```
file name: mipscore.v
```

```
    1  /*************************************************************************************************/
    2  /* Sample Verilog HDL Code for CSC.T363 Computer Architecture                 Arch Lab. TOKYO TECH */
    3  /*************************************************************************************************/
    4  `include "define.v"
    5  /*************************************************************************************************/
    6  /* MipsCore: 32bit-MIPS 5-stage pipelining processor                                             */
    7  /*************************************************************************************************/
    8  module MIPSCORE(CLK, RST_X, STALL, STAT, I_ADDR, I_IN, D_ADDR, D_IN, D_OUT, D_OE, D_WE, IO_IN);
    9      input  wire         CLK, RST_X, STALL;
   10      output reg  [2:0]   STAT;
   11      output wire [`ADDR] I_ADDR, D_ADDR;
   12      input  wire [31:0]  I_IN, D_IN, IO_IN;
   13      output wire [31:0]  D_OUT;
   14      output wire [3:0]   D_WE;
   15      output wire         D_OE;
   16
   17      reg [`ADDR]  pc;          // program counter
   18      assign I_ADDR = pc;
   19      /*************************************************************************************************/
   20      reg [`ADDR] IfId_npc;     // IF-ID pipeline reg: next pc, pc + 4
   21      reg [31:0]  IfId_ir;      // IF-ID pipeline reg: instruction
   22
   23      reg [`ADDR] IdEx_npc;     // ID-EX pipeline reg: next pc, pc + 4
   24      reg [31:0]  IdEx_rrs;     // ID-EX pipeline reg: fetched operand of R[rs]
   25      reg [31:0]  IdEx_rrt;     // ID-EX pipeline reg: fetched operand of R[rt]
   26      reg [4:0]   IdEx_dst;     // ID-EX pipeline reg: decoded destination reg number
   27      reg [31:0]  IdEx_ir;      // ID-EX pipeline reg: instruction
   28      reg [`OPNT] IdEx_opn;     // ID-EX pipeline reg: decoded operation ID
   29      reg [`ATTR] IdEx_attr;    // ID-EX pipeline reg: decoded instruction attribute
   30
   31      reg [4:0]   ExMa_dst;     // EX-MA pipeline reg: decoded destination reg number
   32      reg [31:0]  ExMa_rslt;    // EX-MA pipeline reg: execution result
   33      reg [3:0]   ExMa_mwe;     // EX-MA pipeline reg: mem write enable
   34      reg         ExMa_oe;      // EX-MA pipeline reg: data memory output enable
   35      reg [4:0]   ExMa_lds;     // EX-MA pipeline reg: load selector
   36      reg [31:0]  ExMa_std;     // EX-MA pipeline reg: store data
   37
   38      reg [4:0]   MaWb_dst;     // MA-WB pipeline reg: decoded destination reg number
   39      reg [31:0]  MaWb_rslt;    // MA-WB pipeline reg: execution result
   40
   41      /*************************************************************************************************/
   42      reg [`ADDR] IdTPC;        // wire, calculated branch Taken address (Taken Program Counter)
   43      reg         IdC;          // wire, Calculated branch result, branch Taken or Untaken
   44      reg         IdB;          // wire, branch/jump instruction ?
   45      wire [31:0] MaRSLT;       // wire, execution result on MA stage
   46      wire        bstall;       // stall signal by branch instruction
   47      wire        PSTALL = STALL;  // pipeline stall
   48
   49      /*************************************************************************************************/
   50      /* Stage 1 : IF, instruction fetch                                                               */
   51      /*************************************************************************************************/
   52      wire [`ADDR] IfNPC = pc + 4;
   53
   54      always @( posedge CLK ) begin ///// update program counter
   55          if(!RST_X)                pc <= 0;
   56          else if(!PSTALL && !bstall) pc <= (IdC) ? IdTPC : IfNPC; // If branch taken, uses taken PC
   57      end
   58
   59      always @( posedge CLK ) begin  ///// update pipeline registers
   60          if(!RST_X) {IfId_npc, IfId_ir} <= 0;
