CS 224

LAB 4

SEC 1

İPEK ÖZTAŞ

22003250

PART 1:

a)

Location	Machine Instruction (hex)	Assembly Language
8'h00	32'h2014fff6	addi $s4, $zero, 0XFF6
8'h04	32'h20090007	addi $t1, $zero, 0X007
8'h08	32'h22820003	addi $v0, $s4, 0X0003
8'h0c	32'h01342025	or $a0, $t1, $s4
8'h10	32'h00822824	and $a1, $a0, $v0
8'h14	32'h00a42820	add $a1, $a1, $a0
8'h18	32'h1045003d	beq $v0, $a1, 0X003D
8'h1c	32'h0054202a	slt $a0, $v0, $s4
8'h20	32'h10040001	beq $zero, $a0, 0X0001
8'h24	32'h00002820	add $a1, $zero, $zero
8'h28	32'h0289202a	slt $a0, $s4, $t1
8'h2c	32'h00853820	add $a3, $a0, $a1
8'h30	32'h00e23822	sub $a3, $a3, $v0
8'h34	32'hac470057	sw $a3, 0X0057, $v0
8'h38	32'h8c020050	lw $v0, 0X0050, $zero
8'h3c	32'h08000011	j 0X0000011
8'h40	32'h20020001	addi $v0, $zero, 0X0001
8'h44	32'h2282005a	addi $v0, $s4, 0X005A
8'h48	32'h08000012	j 0X0000012

Table1: Assembly instructions

d)

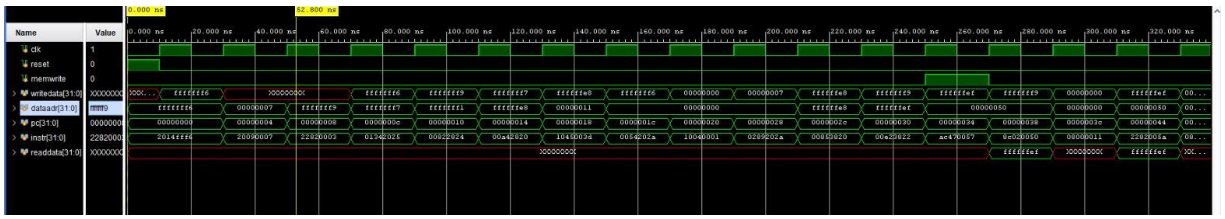

Figure 1: Waveform

e)

i) In an R-type instruction what does writedata correspond to?

In R-type, it corresponds to the rd value. Its data is written to the register.

ii) Why is writedata undefined for some of the early instructions in the program?

The instructions that are undefined for writedata are all I-type functions, since writedata isn't used therefore not necessarily defined.

iii) In which instructions memwrite becomes 1?

store word (sw) This instruction stores the contents of a register to memory.

iv) Why is dataaddr 0xfffffffe8 when PC is 0x14?

The dataaddr is set to 0xfffffffe8 for the instruction add $a1, $a1, $a0 because this is the address where the value of register $a1 is stored in memory. In this instruction, the contents of register $a1 are added to the contents of register $a0, and the result is stored back in register $a1. However, this instruction also has the side effect of storing the previous value of register $a1 back to memory. The imem module is designed to simulate a simple memory system, where each memory address corresponds to a single byte. In this case, the contents of register $a1 are stored in memory starting at address 0xfffffffe8. When this instruction is executed, the value of $a1 is first read from memory at address 0xfffffffe8 and then updated with the new value. Finally, the updated value of $a1 is stored back to memory at the same address.

v) In the example program, when is the output of readdata defined and why?

In load word (lw) instruction, 'memread' is set to 1, which signals the 'dmem' module to read 4-byte data from the memory location specified by the 'dataaddr' and write it to 'readdate'.

f)

```
module alu(input  logic [31:0] a, b,
           input  logic [2:0]  alucont,
           output logic [31:0] result,
           output logic zero);


    always_comb
        case(alucont)
            3'b010: result = a + b;
            3'b110: result = a - b;
            3'b000: result = a & b;
            3'b001: result = a | b;
            3’b011: result = a^b;
            3'b111: result = (a < b) ? 1 : 0;
            default: result = {32{1'bx}};
        endcase

    assign zero = (result == 0) ? 1'b1 : 1'b0;
endmodule
```

PART 2:

a) lui: This I-type instruction loads the value in the imm field to the most significant 2 bytes of RF[rt]. The two least significant bytes are set to 0. Example: lui $t0, 0x1001.

lui: *IM[PC]*
RF[rt] ← *immed16 || 16'b0*
PC ← *PC + 4*

xori: This I-type instruction zero-extends the given immediate then bitwise XORs it with RF[rs]. The result is written into RF[rt]. Example: xori $t0, $t1, 0x0040.

