Thomas Elliott Worrall

**tzw5376@psu.edu**

**Final Project**

**CMPEN 331**

Device: xc**7**k**70**tfbv**676**-**1**

Verilog Code:

**DataPath**

|  |
| --- |
| **`timescale** **1**ns / **1**ps  **module** Datapath(  clk,  pc\_internal,  dinstOut\_internal,  ewreg,  em2reg,  ewmem,  ealuc,  ealuimm,  edestReg,  eqa,  eqb,  eimm32,  wwreg,  wm2reg,  wdestReg,  wr,  wdo  );    **input** clk;  **output** **wire** [**31**:**0**] pc\_internal;  **output** **wire** [**31**:**0**] dinstOut\_internal;  **output** **wire** ewreg;  **output** **wire** em2reg;  **output** **wire** ewmem;  **output** **wire** [**3**:**0**] ealuc;  **output** **wire** ealuimm;  **output** **wire** [**4**:**0**] edestReg;  **output** **wire** [**31**:**0**] eqa;  **output** **wire** [**31**:**0**] eqb;  **output** **wire** [**31**:**0**] eimm32;  **output** **wire** wwreg;  **output** **wire** wm2reg;  **output** **wire** [**4**:**0**] wdestReg;  **output** **wire** [**31**:**0**] wr;  **output** **wire** [**31**:**0**] wdo;  **wire** [**31**:**0**] nextpc\_internal;  **wire** [**31**:**0**] instOut\_internal;  **wire** [**4**:**0**] rs;  **wire** [**4**:**0**] rt;  **wire** [**4**:**0**] rd;  **wire** [**31**:**0**] imm32\_internal;  **wire** [**31**:**0**] qb\_internal;  **wire** [**31**:**0**] qa\_internal;  **wire** [**4**:**0**] destReg\_internal;  **wire** aluimm\_internal;  **wire** [**3**:**0**] aluc\_internal;  **wire** wmem\_internal;  **wire** m2reg\_internal;  **wire** wreg\_internal;  **wire** regrt;  **wire** [**31**:**0**] b;  **wire** [**31**:**0**] r;  **wire** mwreg;  **wire** mm2reg;  **wire** mwmem;  **wire** [**4**:**0**] mdestReg;  **wire** [**31**:**0**] mr;  **wire** [**31**:**0**] mqb;  **wire** [**31**:**0**] mdo;  **wire** [**31**:**0**] wbData;  //begin new  **wire** [**1**:**0**] fwda\_signal;  **wire** [**1**:**0**] fwdb\_signal;  **wire** [**31**:**0**] fwdaOut;  **wire** [**31**:**0**] fwdbOut;  //end new  programCounter pcMod(clk, nextpc\_internal, pc\_internal);  pcAdder pcAddMod(pc\_internal, nextpc\_internal);  instructionMemory instMemMod(pc\_internal, instOut\_internal);  ifidPipelineRegister ifidRegMod(instOut\_internal, clk, dinstOut\_internal);  controlUnit controlMod(dinstOut\_internal, mdestReg, mm2reg, mwreg, edestReg, em2reg, ewreg, wreg\_internal, m2reg\_internal,  wmem\_internal, aluc\_internal, aluimm\_internal, regrt, fwdb\_signal, fwda\_signal);  **assign** rs = dinstOut\_internal[**25**:**21**];  **assign** rt = dinstOut\_internal[**20**:**16**];  **assign** rd = dinstOut\_internal[**15**:**11**];  regrtMultiplexer rgMultMod(dinstOut\_internal[**20**:**16**],  dinstOut\_internal[**15**:**11**], regrt, destReg\_internal);  fwda fwdaMod(fwda\_signal, qa\_internal, r, mr, mdo, fwdaOut);  fwdb fwdbMod(fwdb\_signal, qb\_internal, r, mr, mdo, fwdbOut);  registerFile regfileMod(dinstOut\_internal[**25**:**21**], dinstOut\_internal[**20**:**16**],  wdestReg, wbData, wwreg, clk, qa\_internal, qb\_internal); //NEW EDITS  immediateExtender ieMod(dinstOut\_internal[**15**:**0**], imm32\_internal);  idexePipelineRegister idexePipeRegMod(wreg\_internal, m2reg\_internal,  wmem\_internal, aluc\_internal, aluimm\_internal, destReg\_internal,  fwdaOut, fwdbOut, imm32\_internal, clk, ewreg, em2reg, ewmem, ealuc,  ealuimm, edestReg, eqa, eqb, eimm32);  aluMultMod aluMultiplexer(eqb, eimm32, ealuimm, b);  aLUMod alu(eqa, b, ealuc, r);  exememRegMod exememPipelineRegister(ewreg, em2reg, ewmem, edestReg, r, eqb,  clk, mwreg, mm2reg, mwmem, mdestReg, mr, mqb);  dataMemMod dataMemory(mr, mqb, mwmem, clk, mdo);  memwbReg memwbPipelineRegister(mwreg, mm2reg, mdestReg, mr, mdo, clk, wwreg,  wm2reg, wdestReg, wr, wdo);  wbMux wbMuxModule(wr, wdo, wm2reg, wbData);      **endmodule** |

