CS224 Lab No: 4

Section No: 1

Halil Arda Özongun

22202709

PART-1

b)

Address (Hex)	Machine Code (Hex)	Assembly Language Instruction
0x00000000	0x20020005	addi $2, $0, 5
0x00000004	0x2003000c	addi $3, $0, 12
0x00000008	0x2067fff7	addi $7, $3, -9
0x0000000C	0x00e22025	or $4, $7, $2
0x00000010	0x00642824	and $5, $3, $4
0x00000014	0x00a42820	add $5, $5, $4
0x00000018	0x10a7000a	beq $5, $7, 10
0x0000001C	0x0064202a	slt $4, $3, $4
0x00000020	0x10800001	beq $4, $0, 1
0x00000024	0x20050000	addi $5, $0, 0
0x00000028	0x00e2202a	slt $4, $7, $2
0x0000002C	0x00853820	add $7, $4, $5
0x00000030	0x00e23822	sub $7, $7, $4
0x00000034	0xac670044	sw $7, 68($3)
0x00000038	0x8c020050	lw $2, 80($0)
0x0000003C	0x08000011	j 0x00000044
0x00000040	0x20020001	addi $2, $0, 1
0x00000044	0xac020054	sw $2, 84($0)
0x00000048	0x08000012	j 0x00000048

c)

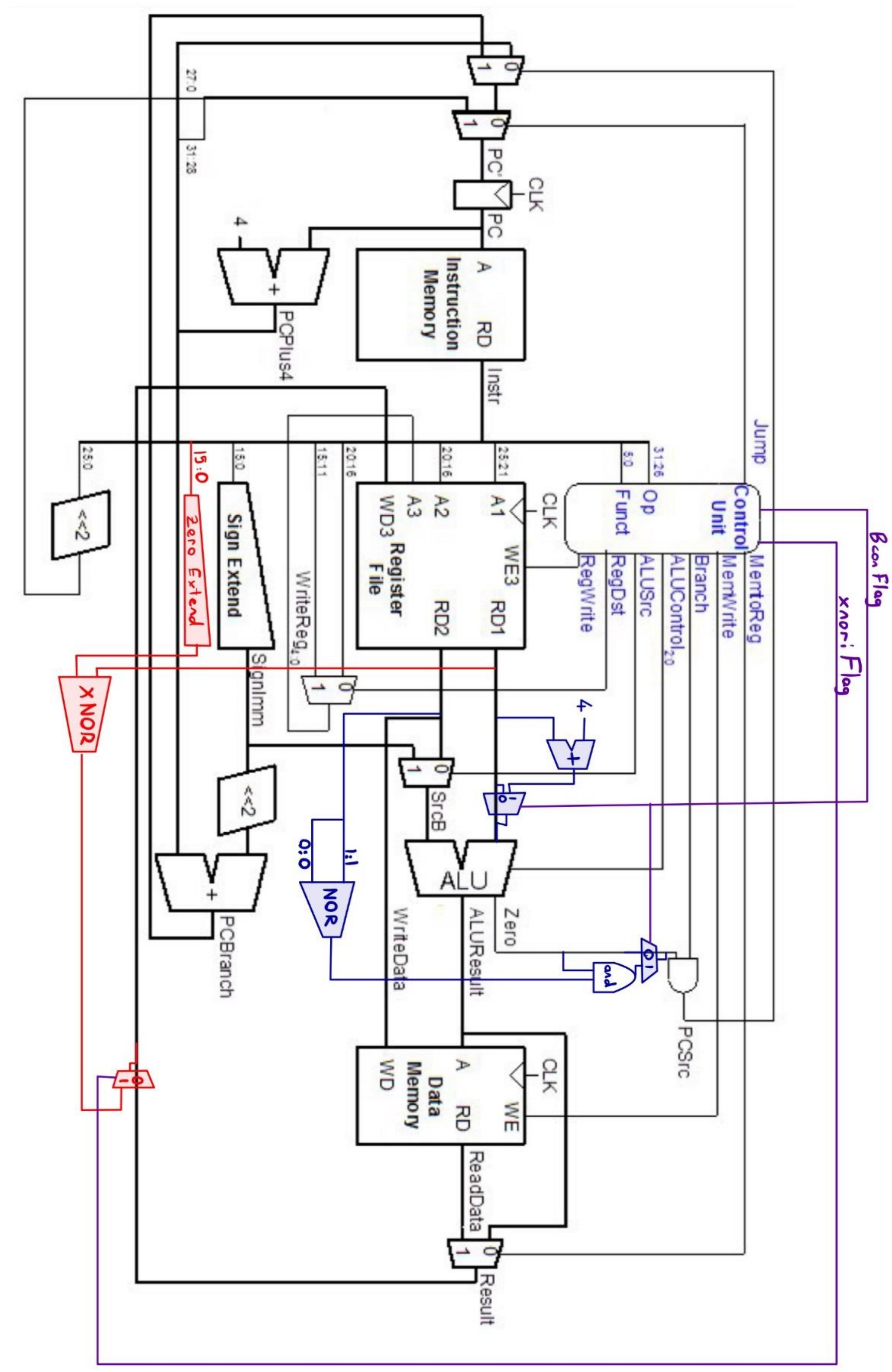
Bcon Instruction (I-type):

```
IM[PC]
if ((RF[rt] – RF[rs] == 4) (~(((RF[rt] >> 1) & 0x1) | (RF[rt] & 0x1)) == 1))
    PC ← PC + 4 + (SignExt(immed) << 2)
else
    PC ← PC + 4
```