xori: *IM[PC]*
RF[rt] ← *RF[rs] xor ZeroExt(immed)*
PC ← *PC + 4*

b) Datapath

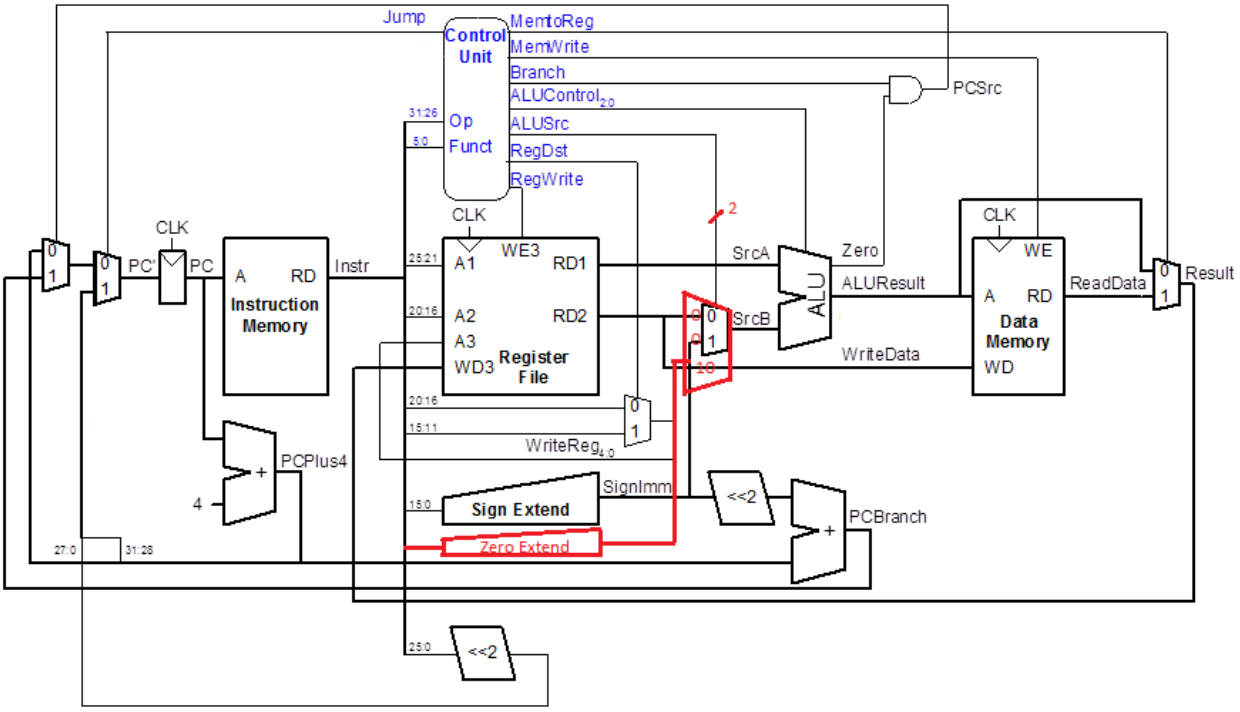

Figure 2: Modified Datapath

c) Main Decoder Table

Instruc-tion	**$Op_{5:0}$**	**Reg Write**	**Reg Dst**	***Alu $Src_{1:0}$***	**Branch**	**Mem Write**	**Mem toReg**	**ALU $Op_{2:0}$**	**Jump**
R-type	000000	1	1	*00*	0	0	0	010	0
lw	100011	1	0	*01*	0	0	1	000	0
sw	101011	0	X	*01*	0	1	X	000	0
beq	000100	0	X	*00*	1	0	X	001	0
addi	001000	1	0	*01*	0	0	0	000	0
j	000010	0	X	*XX*	X	0	X	XXX	1
lui	*110000*	*1*	*0*	*01*	*0*	*0*	*0*	*100*	*0*
xori	*001110*	*1*	*0*	*10*	*0*	*0*	*0*	*011*	*0*

Table 2: Main Decoder

ALU Decoder Table

ALUOp	Funct	ALUControl
000	x	010 (add)
001	x	110 (sub)
010	100000 (add)	010 (add)
010	100010 (sub)	110 (sub)
010	100100 (and)	000 (and)
010	100101 (or)	001 (or)
010	101010 (slt)	111 (set less than)
011	*x*	*101 (xor)*
100	*x*	*011(shift left 16)*

Table 3: ALU Decoder

Xori implementation requires Zero Extend (16 bit) which has the Instruction[15:0] as the input. The output goes to ALU SrcB. Therefore, the mux is extended. ALUSrc 10 indicates that the ALU input comes from the Zero Extend. xor operation is added to ALUControl Unit (101) with the ALUOp 011.

Lui implementation would not require any changes to the data path and would feed the zero extended (Extend=0) of Instruction[15:0] to the second input of the ALU (ALUSrc=1), set the

ALUOp bits=100 (011 is used for xori) for a logical 16-bit left shift operation at the ALU specified by the ALU control, then set MemtoReg=0, set RegWrite=1 and set RegDst=0 to write the ALU result to Reg Rt.

Since we add two new operations to the ALU (xor and shift left 16-bit) and there are already 5 operations, ALUOp requires 3 bits in order to indicate total of 7 operations.