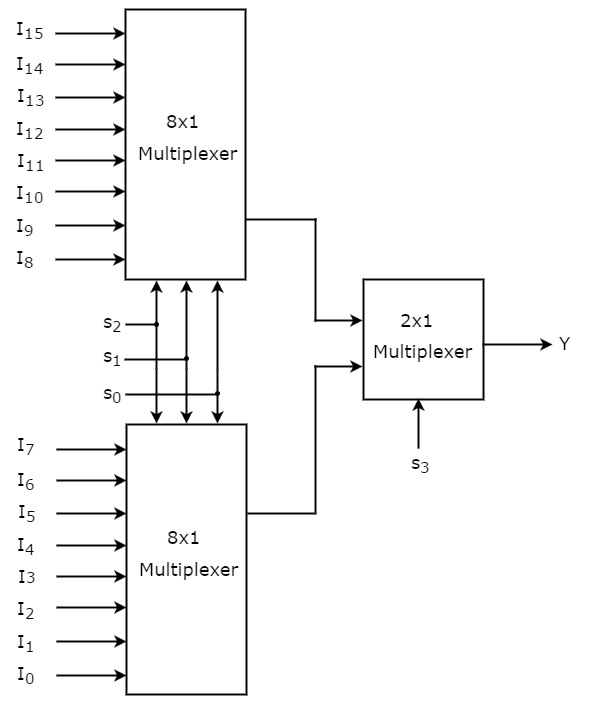
|  |  |
| --- | --- |
| COMPUTER ORGANIZATION : CSE 331 |  |
| **# assignment 3 REPORT #** |

LEMYE CEREN GUMUS - 151044071

Ceren GÜMÜŞ

COMPUTER ORGANIZATION : CSE 331

# assignment 3 REPORT #



First,16 X 1 mux was designed.

**Module MUX\_16(*result,select,input0,input1,input2,input3,input4,input5,***

***input6,input7,input8,input9,input10,input11,input12,input13,input14,input15*);**

***input input0,input1,input2,input3,input4,input5,input6,input7,input8, input9,input10,input11,input12,input13,input14,input15;***

**input [3:0] select;**

**output result;**

**wire for\_mux1,for\_mux2;**

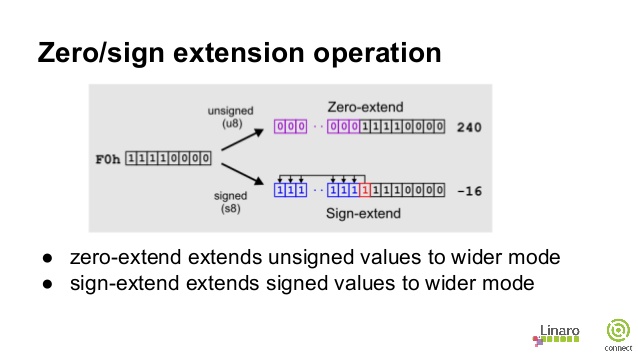
**MUX\_8 r1(for\_mux1, select[2:0],input0,input1,input2,input3,input4,input5,input6,input7);**