   61          else if(!PSTALL && !bstall) begin
   62              IfId_npc <= IfNPC;
   63              IfId_ir  <= I_IN;  // if branch taken then NOP else instruction from imem.
   64          end
   65      end
   66
   67      /*************************************************************************************************/
   68      /* Stage 2 : ID, instruction decode & operand fetch                                              */
   69      /*************************************************************************************************/
   70      wire [5:0] IdOP   = IfId_ir[31:26]; // opcode field of instruction
   71      wire [4:0] IdRS   = IfId_ir[25:21]; // rs field of instruction
   72      wire [4:0] IdRT   = IfId_ir[20:16]; // rt field of instruction
   73      wire [4:0] IdRD   = IfId_ir[15:11]; // rd field of instruction
   74      wire [5:0] IdFCT  = IfId_ir[5:0];   // funct field of instruction
   75      wire [31:0] IdRRS, IdRRT;           // register value of rs and rt
   76
   77      assign bstall = IdB &&
   78                      ((IdRS!=0 && (IdRS==IdEx_dst || IdRS==ExMa_dst)) ||
   79                       (IdRT!=0 && (IdRT==IdEx_dst || IdRT==ExMa_dst))); // branch stall
   80
   81      GPR gpr(.CLK(CLK), .REGNUM0(IdRS), .REGNUM1(IdRT), .DOUT0(IdRRS), .DOUT1(IdRRT), // register file
   82              .REGNUM2(ExMa_dst), .DIN0(MaRSLT), .WE0(!PSTALL));
   83
   84      reg [6:0]   IdOPN;   // instruction operation number (unique operation ID)
   85      reg [4:0]   IdDST;   // destination register
   86      reg [`ATTR] IdATTR;  // instruction attribute
```



```
   87      always @(IfId_ir) begin
   88          IdOPN   = `ERROR____;
   89          IdDST  = 0;
   90          IdATTR = 0;
   91          case (IdOP) // OP
   92              6'h00: case (IdFCT) // FUNCT
   93                  6'h00: begin IdOPN= (IdRD) ? `SLL______ : `NOP______; IdDST=IdRD; end
   94                  6'h02: begin IdOPN=`SRL______; IdDST=IdRD;                        end
   95                  6'h03: begin IdOPN=`SRA______; IdDST=IdRD;                        end
   96                  6'h04: begin IdOPN=`SLLV_____; IdDST=IdRD;                        end
   97                  6'h06: begin IdOPN=`SRLV_____; IdDST=IdRD;                        end
   98                  6'h07: begin IdOPN=`SRAV_____; IdDST=IdRD;                        end
   99                  6'h08: begin IdOPN=`JR_______; IdDST=0;                           end
  100                  6'h09: begin IdOPN=`JALR_____; IdDST=31;                          end
  101                  6'h20: begin IdOPN=`ADD______; IdDST=IdRD;                        end
  102                  6'h21: begin IdOPN=`ADDU_____; IdDST=IdRD;                        end
  103                  6'h22: begin IdOPN=`SUB______; IdDST=IdRD;                        end
  104                  6'h23: begin IdOPN=`SUBU_____; IdDST=IdRD;                        end
  105                  6'h24: begin IdOPN=`AND______; IdDST=IdRD;                        end
  106                  6'h25: begin IdOPN=`OR_______; IdDST=IdRD;                        end
  107                  6'h26: begin IdOPN=`XOR______; IdDST=IdRD;                        end
  108                  6'h27: begin IdOPN=`NOR______; IdDST=IdRD;                        end
  109                  6'h2a: begin IdOPN=`SLT______; IdDST=IdRD;                        end
  110                  6'h2b: begin IdOPN=`SLTU_____; IdDST=IdRD;                        end
  111              endcase
  112              6'h01: case (IdRT)
  113                  5'h00: begin IdOPN=`BLTZ_____; IdDST=0;                           end
  114                  5'h01: begin IdOPN=`BGEZ_____; IdDST=0;                           end
  115              endcase
  116              6'h02:     begin IdOPN=`J________; IdDST=0;                           end
  117              6'h03:     begin IdOPN=`JAL______; IdDST=31;                          end
  118              6'h04:     begin IdOPN=`BEQ______; IdDST=0;                           end