Xnori Instruction (I-type):

```
IM[PC]
RF[rt] ← ~ (RF[rs] ⊕ ZeroExtImm)
PC ← PC + 4
```

d)



CLK
PC'
PC
Instruction Memory
A
RD
Instr
4
+
PCPlus4
27:0
31:28
Jump
Control Unit
Op
Funct
31:26
5:0
MemtoReg
MemWrite
Branch
ALUControl2:0
ALUSrc
RegDst
RegWrite
25:21
20:16
15:11
15:0
25:0
<<2
A1
A2
A3
WD3
WE3
RD1
RD2
Register File
WriteReg4:0
Sign Extend
SignImm
15:0
Zero Extend
XNOR
SrcB
ALU
Zero
ALUResult
WriteData
NOR
1:1
0:0
and
PCBranch
PCSrc
Data Memory
WE
RD
WD
ReadData
Result
Bcon Flag
xnori Flag
0
1

e)

Instruction	Opcode	Reg Write	RegDst	ALU Src	Branch	MemWrite	MemToReg	ALUOp	Jump	Bcon Flag	Xnori Flag
R-type	000000	1	1	0	0	0	0	10	0	0	0
lw	100011	1	0	1	0	0	1	00	0	0	0
sw	101011	0	X	1	0	1	X	00	0	0	0
beq	000100	0	X	0	1	0	X	01	0	0	0
addi	001000	1	0	1	0	0	0	00	0	0	0
j	000010	0	X	X	X	0	X	XX	1	0	0
bcon	110101	0	X	0	1	0	X	01	0	1	0
xnori	110100	1	0	X	0	0	X	XX	0	0	1

f)

```
.text

	addi $t0, $zero, 12
	addi $t1, $zero, 15
	bcon $t0, $t1, 3	# sholdn’t take branch
	addi $t2, $zero, 19
	bcon $t2, $t1, 3	# sholdn’t take branch
	addi $t0, $zero, 4
	addi $t1, $zero, 8
	bcon $t0, $t1, 2	# must take branch
	addi $t0, $zero, 4
	addi $t0, $zero, 1111
	addi $t0, $zero, a32e	# branch comes here
	xnori $t2, $t0, 0x0f0f
	xnori $t2, $t0, 0xffff
	xnori $t2, $t0, 0xa32e
```

g)

New Modules:

1. xnor_gate

```
module xnor_gate (input logic [31:0] a, input logic [31:0] b, output logic [31:0] y);
            assign y = ~(a ^ b);
endmodule
```

2. zeroext

```
module zeroext (input  logic[15:0] a,
             output logic[31:0] y);
      assign y = {16'b0, a};
endmodule
```

Updated Modules:

1. MIPS

```
`timescale 1ns / 1ps

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, pcsrc, zero, alusrc, regdst, regwrite, jump;
  logic       XnorFlag, BconFlag;
  logic [2:0]  alucontrol;

  controller c (instr[31:26], instr[5:0], zero, memtoreg, memwrite, pcsrc,
                  alusrc, regdst, regwrite, jump, alucontrol,
                  XnorFlag, BconFlag);

  datapath dp (clk, reset, memtoreg, pcsrc, alusrc, regdst, regwrite, jump,
                   alucontrol, zero, pc, instr, aluout, writedata, readdata,
                   XnorFlag, BconFlag);

endmodule
```