**MUX\_8 r2(for\_mux2, select[2:0],input8,input9,input10,input11,input12,input13,input14,input15);**

**MUX\_2 res(result,for\_mux1,for\_mux2,select[3]);**

**endmodule**

second concat is implemented



**module concat(extend\_data,shamt);**

**output [31:0] extend\_data;**

**input [4:0] shamt;**

**and and0(extend\_data[0],shamt[0],1);**

**and and1(extend\_data[1],shamt[1],1);**

**and and2(extend\_data[2],shamt[2],1);**

**and and3(extend\_data[3],shamt[3],1);**

**and and4(extend\_data[4],shamt[4],1);**

**and and5(extend\_data[5],0,0);**

**and and6(extend\_data[6],0,0);**

**and and7(extend\_data[7],0,0);**

**and and8(extend\_data[8],0,0);**

**and and9(extend\_data[9],0,0);**

**and and10(extend\_data[10],0,0);**

**and and11(extend\_data[11],0,0);**

**and and12(extend\_data[12],0,0);**

**and and13(extend\_data[13],0,0);**

**and and14(extend\_data[14],0,0);**

**and and15(extend\_data[15],0,0);**

**and and16(extend\_data[16],0,0);**

**and and17(extend\_data[17],0,0);**

**and and18(extend\_data[18],0,0);**

**and and19(extend\_data[19],0,0);**

**and and20(extend\_data[20],0,0);**

**and and21(extend\_data[21],0,0);**

**and and22(extend\_data[22],0,0);**

**and and23(extend\_data[23],0,0);**

**and and24(extend\_data[24],0,0);**

**and and25(extend\_data[25],0,0);**

**and and26(extend\_data[26],0,0);**

**and and27(extend\_data[27],0,0);**

**and and28(extend\_data[28],0,0);**

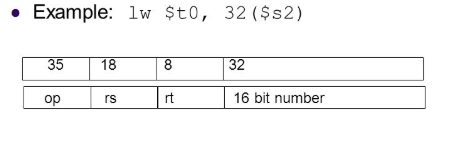
**and and29(extend\_data[29],0,0);**

**and and30(extend\_data[30],0,0);**

**and and31(extend\_data[31],0,0);**

**endmodule**

Third parser was implemented



**module parser(regSource,regTarget,immed,opcode,instruction);**

**input [31:0] instruction;**

**output [4:0] regSource,regTarget;**

**output [5:0] opcode;**

**output [15:0] immed;**

**//opcode**

**MUX\_2 m1(opcode[5],instruction[31],instruction[31],1'b0);**

**MUX\_2 m2(opcode[4],instruction[30],instruction[30],1'b0);**

**MUX\_2 m3(opcode[3],instruction[29],instruction[29],1'b0);**

**MUX\_2 m4(opcode[2],instruction[28],instruction[28],1'b0);**

**MUX\_2 m5(opcode[1],instruction[27],instruction[27],1'b0);**

**MUX\_2 m6(opcode[0],instruction[26],instruction[26],1'b0);**

**//controlUnit contSig(regWriteSignal,regDest,destReg,memReadSig,memWriteSig,opcode,funcCode,instruction);**

**//assign regSource = instruction[25:21]; //regSource**

**MUX\_2 m13(regSource[4],instruction[25],instruction[25],1'b0);**

**MUX\_2 m14(regSource[3],instruction[24],instruction[24],1'b0);**

**MUX\_2 m15(regSource[2],instruction[23],instruction[23],1'b0);**

**MUX\_2 m16(regSource[1],instruction[22],instruction[22],1'b0);**

**MUX\_2 m17(regSource[0],instruction[21],instruction[21],1'b0);**

**//assign regTarget = instruction[20:16]; //regTarget**

**MUX\_2 m18(regTarget[4],instruction[20],instruction[20],1'b0);**

**MUX\_2 m19(regTarget[3],instruction[19],instruction[19],1'b0);**

**MUX\_2 m20(regTarget[2],instruction[18],instruction[18],1'b0);**

**MUX\_2 m21(regTarget[1],instruction[17],instruction[17],1'b0);**

**MUX\_2 m22(regTarget[0],instruction[16],instruction[16],1'b0);**

**//assign immed = instruction[15:0]; //immediate field**

**MUX\_2 m23(immed[15],instruction[15],instruction[15],1'b0);**

**MUX\_2 m24(immed[14],instruction[14],instruction[14],1'b0);**

**MUX\_2 m25(immed[13],instruction[13],instruction[13],1'b0);**

**MUX\_2 m26(immed[12],instruction[12],instruction[12],1'b0);**

**MUX\_2 m27(immed[11],instruction[11],instruction[11],1'b0);**

**MUX\_2 m28(immed[10],instruction[10],instruction[10],1'b0);**

**MUX\_2 m29(immed[9],instruction[9],instruction[9],1'b0);**

**MUX\_2 m30(immed[8],instruction[8],instruction[8],1'b0);**

**MUX\_2 m31(immed[7],instruction[7],instruction[7],1'b0);**

**MUX\_2 m32(immed[6],instruction[6],instruction[6],1'b0);**

**MUX\_2 m33(immed[5],instruction[5],instruction[5],1'b0);**

**MUX\_2 m34(immed[4],instruction[4],instruction[4],1'b0);**

**MUX\_2 m35(immed[3],instruction[3],instruction[3],1'b0);**

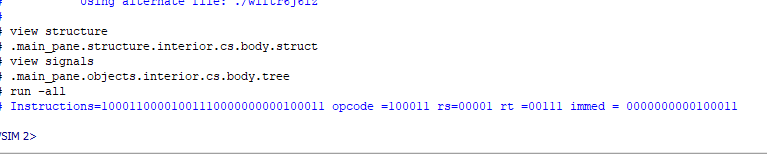
**MUX\_2 m36(immed[2],instruction[2],instruction[2],1'b0);**

