Preliminary Design Report
Lab 3

Section 3

Abdullah Al Wali

21402793

Machine Code Disassembly

Location	Machine Instruction	Assembly Language
0x00	0x20020005	addi $2 , $0, 5
0x04	0x2003000c	addi $3, $0, 12
0x08	0x2067fff7	addi $7, $3, -9
0x0c	0x00e22025	or $4, $7, $2
0x10	0x00642824	and $5, $3, $4
0x14	0x00a42820	add $5, $5, $4
0x18	0x10a7000a	beq $5, $7, 10
0x1c	0x0064202a	slt $4, $3, $4
0x20	0x10800001	beq $4, $0, 1
0x24	0x20050000	addi $5, $0, 0
0x28	0x00e2202a	slt $4, $7, $2
0x2c	0x00853820	add $7, $4, $5
0x30	0x00e23822	sub $7, $7, 2
0x34	0xac670044	sw $7, 68($3)
0x38	0x8c020050	lw $2, 80($0)
0x3c	0x08000011	j 0x44
0x40	0x20020001	addi $2, $0, 1
0x44	0xac020054	sw $2, 84($0)
0x48	0x08000012	j 0x48

RTL Expressions for new Instructions

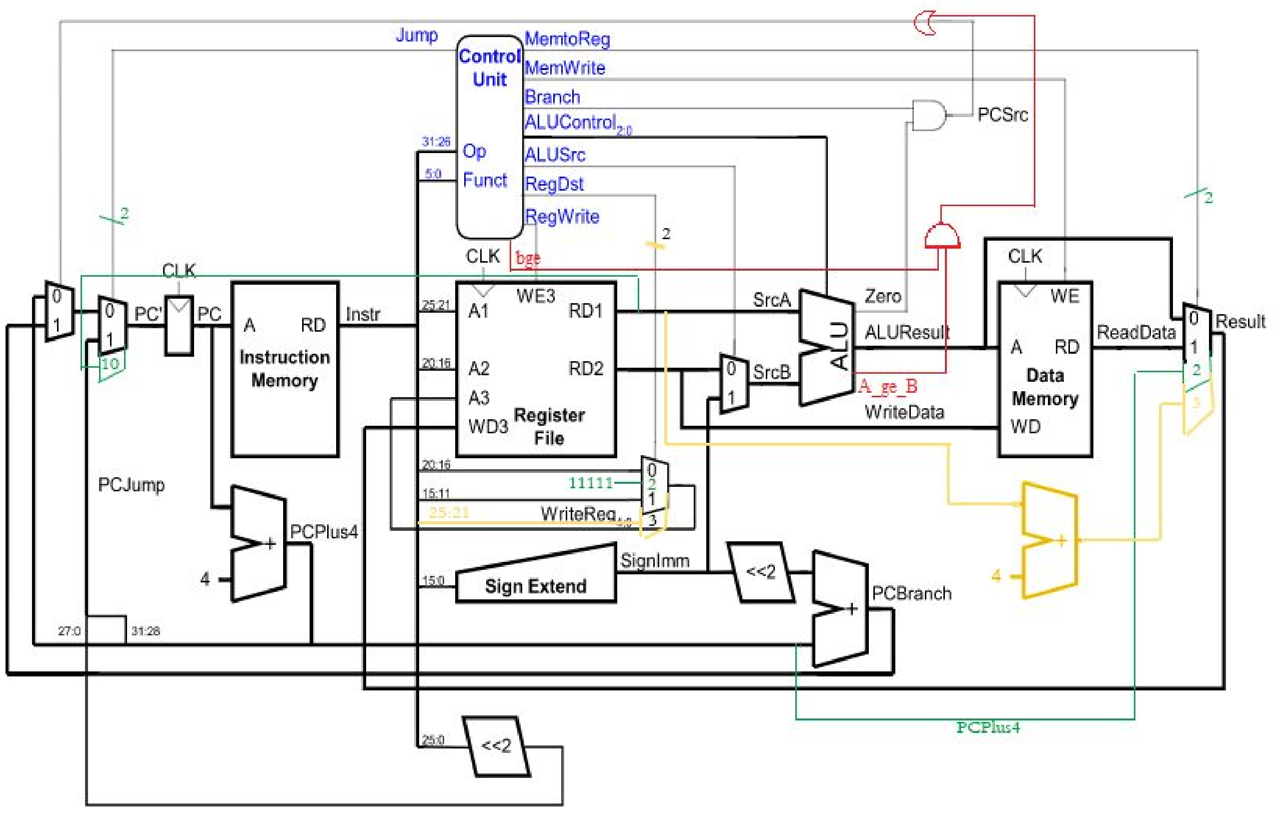
bge:
IM[PC]
if (RF[rs] >= RF[rt]) PC ← PC + 4 + signExtend(imm26) x 4
else PC ← PC + 4

jalr:
IM[PC]
RF[31] ← PC + 4
PC ← RF[rs]

swapRM:
IM[PC]
RF[rt] ← DM[RF[rs]+SignExt(imm)]
DM[RF[rs]+SignExt(imm)] ← RF[rt]
PC ← PC + 4

push:
IM[PC]
DM[RF[rs] +SignExt(imm)] ← RF[rt]
RF[rs] ← RF[rs] + 4
PC ← PC + 4

Changes in Datapath



Changes in Control Unit

Instruction	$Op_{5:0}$	Reg Write	Reg $Dst_{1:0}$	Alu Src	Branch	Mem Write	Mem $toReg_{1:0}$	ALU $Op_{1:0}$	Jump $_{1:0}$	bge
R-type	000000	1	01	0	0	0	00	1x	00	0
lw	100011	1	00	1	0	0	01	00	00	0
sw	101011	0	XX	1	0	1	XX	00	00	0
beq	000100	0	XX	0	1	0	XX	01	00	0
j	000010	0	XX	X	X	0	XX	XX	01	0
addi	001000	1	00	1	0	0	00	00	00	0
bge	011100	0	XX	0	0	0	XX	01	00	1
jalr	011101	1	10	X	0	0	10	XX	10	0
swapRM	011110	1	00	1	0	1	01	00	00	0
push	011111	1	11	0	0	1	11	00	00	0