**Program Counter Module**

|  |
| --- |
| **`timescale** **1**ns / **1**ps  **module** programCounter(  **input** clk,  **input** [**31**:**0**] nextPC,  **output** **reg** [**31**:**0**] pc  );  **initial**  **begin**  pc = **32'd100**;  **end**    **always** @(**posedge** clk) **begin**  pc = nextPC;  **end**  **endmodule** |

**PC Adder Module**

|  |
| --- |
| **`timescale** **1**ns / **1**ps  **module** pcAdder(  **input** [**31**:**0**] pc,  **output** **reg** [**31**:**0**] nextPC  );  **localparam** [**2**:**0**] a = **3'b100**;  **always** @(\*) **begin**  nextPC = pc + a;  **end**  **endmodule** |

**Instruction Memory Module**

|  |
| --- |
| **`timescale** **1**ns / **1**ps  **module** instructionMemory(  **input** [**31**:**0**] pc,  **output** **reg** [**31**:**0**] instOut  );  **reg** [**31**:**0**] memory [**0**:**63**];  **initial**  **begin**  memory[**0**] = **32'hA00000AA**;  memory[**1**] = **32'h10000011**;  memory[**2**] = **32'h20000022**;  memory[**3**] = **32'h30000033**;  memory[**4**] = **32'h40000044**;  memory[**5**] = **32'h50000055**;  memory[**6**] = **32'h60000066**;  memory[**7**] = **32'h70000077**;  memory[**8**] = **32'h80000088**;  memory[**9**] = **32'h90000099**;  memory[**25**] = {  **6'b000000**, // r-type (add $3 $1 $2)  **5'b00001**, // rs - $1  **5'b00010**, //rt - $2  **5'b00011**, //rd - $3  **5'b00000**,  **6'b100000** //add    };  memory[**26**] = {  **6'b000000**, // r-type (sub $4 $9 $3)  **5'b01001**, // rs - $9  **5'b00011**, //rt - $3  **5'b00100**, //rd - $4  **5'b00000**,  **6'b100010** //sub    };  memory[**27**] = {  **6'b000000**, // r-type (or $5 $3 $9)  **5'b00011**, // rs - $3  **5'b01001**, //rt - $9  **5'b00101**, //rd - $5  **5'b00000**,  **6'b100101** //or    };  memory[**28**] = {  **6'b000000**, // r-type (xor $6 $3 $9)  **5'b00011**, // rs - $3  **5'b01001**, //rt - $9  **5'b00110**, //rd - $6  **5'b00000**,  **6'b100110** //xor    };  memory[**29**] = {  **6'b000000**, // r-type (and $7 $3 $9)  **5'b00011**, // rs - $3  **5'b01001**, //rt - $9  **5'b00111**, //rd - $7  **5'b00000**,  **6'b100100** //and    };    **end**  **always** @(\*) **begin**  instOut = memory[pc[**7**:**2**]];  **end**  **endmodule** |

**IFID Pipeline Register Module**

|  |
| --- |
| **`timescale** **1**ns / **1**ps  **module** ifidPipelineRegister(  **input** [**31**:**0**] instOut,  **input** clk,  **output** **reg** [**31**:**0**] dinstOut  );    **always** @(**posedge** clk) **begin**  dinstOut = instOut;  **end**  **endmodule** |

