ECE 212 – Digital Circuits II
Extra Credit: Modified MIPS Single-Cycle Processor
Irwin Frimpong
4/25/20

Introduction

In this assignment, the task was to modify the MIPS Single Cycle Processor to perform the lui (Load Upper Immediate) function. The lui I-Type MIPS function takes an rt register, an rs register which is empty, and a 16-bit immediate value and loads the first 16 bits of the rt register with the specified immediate value. This called for a minor modification of the datapath of the processor which would be discussed in detail in the following section.

Implementation/ Results

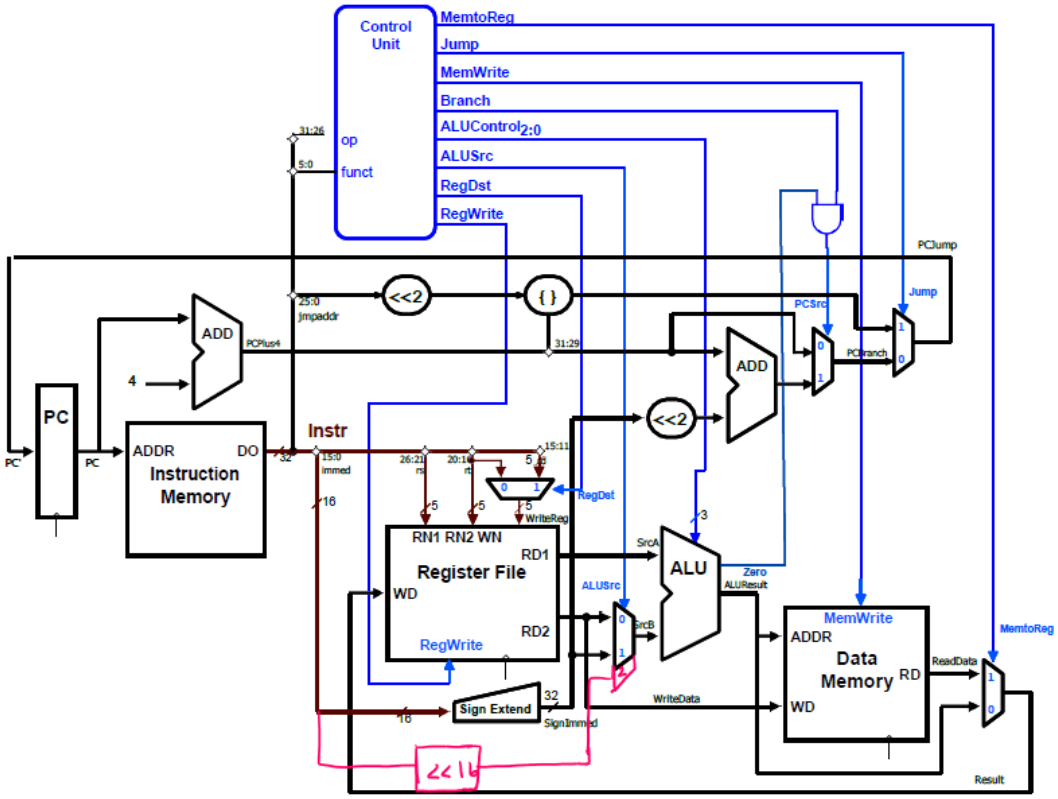

Figure 1 – Single-cycle MIPS processor

Figure 1: *Depicted above is the modification made to the datapath. Since we required the 16-bit immediate value to be loaded in the MSB bits of the register rt, it was required that we perform a leftward shift of 16 on the 32-bit instruction to only hold the immediate value followed by zeros. Since this is a new value not being used by any other MIPS I-Type instructions, it called for a modification the multiplexer that feeds into ScrB of the ALU to support three selections. This ultimately required us to modify the ALUSrc control signal to be two bits instead of it previously being a one-bit value.*

ECE 212 – Digital Circuits II
Extra Credit: Modified MIPS Single-Cycle Processor
Irwin Frimpong
4/25/20

Extended functionality. Main Decoder:

Instruction	$Op_{5:0}$	RegWrite	RegDst	AluSrc	Branch	MemWrite	MemtoReg	$ALUOp_{1:0}$	Jump	BNE Flag
R-type	000000	1	1	0	0	0	0	10	0	0
lw	100011	1	0	01	0	0	1	00	0	0
sw	101011	0	X	01	0	1	X	00	0	0
beq	000100	0	X	00	1	0	X	01	0	0
addi	001000	1	0	01	0	0	0	00	0	0
j	000010	0	X	X	X	0	X	XX	1	0
ori	001101	1	0	01	0	0	0	11	0	0
bne	000101	0	0	00	1	0	0	01	0	1
lui	001111	1	0	10	0	0	0	00	0	0

[1:0]

