      pc_ctrl <= latch_pc;
      md_ctrl <= latch_md;
      op_ctrl <= latch_op;
      nmi_ctrl <= latch_nmi;
      ea_ctrl <= latch_ea;
      // decrement sp
      left_ctrl <= sp_left;
      right_ctrl <= plus_one_right;
      alu_ctrl <= alu_sub16;
      cc_ctrl <= latch_cc;
      sp_ctrl <= load_sp;
      // write cc
      addr_ctrl <= push_ad;
      dout_ctrl <= cc_dout;
```

```
      nmi_ctrl <= latch_nmi;
      //
      // nmi is edge triggered
      // nmi_req is cleared when nmi goes low.
      //
      if(nmi_req == 1'b1) begin
        iv_ctrl <= nmi_iv;
        next_state <= vect_hi_state;
      end
      else begin
        //
        // IRQ is level sensitive
        //
        if((irq == 1'b1) && (cc[IBIT]  == 1'b0)) begin
          iv_ctrl <= irq_iv;
          next_state <= int_mask_state;
        end
        else if((irq_icf == 1'b1) && (cc[IBIT]  == 1'b0)) begin
          iv_ctrl <= icf_iv;
          next_state <= int_mask_state;
        end
        else if((irq_ocf == 1'b1) && (cc[IBIT]  == 1'b0)) begin
          iv_ctrl <= ocf_iv;
          next_state <= int_mask_state;
        end
        else if((irq_tof == 1'b1) && (cc[IBIT]  == 1'b0)) begin
          iv_ctrl <= tof_iv;
          next_state <= int_mask_state;
        end
        else if((irq_sci == 1'b1) && (cc[IBIT]  == 1'b0)) begin
          iv_ctrl <= sci_iv;
          next_state <= int_mask_state;
        end
        else begin
          case(op_code)
          8'b 00111110 : begin
            // WAI (wait for interrupt)
            iv_ctrl <= latch_iv;
            next_state <= int_wai_state;
          end
          8'b 00111111 : begin
            // SWI (Software interrupt)
            iv_ctrl <= swi_iv;
            next_state <= vect_hi_state;
          end
          default : begin
            // bogus interrupt (return)
            iv_ctrl <= latch_iv;
            next_state <= rti_state;
          end
          endcase
        end
      end
    end
    int_wai_state : begin
      // default
      acca_ctrl <= latch_acca;
      accb_ctrl <= latch_accb;
      ix_ctrl <= latch_ix;
      pc_ctrl <= latch_pc;
      md_ctrl <= latch_md;
      op_ctrl <= latch_op;
      ea_ctrl <= latch_ea;
      // enable interrupts
      left_ctrl <= sp_left;
      right_ctrl <= plus_one_right;
      alu_ctrl <= alu_cli;
      cc_ctrl <= load_cc;
      sp_ctrl <= latch_sp;
      // idle bus
      addr_ctrl <= idle_ad;
      dout_ctrl <= cc_dout;
      if((nmi_req == 1'b1) && (nmi_ack == 1'b0)) begin
        iv_ctrl <= nmi_iv;
        nmi_ctrl <= set_nmi;
        next_state <= vect_hi_state;
      end
      else begin
        //
        // nmi request is not cleared until nmi input goes low
        //
        if((nmi_req == 1'b0) && (nmi_ack == 1'b1)) begin
          nmi_ctrl <= reset_nmi;
        end
        else begin
          nmi_ctrl <= latch_nmi;
        end
        //
        // IRQ is level sensitive
        //
        if((irq == 1'b1) && (cc[IBIT]  == 1'b0)) begin
```

```
          iv_ctrl <= irq_iv;
          next_state <= int_mask_state;
        end
        else if((irq_icf == 1'b1) && (cc[IBIT]  == 1'b0)) begin
          iv_ctrl <= icf_iv;
          next_state <= int_mask_state;
        end
        else if((irq_ocf == 1'b1) && (cc[IBIT]  == 1'b0)) begin
          iv_ctrl <= ocf_iv;
          next_state <= int_mask_state;
        end
        else if((irq_tof == 1'b1) && (cc[IBIT]  == 1'b0)) begin
          iv_ctrl <= tof_iv;
          next_state <= int_mask_state;
        end
        else if((irq_sci == 1'b1) && (cc[IBIT]  == 1'b0)) begin
          iv_ctrl <= sci_iv;
          next_state <= int_mask_state;
        end
        else begin
          iv_ctrl <= latch_iv;
          next_state <= int_wai_state;
        end
      end
    end
    int_mask_state : begin
      // default
      acca_ctrl <= latch_acca;
      accb_ctrl <= latch_accb;
      ix_ctrl <= latch_ix;
      pc_ctrl <= latch_pc;
      md_ctrl <= latch_md;
      iv_ctrl <= latch_iv;
      op_ctrl <= latch_op;
      nmi_ctrl <= latch_nmi;
      ea_ctrl <= latch_ea;
      // Mask IRQ
      left_ctrl <= sp_left;
      right_ctrl <= zero_right;
      alu_ctrl <= alu_sei;
      cc_ctrl <= load_cc;
      sp_ctrl <= latch_sp;
      // idle bus cycle
      addr_ctrl <= idle_ad;
      dout_ctrl <= md_lo_dout;
      next_state <= vect_hi_state;
    end
    halt_state : begin
      // halt CPU.
      // default
      acca_ctrl <= latch_acca;
      accb_ctrl <= latch_accb;
      ix_ctrl <= latch_ix;
      sp_ctrl <= latch_sp;
      pc_ctrl <= latch_pc;
      md_ctrl <= latch_md;
      iv_ctrl <= latch_iv;
      op_ctrl <= latch_op;
      nmi_ctrl <= latch_nmi;
      ea_ctrl <= latch_ea;
      // do nothing in ALU
      left_ctrl <= acca_left;
      right_ctrl <= zero_right;
      alu_ctrl <= alu_nop;
      cc_ctrl <= latch_cc;
      // idle bus cycle
      addr_ctrl <= idle_ad;
      dout_ctrl <= md_lo_dout;
      if(halt == 1'b1) begin
        next_state <= halt_state;
      end
      else begin
        next_state <= fetch_state;
      end
    end
    default : begin
      // error state halt on undefine states
      // default
      acca_ctrl <= latch_acca;
      accb_ctrl <= latch_accb;
      ix_ctrl <= latch_ix;
      sp_ctrl <= latch_sp;
      pc_ctrl <= latch_pc;
      md_ctrl <= latch_md;
      iv_ctrl <= latch_iv;
      op_ctrl <= latch_op;
      nmi_ctrl <= latch_nmi;
      ea_ctrl <= latch_ea;
      // do nothing in ALU
      left_ctrl <= acca_left;
      right_ctrl <= zero_right;
```

```
      alu_ctrl <= alu_nop;
      cc_ctrl <= latch_cc;
      // idle bus cycle
      addr_ctrl <= idle_ad;
      dout_ctrl <= md_lo_dout;
      next_state <= error_state;
    end
    endcase
  end
// endmodule

  //------------------------------
  //
  // state machine
  //
  //------------------------------
  always @ (negedge clk) //(negedge clk or negedge rst or negedge state or negedge hold)
    begin
    if(rst == 1'b1) begin
      state <= reset_state;
    end
    else if(hold == 1'b1) begin
      state <= state;
    end
    else begin
      state <= next_state;
    end
  end
endmodule
```