**Control Unit Module**

|  |
| --- |
| **`timescale** **1**ns / **1**ps  **module** controlUnit(  **input** [**31**:**0**] Instruction,  **input** [**4**:**0**] mrn, //mDestReg wire (exemem register) NEW  **input** mm2reg, //NEW  **input** mwreg, //NEW  **input** [**4**:**0**] ern, //eDestReg wire (idexe register) NEW  **input** em2reg, //NEW  **input** ewreg, //NEW  **output** **reg** wreg,  **output** **reg** m2reg,  **output** **reg** wmem,  **output** **reg** [**3**:**0**] aluc,  **output** **reg** aluimm,  **output** **reg** regrt,  **output** **reg** [**1**:**0**] fwdb, //NEW  **output** **reg** [**1**:**0**] fwda //NEW  );  **reg** [**5**:**0**] op;  **reg** [**5**:**0**] func;  **reg** [**5**:**0**] rs;  **reg** [**5**:**0**] rt;  **reg** [**5**:**0**] rd;  **always** @(Instruction) **begin**  op = Instruction[**31**:**26**];  func = Instruction[**5**:**0**];  rs = Instruction[**25**:**21**];  rt = Instruction[**20**:**16**];  rd = Instruction[**15**:**11**];  **end**  **always** @(\*) **begin**  fwda = **2'b00**;  fwdb = **2'b00**;  **if** (rs == ern) **begin**  fwda = **2'b01**;  **end** **else** **if** (rt == mrn) **begin**  fwdb = **2'b10**;  **end** **else** **if** (rt == ern) **begin**  fwdb = **2'b01**;  **end** **else** **if** (rs == mrn) **begin**  fwda = **2'b10**;  **end**  **case** (op)  **6'b000000**:  **begin**  **case** (func)  **6'b100000**: **begin** // ADD instruction  wreg = **1'b1**;  m2reg = **1'b0**;  wmem = **1'b0**;  aluc = **4'b0010**;  aluimm = **1'b0**;  regrt = **1'b0**;  **end**  **6'b100010**: **begin** // SUB instruction (Review)  wreg = **1'b1**;  m2reg = **1'b0**;  wmem = **1'b0**;  aluc = **4'b0110**;  aluimm = **1'b0**;  regrt = **1'b0**;  **end**  **6'b100101**: **begin** // OR instruction (Review)  wreg = **1'b1**;  m2reg = **1'b0**;  wmem = **1'b0**;  aluc = **4'b0001**;  aluimm = **1'b0**;  regrt = **1'b0**;  **end**  **6'b100110**: **begin** // XOR instruction (Review)  wreg = **1'b1**;  m2reg = **1'b0**;  wmem = **1'b0**;  aluc = **4'b1010**;  aluimm = **1'b0**;  regrt = **1'b0**;  **end**  **6'b100100**: **begin** // AND instruction (Review)  wreg = **1'b1**;  m2reg = **1'b0**;  wmem = **1'b0**;  aluc = **4'b0000**;  aluimm = **1'b0**;  regrt = **1'b0**;  **end**  **endcase**  **end**  **6'b100011**: **begin** // lw -- load word  wreg = **1'b1**;  m2reg = **1'b1**;  wmem = **1'b0**;  aluc = **4'b0010**;  aluimm = **1'b1**;  regrt = **1'b1**;  **end**  **endcase**  **end**  **endmodule** |

**Reg RT Multiplexer Module**

|  |
| --- |
| **`timescale** **1**ns / **1**ps  **module** regrtMultiplexer(  **input** [**4**:**0**] rt,  **input** [**4**:**0**] rd,  **input** regrt,  **output** **reg** [**4**:**0**] destReg  );  **always** @(\*) **begin**  **if** (regrt == **0**) **begin**  destReg = rd;  **end** **else** **begin**  destReg = rt;  **end**  **end**  **endmodule** |

**Register File Module**

|  |
| --- |
| **`timescale** **1**ns / **1**ps  **module** registerFile(  **input** [**4**:**0**] rs,  **input** [**4**:**0**] rt,  **input** [**4**:**0**] wdestReg,  **input** [**31**:**0**] wbData,  **input** wwreg,  **input** clk,  **output** **reg** [**31**:**0**] qa,  **output** **reg** [**31**:**0**] qb  );  **reg** [**31**:**0**] registers [**31**:**0**];  **integer** x;  **initial** **begin**  //BEGIN New  registers[**0**] = **32'h00000000**;  registers[**1**] = **32'hA00000AA**;  registers[**2**] = **32'h10000011**;  registers[**3**] = **32'h20000022**;  registers[**4**] = **32'h30000033**;  registers[**5**] = **32'h40000044**;  registers[**6**] = **32'h50000055**;  registers[**7**] = **32'h60000066**;  registers[**8**] = **32'h70000077**;  registers[**9**] = **32'h80000088**;  registers[**10**] = **32'h90000099**;  //END New  **for** (x = **11**; x<**32**; x = x+**1**) **begin**  registers[x] = **32'd0**;  **end**  **end**  **always** @(\*) **begin**  qa = registers[rs];  qb = registers[rt];  **end**  **always** @(**negedge** clk) **begin**  **if** (wwreg == **1**) **begin**  registers[wdestReg] = wbData;  **end**  **end**  **endmodule** |