Table 1 -Main Decoder Control Signals

Code Listings

MIPS Module

```
module mips(input  logic        clk, reset,
            output logic [31:0] pc,
            input  logic [31:0] instr,
            output logic        memwrite,
            output logic [31:0] aluout, writedata,
            input  logic [31:0] readdata);

  logic memtoreg;
  logic [1:0] alusrc;
  logic regdst, regwrite, jump, pcsrc, zero, bneFlag;
  logic [2:0] alucontrol;

  controller c(instr[31:26], instr[5:0], zero,
               memtoreg, memwrite, pcsrc,
               alusrc, regdst, regwrite, jump,
               alucontrol, bneFlag);
  datapath dp(clk, reset, memtoreg, pcsrc,
              alusrc, regdst, regwrite, jump,
              alucontrol,
              zero, pc, instr,
              aluout, writedata, readdata, bneFlag);
endmodule
```

ECE 212 – Digital Circuits II
Extra Credit: Modified MIPS Single-Cycle Processor
Irwin Frimpong
4/25/20

Controller Module

```
module controller(input  logic [5:0] op, funct,
                  input  logic       zero,
                  output logic       memtoreg, memwrite,
                  output logic       pcsrc,
                  output logic[1:0]  alusrc,
                  output logic       regdst, regwrite,
                  output logic       jump,
                  output logic [2:0] alucontrol,
                  output logic       bneFlag);
  logic [1:0] aluop;
  logic       branch;
   maindec md(op, memtoreg, memwrite, branch,alusrc, regdst, regwrite, jump, aluop, bneFlag);
   aludec  ad(funct, aluop, alucontrol);
   assign pcsrc = branch & zero;
endmodule
```

MainDecoder Module

```
module maindec(input  logic [5:0] op,
               output logic       memtoreg, memwrite,
               output logic       branch,
               output logic [1:0]  alusrc,
               output logic       regdst, regwrite,
               output logic       jump,
               output logic [1:0] aluop,
               output logic       bneFlag);

  logic [10:0] controls;
  assign {regwrite, regdst, alusrc, branch, memwrite,
          memtoreg, jump, aluop, bneFlag} = controls;

  always_comb
    case(op)
      6'b000000: controls <= 11'b11000000100; // RTYPE
      6'b100011: controls <= 11'b10010010000; // LW
      6'b101011: controls <= 11'b00010100000; // SW
      6'b000100: controls <= 11'b00001000010; // BEQ
      6'b001000: controls <= 11'b10010000000; // ADDI
      6'b000010: controls <= 11'b00000001000; // J
      6'b001101: controls <= 11'b10010000110; // ORI
      6'b000101: controls <= 11'b00001000011; // BNE
      6'b001111: controls <= 11'b10100000000; // LUI

      default:   controls <= 11'bxxxxxxxxxxx; // illegal op
    endcase
endmodule
```

Datapath Module

```
module datapath(input  logic        clk, reset,

                input  logic        memtoreg, pcsrc,
                input  logic [1:0]   alusrc,
                input logic          regdst,
                input  logic        regwrite, jump,
                input  logic [2:0]  alucontrol,
                output logic        zero,
                output logic [31:0] pc,
                input  logic [31:0] instr,
                output logic [31:0] aluout, writedata,
                input  logic [31:0] readdata,
                input  logic        bneFlag);

  logic [4:0]  writereg;
```

ECE 212 – Digital Circuits II
Extra Credit: Modified MIPS Single-Cycle Processor
Irwin Frimpong
4/25/20

```
  logic [31:0] pcnext, pcnextbr, pcplus4, pcbranch;
  logic [31:0] signimm, signimmsh;
  logic [31:0] srca, srcb;
  logic [31:0] result;

  // next PC logic
  flopr #(32) pcreg(clk, reset, pcnext, pc);
  adder       pcadd1(pc, 32'b100, pcplus4);
  sl2         immsh(signimm, signimmsh);
  adder       pcadd2(pcplus4, signimmsh, pcbranch);
  mux2 #(32)  pcbrmux(pcplus4, pcbranch, pcsrc, pcnextbr);
  mux2 #(32)  pcmux(pcnextbr, {pcplus4[31:28],
                    instr[25:0], 2'b00}, jump, pcnext);
  // register file logic
  regfile     rf(clk, regwrite, instr[25:21], instr[20:16],
                 writereg, result, srca, writedata);
  mux2 #(5)   wrmux(instr[20:16], instr[15:11],regdst, writereg);
  mux2 #(32)  resmux(aluout, readdata, memtoreg, result);
  signext     se(instr[15:0], signimm);

  // ALU logic
   mux3 #(32) srcbmux(writedata, signimm,instr << 16, alusrc, srcb);
   alu  alu(srca, srcb, alucontrol, aluout, zero, bneFlag);
endmodule
```