MIPS Test Program

```
Begin:
addi $2, $0, 5
push $2                 #push           #20020005
swapRM $4, 0($29)       #swapRM
jalr $2
add $5, $5, $4          #add
and $5, $3, $4          #and
sub $7, $7, 2           #sub
or $4, $2, $7           #or
slt $4, $3, $4          #slt
sw $7, 68($3)           #sw

lw $2, 80($0)           #lw
beq $4, $0, End         #beq
j Begin                 #j
End:


function:
add $5, $5, $4
sub $7, $7, 2
j $ra
```

Verilog Code

Verilog Modules to be changed:

- datapath
- maindec
- alu
- controller

Modules to be added:

- mux3
- mux4

---

datapath:

```
module datapath(input         clk, reset,
               input      [1:0]   memtoreg,
                input             pcsrc,
               input        alusrc,
                input  [1:0]    regdst,
               input        regwrite,
                input [1:0]      jump,
               input  [2:0]  alucontrol,
               output       zero,
                output A_ge_B,
               output [31:0] pc,
               input  [31:0] instr,
               output [31:0] aluout, writedata,
               input  [31:0] readdata);

  wire [4:0]  writereg;
  wire [31:0] pcnext, pcnextbr, pcplus4, pcbranch;
  wire [31:0] signimm, signimmsh;
  wire [31:0] srca, srcb;
  wire [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)  branchmux(pcplus4, pcbranch, pcsrc,
                    pcnextbr);
  mux3 #(32)  jumpmux(pcnextbr, {pcplus4[31:28],
                  instr[25:0], 2'b00}, srca,
                  jump, pcnext);

  // register file logic
  regfile     rf(clk, regwrite, instr[25:21],
               instr[20:16], writereg,
               result, srca, writedata);
  adder              srcaAdder( srca, 32'b100, srcaplus4);
  mux4 #(5)   w_addrmux(instr[20:16], instr[15:11], 5'b11111, instr[25:21],
                  regdst, writereg);
  mux4 #(32)  w_datamux(aluout, readdata, pcplus4, srcaplus4,
                   memtoreg, result);
  signext     se(instr[15:0], signimm);

  // ALU logic
  mux2 #(32)  srcbmux(writedata, signimm, alusrc,
                   srcb);
  alu        alu(srca, srcb, alucontrol,
               aluout, zero, A_ge_B);
endmodule
```

maindec:

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

  reg [12:0] controls;

  assign {regwrite, regdst, alusrc, branch,
         memwrite, memtoreg, aluop, jump,bge } = controls;

  always @(*)
    case(op)
      6'b000000: controls <= 13'b101000001x000; // R-type
      6'b100011: controls <= 13'b1001000100000; // LW
      6'b101011: controls <= 13'b0xx101xx00000; // SW
      6'b000100: controls <= 13'b0xx010xx01000; // BEQ
      6'b001000: controls <= 13'b1001000000000; // ADDI
      6'b000010: controls <= 13'b1001000000000; // J
      6’b011100: controls <= 0xx000xx01001; //bge
       6’b011101: controls <=110x0010xx100 ; //jalr
       6’b011110: controls <= 1001010100000; //swapRM
       6’b011111: controls <= 1110011100000; //push
      default:   controls <= 9'bxxxxxxxxx; // ???
    endcase
endmodule
```

---

alu:

```
module alu (
    input [31:0] A,
    input [31:0] B,
    input [2:0] S,
    output reg [31:0]  Y,
    output reg zero,
    output A_gt_B
    );

              always @ (*) begin
                          case (S)
                                      3'b000: Y <= (A && B); //AND
                                      3'b001: Y <= (A || B); //OR
                                      3'b010: Y <= A + B; //add
                                      3'b100: Y <= A + B; // Add
                                      3'b101: Y <= A - B; //subtract
                                      3'b111: Y <= if ( A < B) Y <= 32b’1; else Y = 32’b0;
                                      default: Y <= 31’b0; //default
                          endcase
             end
             always @ (*)
                          if (Y== 32’b0) zero = 1; else zero = 0;
             assign A_gt_B = ~Y[31];
endmodule
```

Controller:

```
module controller(input  [5:0] op, funct,
                  input       zero,
                   input A_gt_b,
                  output [1:0]  memtoreg,
                   output      memwrite,
                  output       pcsrc, alusrc,
                  output [1:0]       regdst,
                  output            regwrite,
                  output [1:0]      jump,
                  output [2:0] alucontrol);

    wire [1:0] aluop;
    wire       branch, bge;

    maindec md (op, regwrite, regdst, alusrc, branch,
              memwrite, memtoreg, aluop, jump, bge);
    aludec  ad (funct, aluop, alucontrol);

    assign pcsrc = (bge & A_gt_b) || (branch & zero);
endmodule
```

---

mux3:

```
module mux3 (
input [31:0] d0, d1, d2,
input [1:0] s,
output [31:0] y);
assign y = s[1] ? d2 : (s[0] ? d1 : d0);
endmodule
```

---

mux4:

```
module mux4 (
input [31:0] d0, d1, d2, d3
input [1:0] s,
output [31:0] y);
assign y = s[1] ? (s[0]? d3: d2) : (s[0] ? d1 : d0);
endmodule
```