2. Maindec

```
`timescale 1ns / 1ps

module maindec (input logic[5:0] op,
                output logic memtoreg, memwrite, branch,
                output logic alusrc, regdst, regwrite, jump,
                output logic[1:0] aluop,
                output logic XnorFlag, BconFlag
                 );
  logic [10:0] controls;

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

 always_comb
   case(op)
     6'b000000: controls <= 11'b11000010000; // R-type
     6'b100011: controls <= 11'b10100100000; // LW
     6'b101011: controls <= 11'b00101000000; // SW
     6'b000100: controls <= 11'b00010001000; // BEQ
     6'b001000: controls <= 11'b10100000000; // ADDI
     6'b000010: controls <= 11'b00000000100; // J
     6'b110101: controls <= 11'b00010001001; // bcon
     6'b110100: controls <= 11'b10000000010; // xnori
     default:   controls <= 11'bxxxxxxxxxxx; // illegal op
   endcase
endmodule
```

3. Datapath

```
`timescale 1ns / 1ps

module datapath (input  logic clk, reset, memtoreg, pcsrc, alusrc, 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 XnorFlag, BconFlag

              );


 logic [4:0]  writereg;
 logic [31:0] pcnext, pcnextbr, pcplus4, pcbranch;
 logic [31:0] signimm, signimmsh, srca, srcb, result;

 logic [31:0] srca_output_of_regFile, result_before_mux;
 logic [31:0] added4, lasttwobit;
 logic nZero, aluzero;
 logic [31:0] zeroExtended, xnorOutput;


 // 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_output_of_regFile, writedata);

  mux2 #(5)    wrmux (instr[20:16], instr[15:11], regdst, writereg);
  mux2 #(32)  resmux (aluout, readdata, memtoreg, result_before_mux);
  signext        se (instr[15:0], signimm);

 // ALU logic
  mux2 #(32)  srcbmux (writedata, signimm, alusrc, srcb);
  alu         alu (srca, srcb, alucontrol, aluout, aluzero);

  // xnori logic
  zeroext zeroext1(instr[15:0], zeroExtended);

  xnor_gate xnor_gate1(srca_output_of_regFile, zeroExtended, xnorOutput);

  mux2 #(32) xnorMux(result_before_mux, xnorOutput, XnorFlag, result);


  // bcon logic


  adder add4 (srca_output_of_regFile, 32'd4, added4);

  mux2 #(32)  bconmux (srca_output_of_regFile, added4, BconFlag, srca);


  assign lasttwobit = ~(writedata[1] | writedata[0]); // nor last two bits


  assign nZero = lasttwobit & aluzero;

  mux2 #(1)  zeromux (aluzero, nZero, BconFlag, zero);



endmodule
```

4. Regfile

```
`timescale 1ns / 1ps

module regfile (input    logic clk, we3,
                input    logic[4:0]  ra1, ra2, wa3,
                input    logic[31:0] wd3,
                output   logic[31:0] rd1, rd2);

  logic [31:0] rf [31:0];

// fill rf with zeros
  initial begin
    for (int i = 0; i < 32; i++) begin
      rf[i] = 32'h0;
    end
  end

  always_ff@(posedge clk)
     if (we3)
         rf [wa3] <= wd3;

  assign rd1 = (ra1 != 0) ? rf [ra1] : 0;
  assign rd2 = (ra2 != 0) ? rf[ ra2] : 0;
```

```
endmodule
```

5. Dmem

```
`timescale 1ns / 1ps

module dmem (input  logic        clk, we,
             input  logic[31:0]  a, wd,
             output logic[31:0]  rd);

  logic  [31:0] RAM[63:0];

  // initialize ram with zeros
  initial begin
    for (int i = 0; i < 64; i++) begin
      RAM[i] = 32'h0;
    end
  end

  assign rd = RAM[a[31:2]];    // word-aligned  read (for lw)

  always_ff @(posedge clk)
   if (we)
     RAM[a[31:2]] <= wd;      // word-aligned write (for sw)

endmodule
```