Mux3 Module

```
module mux3 #(parameter WIDTH = 8)
             (input  logic [WIDTH-1:0] d0, d1, d2,
              input  logic [1:0]       s,
              output logic [WIDTH-1:0] y);

  assign #1 y = s[1] ? d2 : (s[0] ? d1 : d0);
endmodule
```

Test Program

MIPS Machine Language	**Hexadecimal**
ori $s0,$0,0x0002	34100002
addi $s1,$0,0x0005	20110005
bne $s0,$s1,0x000c	16110000
lui $s2,$s0,0x0002	3c000002
add $s1,$s1,$0	02208820
slt $t0,$s1,$s0	0230402a
lw $t1,0($s0)	8e090000
sw $s1,32($0)	ac110020

Table 2 – MIPS Machine Language with hexadecimal equivalent obtained via MIPS converter

ECE 212 – Digital Circuits II
Extra Credit: Modified MIPS Single-Cycle Processor
Irwin Frimpong
4/25/20

Vivado Simulation Results

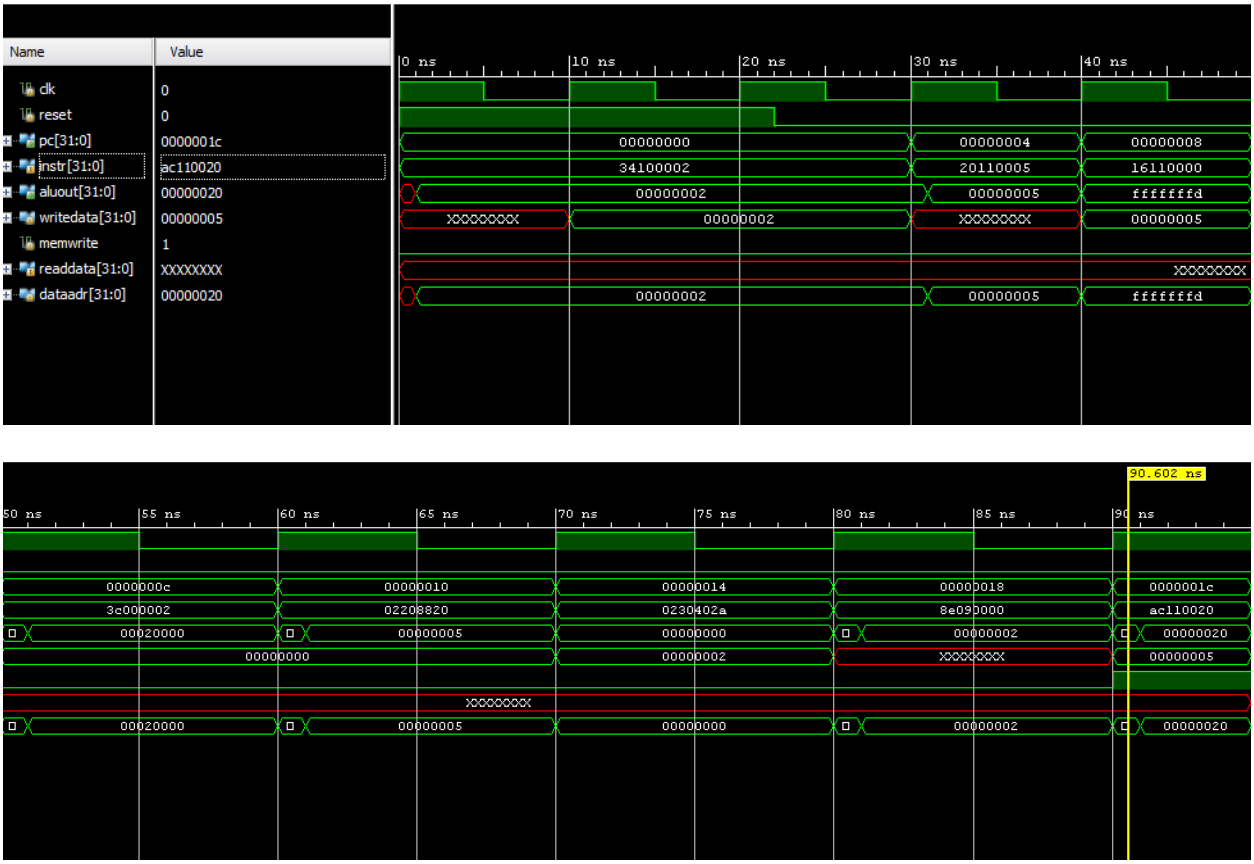

Figure 2 – Simulation Results With Corresponding Signals

Conclusion

This lab was insightful in reinforcing my understanding of the MIPS Single-Cycle Processor and the corresponding datapath associated with said processor. Since the LUI instruction required a slight modification to the datapath, specifically for the ALUSrc control signal, it was imperative for the bit width not only to be changed in the maindecoder module but every other submodule that also depended on such signal. This required me to subtly reverse engineer the datapath and trace wherever the signal was used to ensure that it was also modified in those modules.