  119              6'h05:     begin IdOPN=`BNE______; IdDST=0;                           end
  120              6'h06:     begin IdOPN=`BLEZ_____; IdDST=0;                           end
  121              6'h07:     begin IdOPN=`BGTZ_____; IdDST=0;                           end
  122              6'h08:     begin IdOPN=`ADDI_____; IdDST=IdRT;                        end
  123              6'h09:     begin IdOPN=`ADDIU____; IdDST=IdRT;                        end
  124              6'h0a:     begin IdOPN=`SLTI_____; IdDST=IdRT;                        end
  125              6'h0b:     begin IdOPN=`SLTIU____; IdDST=IdRT;                        end
  126              6'h0c:     begin IdOPN=`ANDI_____; IdDST=IdRT;                        end
  127              6'h0d:     begin IdOPN=`ORI______; IdDST=IdRT;                        end
  128              6'h0e:     begin IdOPN=`XORI_____; IdDST=IdRT;                        end
  129              6'h0f:     begin IdOPN=`LUI______; IdDST=IdRT;                        end
  130              6'h23:     begin IdOPN=`LW_______; IdDST=IdRT; IdATTR=`LD_4B;         end
  131              6'h2b:     begin IdOPN=`SW_______; IdDST=0;    IdATTR=`ST_4B;         end
  132          endcase
  133      end
  134
  135      wire [`ADDR] IdBPC = IfId_npc + ({{16{IfId_ir[15]}}, IfId_ir[15:0]} << 2);
  136      always @(*) begin  ////// branch & jump resolution unit
  137          {IdTPC, IdC, IdB} = 0;
  138          case (IdOP)
  139              6'h04: begin IdB=1; IdTPC = IdBPC; IdC = (IdRRS == IdRRT);         end        // BEQ
  140              6'h05: begin IdB=1; IdTPC = IdBPC; IdC = (IdRRS != IdRRT);         end        // BNE
  141              6'h06: begin IdB=1; IdTPC = IdBPC; IdC = ( IdRRS[31]||(IdRRS==0)); end        // BLEZ
  142              6'h07: begin IdB=1; IdTPC = IdBPC; IdC = (~IdRRS[31]&&(IdRRS!=0)); end        // BGTZ
  143              6'h01: begin IdB=1; IdTPC = IdBPC; IdC = (IdRT) ? ~IdRRS[31] : IdRRS[31]; end // BGEZ,BLTZ
  144              6'h02: begin IdB=1; IdTPC = {IfId_npc[31:28], IfId_ir[25:0], 2'b0}; IdC=1;end // J
  145              6'h03: begin IdB=1; IdTPC = {IfId_npc[31:28], IfId_ir[25:0], 2'b0}; IdC=1;end // JAL
  146              6'h00: if      (IdFCT==6'h08) begin IdB=1; IdTPC = IdRRS; IdC = 1; end        // JR
  147                     else if (IdFCT==6'h09) begin IdB=1; IdTPC = IdRRS; IdC = 1; end        // JALR
  148          endcase
  149      end
  150
  151      always @(posedge CLK) begin ///// update pipeline registers
  152          if(!RST_X) {IdEx_npc, IdEx_rrs, IdEx_rrt, IdEx_dst, IdEx_ir, IdEx_opn, IdEx_attr} <= 0;
  153          else if(!PSTALL) begin
  154              IdEx_npc   <= (bstall) ? 0 : IfId_npc;
  155              IdEx_rrs   <= (bstall) ? 0 : IdRRS;    // data from general-purpose register file
  156              IdEx_rrt   <= (bstall) ? 0 : IdRRT;    // data from general-purpose register file
  157              IdEx_dst   <= (bstall) ? 0 : IdDST;
  158              IdEx_ir    <= (bstall) ? 0 : IfId_ir;
  159              IdEx_opn   <= (bstall) ? 0 : IdOPN;
  160              IdEx_attr  <= (bstall) ? 0 : IdATTR;
  161          end
  162      end
  163
  164      /*************************************************************************************************/
  165      /* Stage 3 : EX, execute & address generation for LD/ST                                          */
  166      /*************************************************************************************************/
  167      wire        [31:0] RRS_U, RRT_U;                    // output of data forwarding unit
  168      wire signed [31:0] RRS_S = RRS_U;                   // signed wire
  169      wire signed [31:0] RRT_S = RRT_U;                   // signed wire
  170      wire         [4:0] SHAMT  = IdEx_ir[10:6];          // Shift amount
  171      wire        [15:0] IMM    = IdEx_ir[15:0];          // Immediate
  172      wire        [31:0] SET32I = {{16{IMM[15]}}, IMM};   // Sign Extended Imm.