**Immediate Extender Module**

|  |
| --- |
| **`timescale** **1**ns / **1**ps  **module** immediateExtender(  **input** [**15**:**0**] imm,  **output** **reg** [**31**:**0**] imm32  );  **always** @(\*) **begin**  imm32 = {{ **16**{imm[**15**]}}, imm[**15**:**0**]};  **end**  **endmodule** |

**ID EXE Pipeline Register Module**

|  |
| --- |
| **`timescale** **1**ns / **1**ps  **module** idexePipelineRegister(  **input** wreg,  **input** m2reg,  **input** wmem,  **input** [**3**:**0**] aluc,  **input** aluimm,  **input** [**4**:**0**] destReg,  **input** [**31**:**0**] qa,  **input** [**31**:**0**] qb,  **input** [**31**:**0**] imm32,  **input** clk,  **output** **reg** ewreg,  **output** **reg** em2reg,  **output** **reg** ewmem,  **output** **reg** [**3**:**0**] ealuc,  **output** **reg** ealuimm,  **output** **reg** [**4**:**0**] edestReg,  **output** **reg** [**31**:**0**] eqa,  **output** **reg** [**31**:**0**] eqb,  **output** **reg** [**31**:**0**] eimm32  );  **always** @(**posedge** clk) **begin**  ewreg = wreg;  em2reg = m2reg;  ewmem = wmem;  ealuc = aluc;  ealuimm = aluimm;  edestReg = destReg;  eqa = qa;  eqb = qb;  eimm32 = imm32;  **end**  **endmodule** |

**ALU Multiplexer Module**

|  |
| --- |
| **`timescale** **1**ns / **1**ps  **module** aluMultMod(  **input** [**31**:**0**] eqb,  **input** [**31**:**0**] eimm32,  **input** ealuimm,  **output** **reg** [**31**:**0**] b  );    **always** @(\*) **begin**  **if** (ealuimm == **0**) **begin**  b = eqb;  **end** **else** **begin**  b = eimm32;  **end**  **end**  **endmodule** |

**ALU Module**

|  |
| --- |
| **`timescale** **1**ns / **1**ps  **module** aLUMod(  **input** [**31**:**0**] eqa,  **input** [**31**:**0**] b,  **input** [**3**:**0**] ealuc,  **output** **reg** [**31**:**0**] r  );  **always** @(\*) **begin**  **case** (ealuc)  **4'b0000** : r = eqa & b; //AND  **4'b0001** : r = eqa | b; //OR  **4'b0010** : r = eqa + b; //ADD  **4'b0110** : r = eqa - b; //SUB  **4'b0111** : r = eqa < b ? **1** : **0**; //set-less-than  **4'b1100** : r = ~(eqa | b); //NOR  **4'b1010** : r = eqa ^ b; //XOR  **endcase**  **end**  **endmodule** |

**EX MEM Pipeline Register Module**

|  |
| --- |
| **`timescale** **1**ns / **1**ps  **module** exememRegMod(  **input** ewreg,  **input** em2reg,  **input** ewmem,  **input** [**4**:**0**] edestReg,  **input** [**31**:**0**] r,  **input** [**31**:**0**] eqb,  **input** clk,  **output** **reg** mwreg,  **output** **reg** mm2reg,  **output** **reg** mwmem,  **output** **reg** [**4**:**0**] mdestReg,  **output** **reg** [**31**:**0**] mr,  **output** **reg** [**31**:**0**] mqb  );  **always** @(**posedge** clk) **begin**  mwreg = ewreg;  mm2reg = em2reg;  mwmem = ewmem;  mdestReg = edestReg;  mr = r;  mqb = eqb;  **end**  **endmodule** |

**Data Memory Module**

|  |
| --- |
| **`timescale** **1**ns / **1**ps  **module** dataMemMod(  **input** [**31**:**0**] mr,  **input** [**31**:**0**] mqb,  **input** mwmem,  **input** clk,  **output** **reg** [**31**:**0**] mdo  );    **reg** [**31**:**0**] memory [**0**:**63**]; //double check  **initial**  **begin**  memory[**0**] = **32'hA00000AA**;  memory[**1**] = **32'h10000011**;  memory[**2**] = **32'h20000022**;  memory[**3**] = **32'h30000033**;  memory[**4**] = **32'h40000044**;  memory[**5**] = **32'h50000055**;  memory[**6**] = **32'h60000066**;  memory[**7**] = **32'h70000077**;  memory[**8**] = **32'h80000088**;  memory[**9**] = **32'h90000099**;  **end**  **always** @(\*) **begin**  mdo = memory[mr[**7**:**2**]];  **end**  **always** @(**negedge** clk) **begin**  **if** (mwmem == **1**) **begin**  memory[mr[**7**:**2**]] = mqb;  **end**  **end**  **endmodule** |