**MUX\_2 m37(immed[1],instruction[1],instruction[1],1'b0);**

**MUX\_2 m38(immed[0],instruction[0],instruction[0],1'b0);**

**Endmodule**

**Result :**



Control unit was implemented according to op code

10 0000 – LB – 16 X 1 RES : 0

10 0100 – LBU – 16X1 – RES :4

10 0001- LHU – 16X1 – RES:1

10 0101 – LH – 16X1 – RES : 5

00 1111 – LUI – 16X1 – RES : 15

10 0011 – LW – 16X1 – RES : 3

10 1000 – SB – 16X1 – RES : 8

10 1001 – SH – 16X1 – RES : 9

10 1011 – SW – 16X1 – RES : 11

**module controlUnit(regWr,rd,regDest,memRead,memWr,opCode,functCode,instruction);**

**output regWr,regDest,memRead,memWr;**

**output [4:0] rd;**

**input [5:0] opCode,functCode;**

**input[31:0] instruction;**

**wire [1:0] select;**

**MUX\_2 m0(rd[4],instruction[20],instruction[20],1'b0);**

**MUX\_2 m1(rd[3],instruction[19],instruction[19],1'b0);**

**MUX\_2 m2(rd[2],instruction[18],instruction[18],1'b0);**

**MUX\_2 m3(rd[1],instruction[17],instruction[17],1'b0);**

**MUX\_2 m4(rd[0],instruction[16],instruction[16],1'b0);**

**MUX\_2 m5(regDest,1'b1,1'b0,1'b0);**

**MUX\_16 m6(memRead,opCode[3:0],1'b1,1'b1,1'b1,1'b1,1'b1,1'b1,1'b1,1'b1,1'b0,1'b0,1'b1,1'b0,1'b1,1'b1,1'b1,1'b1);**

**MUX\_16 m7(memWr,opCode[3:0],1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b1,1'b1,1'b0,1'b1,1'b0,1'b0,1'b0,1'b0);**

**MUX\_16 m8(regWr,opCode[3:0],1'b1,1'b1,1'b1,1'b1,1'b1,1'b1,1'b1,1'b1,1'b0,1'b0,1'b1,1'b0,1'b1,1'b1,1'b1,1'b1);**

Data.mem was defined for store instruction.