  173      wire       [`ADDR] SETADI = SET32I[`ADDR] << 2;     // shifted immediate of address bit width
  174      wire       [`ADDR] JADDR  = IdEx_ir[`ADDR] << 2;    // Juma address
```

```
175     wire       [‘ADDR] ExA = RRS_U[‘ADDR] + SET32I[‘ADDR]; // mem address of LD/ST
176     reg         [31:0] ExRSLT;                             // execution result
177     reg          [3:0] ExWE;                               // memory write enable vector
178
179     FORWARDING frs(.SRC(IdEx_ir[25:21]), .DIN0(IdEx_rrs), .DOUT(RRS_U),
180                    .DST1(ExMa_dst), .DIN1(ExMa_rslt), .DST2(MaWb_dst), .DIN2(MaWb_rslt));
181     FORWARDING frt(.SRC(IdEx_ir[20:16]), .DIN0(IdEx_rrt), .DOUT(RRT_U),
182                    .DST1(ExMa_dst), .DIN1(ExMa_rslt), .DST2(MaWb_dst), .DIN2(MaWb_rslt));
183
184     always @(*) begin
185         {ExRSLT, ExWE} = 0;
186         case ( IdEx_opn )
187           ‘ADD_____ : begin ExRSLT = RRS_U + RRT_U;                                        end
188           ‘ADDI____ : begin ExRSLT = RRS_U + SET32I;                                       end
189           ‘ADDIU___ : begin ExRSLT = RRS_U + SET32I;                                       end
190           ‘ADDU____ : begin ExRSLT = RRS_U + RRT_U;                                        end
191           ‘SUB_____ : begin ExRSLT = RRS_U - RRT_U;                                        end
192           ‘SUBU____ : begin ExRSLT = RRS_U - RRT_U;                                        end
193           ‘AND_____ : begin ExRSLT = RRS_U & RRT_U;                                        end
194           ‘ANDI____ : begin ExRSLT = RRS_U & {16’h0, IMM};                                 end
195           ‘NOR_____ : begin ExRSLT = ~(RRS_U | RRT_U);                                     end
196           ‘OR______ : begin ExRSLT = RRS_U | RRT_U;                                        end
197           ‘ORI_____ : begin ExRSLT = RRS_U | {16’h0, IMM};                                 end
198           ‘XOR_____ : begin ExRSLT = RRS_U ^ RRT_U;                                        end
199           ‘XORI____ : begin ExRSLT = RRS_U ^ {16’h0, IMM};                                 end
200           ‘SLL_____ : begin ExRSLT = RRT_U <<  SHAMT;                                      end
201           ‘SRL_____ : begin ExRSLT = RRT_U >>  SHAMT;                                      end
202           ‘SRA_____ : begin ExRSLT = RRT_S >>> SHAMT;                                      end
203           ‘SLLV____ : begin ExRSLT = RRT_U <<  RRS_U[4:0];                                 end
204           ‘SRLV____ : begin ExRSLT = RRT_U >>  RRS_U[4:0];                                 end
205           ‘SRAV____ : begin ExRSLT = RRT_S >>> RRS_U[4:0];                                 end
206           ‘SLT_____ : begin ExRSLT = (RRS_U[31] ^ RRT_U[31]) ? RRS_U[31] : (RRS_U < RRT_U);    end
207           ‘SLTI____ : begin ExRSLT = (RRS_U[31] ^   IMM[15]) ? RRS_U[31] : (RRS_U < SET32I);   end
208           ‘SLTIU___ : begin ExRSLT = (RRS_U < SET32I);                                     end
209           ‘SLTU____ : begin ExRSLT = (RRS_U < RRT_U);                                      end
210           ‘JAL_____ : begin ExRSLT = IdEx_npc + 4;                                         end
211           ‘JALR____ : begin ExRSLT = IdEx_npc + 4;                                         end
212           ‘LUI_____ : begin ExRSLT = {IMM, 16’h0};                                         end
213           ‘LW______ : begin ExRSLT = {ExA[31:2], 2’b00};                                   end
214           ‘SW______ : begin ExRSLT = {ExA[31:2], 2’b00}; ExWE = 4’b1111;                   end
215         endcase
216     end
217
218     always @( posedge CLK ) begin ///// update pipeline registers