**Mem WB Pipeline Register Module**

|  |
| --- |
| **`timescale** **1**ns / **1**ps  **module** memwbReg(  **input** mwreg,  **input** mm2reg,  **input** [**4**:**0**] mdestReg,  **input** [**31**:**0**] mr,  **input** [**31**:**0**] mdo,  **input** clk,  **output** **reg** wwreg,  **output** **reg** wm2reg,  **output** **reg** [**4**:**0**] wdestReg,  **output** **reg** [**31**:**0**] wr,  **output** **reg** [**31**:**0**] wdo  );  **always** @(**posedge** clk) **begin**  wwreg = mwreg;  wm2reg = mm2reg;  wdestReg = mdestReg;  wr = mr;  wdo = mdo;  **end**  **endmodule** |

**Write Back Multiplexer Module**

|  |
| --- |
| **`timescale** **1**ns / **1**ps  **module** wbMux(  **input** [**31**:**0**] wr,  **input** [**31**:**0**] wdo,  **input** wm2reg,  **output** **reg** [**31**:**0**] wbData  );  **always** @(\*) **begin**  **if** (wm2reg == **0**) **begin**  wbData = wr;  **end** **else** **begin**  wbData = wdo;  **end**  **end**  **endmodule** |

**FWDA Module**

|  |
| --- |
| **`timescale** **1**ns / **1**ps  **module** fwda(  **input** [**1**:**0**] fwda\_signal,  **input** [**31**:**0**] qa,  **input** [**31**:**0**] alur,  **input** [**31**:**0**] emr,  **input** [**31**:**0**] mdo,  **output** **reg** [**31**:**0**] a  );  **always** @(\*) **begin**  **case** (fwda\_signal)  **2'b00**: //No hazards  **begin**  a = qa;  **end**  **2'b01**:  **begin**  a = alur;  **end**  **2'b10**:  **begin**  a = emr;  **end**  **2'b11**:  **begin**  a = mdo;  **end**    **endcase**  **end**  **endmodule** |

**FWDB Module**

|  |
| --- |
| **`timescale** **1**ns / **1**ps  **module** fwdb(  **input** [**1**:**0**] fwdb\_signal,  **input** [**31**:**0**] qb,  **input** [**31**:**0**] alur,  **input** [**31**:**0**] emr,  **input** [**31**:**0**] mdo,  **output** **reg** [**31**:**0**] b  );    **always** @(\*) **begin**  **case** (fwdb\_signal)  **2'b00**: //No hazards  **begin**  b = qb;  **end**  **2'b01**:  **begin**  b = alur;  **end**  **2'b10**:  **begin**  b = emr;  **end**  **2'b11**:  **begin**  b = mdo;  **end**    **endcase**  **end**  **endmodule** |

**Testbench Module**

|  |
| --- |
| **`timescale** **1**ns / **1**ps  **module** testbench();  **reg** clk;  **wire** [**31**:**0**] pc\_tb;  **wire** [**31**:**0**] dinstOut;  **wire** ewreg;  **wire** em2reg;  **wire** ewmem;  **wire** ealuimm;  **wire** [**3**:**0**] ealuc;  **wire** [**4**:**0**] edestReg;  **wire** [**31**:**0**] eqa;  **wire** [**31**:**0**] eqb;  **wire** [**31**:**0**] eimm32;  **wire** wwreg;  **wire** wm2reg;  **wire** [**4**:**0**] wdestReg;  **wire** [**31**:**0**] wr;  **wire** [**31**:**0**] wdo;  **initial** **begin**  clk = **0**;  **end**  Datapath **DP**( .clk (clk), .pc\_internal (pc\_tb), .dinstOut\_internal (dinstOut), .ewreg (ewreg), .em2reg (em2reg), .ewmem (ewmem),  .ealuc (ealuc), .ealuimm (ealuimm), .edestReg (edestReg), .eqa (eqa), .eqb (eqb), .eimm32 (eimm32), .wwreg (wwreg), .wm2reg (wm2reg),  .wdestReg (wdestReg), .wr (wr), .wdo (wdo));  **always** **begin**  #**5**  clk = ~clk;  **end**  **endmodule** |

**Waveform**

**A picture containing timeline

Description automatically generated**

**Design Schematic**

Diagram

Description automatically generated

**IO Planning**

**A picture containing graphical user interface

Description automatically generated**

**Floor Planning**

**Schematic

Description automatically generated**