**module mips\_data\_mem (read\_data, mem\_address, write\_data, sig\_mem\_read, sig\_mem\_write,opCode,clk);**

**output [31:0] read\_data;**

**input [31:0] mem\_address;**

**input [31:0] write\_data;**

**input [5:0] opCode;**

**input sig\_mem\_read;**

**input sig\_mem\_write;**

**input clk;**

**reg [31:0] data\_mem [255:0];**

**reg [31:0] read\_data;**

**initial begin**

**$readmemb("C://altera//13.0//workspace//data.mem", data\_mem);**

**end**

**always @(posedge clk) begin**

**if (sig\_mem\_read) begin**

**read\_data = data\_mem[mem\_address];**

**//$display("MemAddresss = %32b, data=%32b",read\_data,data\_mem[mem\_address]);**

**end**

**if (sig\_mem\_write) begin**

**if (opCode == 6'b101000) begin // Store Byte**

**data\_mem[mem\_address][7:0] = write\_data[7:0];**

**$display("MemAddresss = %32b, data=%32b",mem\_address,data\_mem[mem\_address]);**

**end**

**else if (opCode == 6'b101001) begin // Store Halfword**

**data\_mem[mem\_address][15:0] = write\_data[15:0];**

**$display("MemAddresss = %32b, data=%32b",mem\_address,data\_mem[mem\_address]);**

**end**

**else if (opCode == 6'b101011) begin // Store Word**

**data\_mem[mem\_address] = write\_data[31:0];**

**$display("MemAddresss = %32b, data=%32b",mem\_address,data\_mem[mem\_address]);**

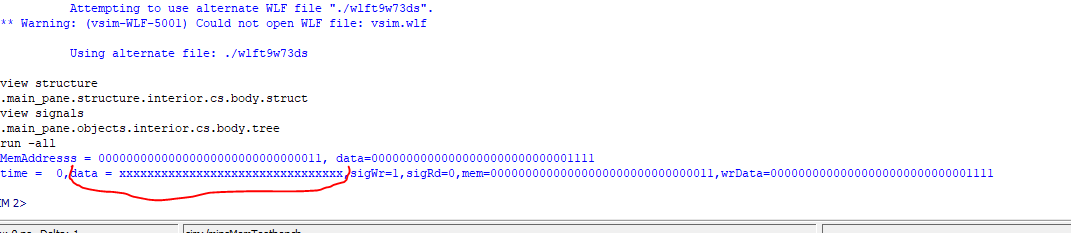
**end**

**$writememb("C://altera//13.0//workspace//data.mem", data\_mem);**

**end**

**end**

**endmodule**

**RESULTS :** 

**FOR SIGWR : 1 ,READDATA : X BUT WRDATA : 000000000000000000000000000001111**



**module mips\_registers**

**(**

**output reg [31:0] read\_data\_1, read\_data\_2,**

**input [31:0] write\_data,**

**input [4:0] read\_reg\_1, read\_reg\_2, write\_reg,**

**input signal\_reg\_write, clk**

**);**

**reg [31:0] registers [31:0];**

**//1 kere okunuyor.**

**//diğer işlemler register file üstünde yapılıyor.**

**initial begin**

**$readmemb("C://altera//13.0//workspace//registers.mem", registers);**

**end**

**always @(posedge clk)**

**begin**

**if (signal\_reg\_write == 1) begin**

**registers[write\_reg] = write\_data;**

**$writememb("C://altera//13.0//workspace//registers.mem", registers);**

**$display("rt = %5b, write\_data =%32b , signal\_reg\_write=%1b ",write\_reg,write\_data,signal\_reg\_write);**

**end**

**end**

**always @(negedge clk)**

**begin**

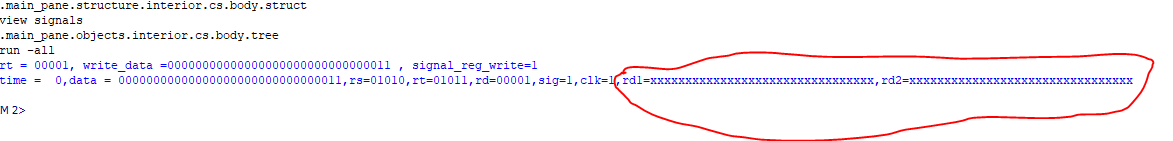
**read\_data\_1 = registers[read\_reg\_1];**

**read\_data\_2 = registers[read\_reg\_2];**

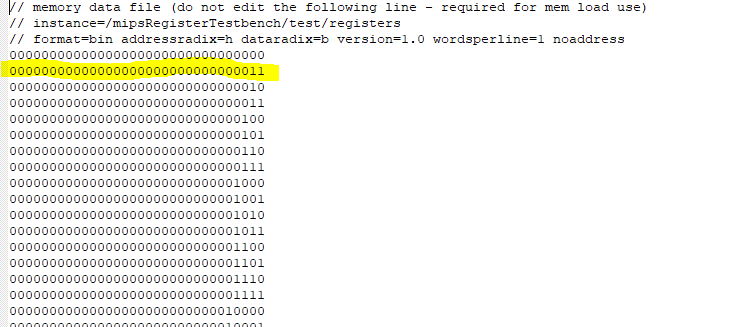
**end**

**endmodule**

**RESULT :**



**FOR SIGWR : 1 ,RD1 :X AND RD2 : X BUT WRDATA : 000000000000000000000000000000001**



MIPS CORE IS DESIGNED IF STORE INSTRUCTIONS COMES , WRITES DATA.MEM ,OR NOT LOAD INSTRUCTIONS COMES WRITES REGISTERS.MEM

module mips\_core(instruction,instr,result,clock,opCode,rs,rt,immed);

input clock;

input [31:0] instr;

input [31:0] instruction;

output [4:0] rs, rt;

output [15:0] immed;

output [31:0] result;