219         if(!RST_X) {ExMa_rslt, ExMa_dst, ExMa_mwe, ExMa_oe, ExMa_lds, ExMa_std} <= 0;
220         else if(!PSTALL) begin
221             ExMa_rslt <= ExRSLT;
222             ExMa_dst  <= IdEx_dst;
223             ExMa_std  <= RRT_U;
224             ExMa_mwe  <= ExWE;
225             ExMa_oe   <= (IdEx_attr & ‘LDST_ANY) ? 1 : 0;
226             ExMa_lds  <= (IdEx_opn==‘LW______) ?  1 : 0;
227         end
228     end
229
230     /*************************************************************************************************/
231     /* Stage 4 : MA, memory access for LD/ST                                                         */
232     /*************************************************************************************************/
233     assign D_ADDR = (ExMa_oe)            ? ExMa_rslt : 0;
234     assign D_OUT  = (ExMa_oe)            ? ExMa_std  : 0;
235     assign D_WE   = (ExMa_oe && !PSTALL) ? ExMa_mwe  : 0;
236     assign D_OE   =  ExMa_oe;
237
238     // wire [31:0] MaLD_ = (ExMa_rslt[23]) ? IO_IN : D_IN; // use I/O input if the high address
239     wire [31:0] MaLD_ = D_IN; // This edition does not use I/O.
240     wire [31:0] MaLDD = MaLD_ >> (8*ExMa_rslt[1:0]);    // loaded data, align data here
241
242     assign MaRSLT = (ExMa_lds) ? MaLDD : ExMa_rslt;
243
244     always @( posedge CLK ) begin ///// update pipeline registers
245         if(!RST_X) {MaWb_rslt, MaWb_dst} <= 0;
246         else if(!PSTALL) begin
247             MaWb_rslt <= MaRSLT;
248             MaWb_dst  <= ExMa_dst;
249         end
250     end
251
252     /* Update STATUS_LED (ERROR_LED)                                                                 */
253     /*************************************************************************************************/
254     wire [‘OPNT] ExOPN = IdEx_opn;
255     always @( posedge CLK ) begin
256         if(!RST_X) STAT <= 0;
257         else if(!PSTALL) begin
258             if(IdOPN==‘ERROR____) STAT[0] <= 1; // instruction decode error!
259             if(((ExOPN==‘LW______ || ExOPN==‘SW______) && ExA[1:0]!=0))
260                STAT[1] <= 1; // memory access alignment error!
261         end
262     end
```



```
263 endmodule
264
265 /*************************************************************************************************/
266 /* 32bitx32 2R/1W General Purpose Registers (Register File)                                      */
267 /*************************************************************************************************/
268 module GPR(CLK, REGNUM0, REGNUM1, REGNUM2, DIN0, WE0, DOUT0, DOUT1);
269     input wire        CLK;
270     input wire  [4:0] REGNUM0, REGNUM1, REGNUM2;
271     input wire [31:0] DIN0;
272     input wire        WE0;
273     output reg [31:0] DOUT0, DOUT1;
274
275     reg [31:0] r[0:31];
276
277     always @(negedge CLK) DOUT0 <= (REGNUM0==0) ? 0 : r[REGNUM0];
278     always @(negedge CLK) DOUT1 <= (REGNUM1==0) ? 0 : r[REGNUM1];
279     always @(posedge CLK) if(WE0) r[REGNUM2] <= DIN0;
280 endmodule
281
282 /***** data forwarding unit                                                                      *****/
283 /*************************************************************************************************/
284 module FORWARDING(SRC, DIN0, DST1, DIN1, DST2, DIN2, DOUT);
285     input  wire  [4:0] SRC, DST1, DST2;   // register number
286     input  wire [31:0] DIN0, DIN1, DIN2;  // 32bit value
287     output wire [31:0] DOUT;              // 32bit data output
288
289     assign DOUT = (SRC!=0 && DST1==SRC) ? DIN1 : (SRC!=0 && DST2==SRC) ? DIN2 : DIN0;
290 endmodule
291 /*************************************************************************************************/
```