output [5:0] opCode;

wire [31:0] rsContent,rscontent,rtContent,rtcontent,rdcontent,memContent,memcontent,WriteReg;

wire [31:0] writeMem,memAddr;

wire [25:0] targetAddress;

wire [4:0] rd,shamt;

wire [5:0] regOpCode;

wire regWriteSignal;

wire memReadSig,memWriteSig;

reg zero;

wire [31:0] writeData1;

wire [31:0] writeData,tempWriteData,upperImmed;

reg [31:0] ALUResult;

wire [27:0] extendAddress;

wire [31:0] rsSigned,rtSigned;

wire [31:0] temp;

wire zeroBit;

wire[31:0] signExtend,writeLBData,writeLBUData,writeLHData,writeLHUData,writeLWData,writeLUIData;

wire[31:0] writeSData,write\_data;

wire [2:0] select\_bits;

wire carryin,carryout,less;

parser pars(rs,rt,immed,opCode,instruction);

MUX\_2 m0(zeroBit,1'b0,1'b0,1'b0);

MUX\_2 m1(select\_bits[2],1'b0,1'b0,1'b0);

MUX\_2 m2(select\_bits[1],1'b0,1'b0,1'b0);

MUX\_2 m3(select\_bits[0],1'b0,1'b0,1'b0);

assign signExtend = {{16{immed[15]}},immed};

assign upperImmed = {immed,16'b0};

assign rsContent = {{26{zeroBit}},rs};

assign rsSigned = {{26{rs[4]}},rs};

assign rtContent = {{26{zeroBit}},rt};

mips\_data\_mem memRead(memContent,rsContent, write\_data,1'b1,memWriteSig,opCode,1'b1);

controlUnit contSig(regWriteSignal,rd,destReg,memReadSig,memWriteSig,opCode,funcCode,instr);

ALU\_32 alu(address,rsContent,signExtend,1'b0,select\_bits,less,carryout);

equaltoTheOther eq(memAddr,address);

equaltoTheOther eq2(writeMem,rtContent);

equaltoTheOther eq3(writeSData,rtContent);

equaltoTheOther eq4(writeLWData,memContent);

assign writeLBUData = {24'b0,memContent[7:0]};

assign writeLBData = {{24{rsSigned[15]}},memContent[7:0]};

assign writeLHUData = {16'b0,memContent[15:0]};

assign writeLHData = {{16{rsSigned[15]}},memContent[15:0]};

equaltoTheOther eq6(writeLUIData,upperImmed);

select\_res selec(WriteReg,opCode,writeLHUData,writeLBData,writeData,writeLHData,writeLBUData,writeLUIData,writeLWData,writeSData);

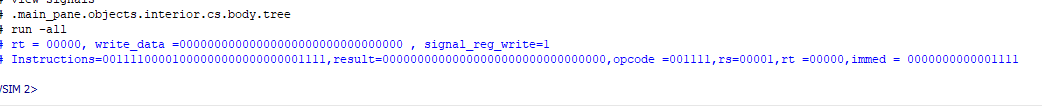
equaltoTheOther eq8(result,WriteReg);

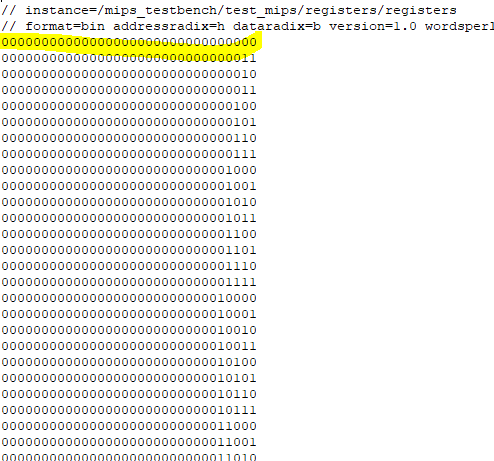
mips\_registers registers(rscontent,rdcontent,WriteReg,rs,rd,rt,regWriteSignal,1'b1);

mips\_data\_mem memWrite(temp,rtContent,WriteReg,memReadSig,memWriteSig,opCode,1'b1);

//mips\_instr\_mem instructionmem(instr, PC);

Endmodule





Ceren GÜMÜŞ

COMPUTER ORGANIZATION : CSE 331