CS 224 Computer Architecture

Preliminary Lab 4

Kutay Tire

22001787

03.04.2022

Part 1.)

a.)

Location	Machine Instruction	Assembly Language Equivalent
0x00	0x20020005	addi \$v0, \$0, 0x5
0x04	0x2003000c	addi \$v1, \$0, 0xC
0x08	0x2067fff7	addi \$a3, \$v1, 0xFFF7
0х0с	0x00e22025	or \$a0, \$a3, \$v0
0x10	0x00642824	and \$a1, \$v1, \$a0
0x14	0x00a42820	add \$a1, \$a1, \$a0
0x18	0x10a7000a	beq \$a1, \$a3, 0xA
0x1c	0x0064202a	slt \$a0, \$v1, \$a0
0x20	0x10800001	beq \$a0, \$0, 0x1
0x24	0x20050000	addi \$a1, \$0, 0x0
0x28	0x00e2202a	slt \$a0, \$a3, \$v0
0x2c	0x00853820	add \$a3, \$a0, \$a1
0x30	0x00e23822	sub \$a3, \$a3, \$v0
0x34	0xac670044	sw \$a3, 0x44(\$v1)
0x38	0x8c020050	lw \$v0, 0x50(\$0)
0х3с	0x08000011	j 0x11
0x40	0x20020001	addi \$v0, \$0, 0x1
0x44	0xac020054	sw \$v0, 0x54(\$0)
0x48	0x08000012	j 0x12

d.)

Name	Value	80.000 ns	90.000 ns	100.000 ns	110.000 ns	120.000 ns	130.000 ns	140.000 ns	150.000 ns	160.000 ns	170.000 ns	180.000 ns	190.000 ns	200.000 ns	210.00
¹ clk	0														
□ reset	1														
> 😻 writed31:0	XXXXXXX		00000005			XXXX	xxxx		0000	0005		0000	0007		0
> 😻 dataa1:0]	00000005		00000005		0000	000c	0000	0003	0000	0007	0000	0004	0000	000ь	0
> 💆 pc[31:0]	00000000		00000000		0000	0004	0000	0008	0000	000c	0000	0010	0000	0014	0
> 💆 instr[31:0]	20020005		20020005		2003	000c	2067	E££7	00e2	2025	0064	2824	00a4	2820	1
> 💆 readd31:0	XXXXXX							xxxxxxxxx							
memwrite	0														

Name	Value									290.000 ns				330.000 ns	340.00
₩ clk	0														
₩ reset	1														
> W writed31:0	XXXXXXX	0000	0003	0000	0007	0000	0000	00000	1005	0000	000ъ	0000	0005	00000007	
> W dataa1:0]	00000005	0000	0008	X	0000	0000		00000	0001	0000	000c	0000	0007	00000050	
> W pc[31:0]	00000000	0000	0018	0000	001c	0000	0020	00000	0028	0000	002c	0000	0030	00000034	
> !! instr[31:0]	20020005	10a7	000a	0064	202a	1080	0001	00e22	202a	0085	3820	00e2	3822	ac670044	
> W readd31:0								XXXXXXXX							
[™] memwrite	0														

Name	Value	340.000 ns	350.000 ns	360.000 ns	370.000 ns	380.000 ns	390.000 ns	400.000 ns	410.000 ns	420.000 ns	430.000 ns	440.000 ns	450.000 ns	460.000 ns	
₩ clk	0														
₩ reset	1														
> W writed31:0	XXXXXXX	00000007	0000	0005	0000	0000	0000	0007	X		00	000000			
> 💆 dataa1:0]	00000005		00000050		0000	0000	0000	0054			00	000000			
> W pc[31:0]	00000000	00000034	0000	0038	0000	003c	0000	0044	X		00	000048			
> 🕨 instr[31:0]	20020005	ac670044	8:02	0050	0800	0011	ac02	0054	08000012			000012			
> W readd31:0	XXXXXXX	XXXXXXXXXX	0000	0007				XXXXXXXXX							
memwrite	0														
				l , , , ,											

e.)

- writedata corresponds to the output of the RD2 that comes from RF [rt]. However, in R type instructions, the data memory is not used as memwrite is always zero. So, it does not have a function for R type instructions.
- **II)** writedata read the value in the rt register. However, in the early instructions, rt register isn't used so it does not contain any value. That's why it appears as undefined.
- readdata is undefined for most of the times because the above instructions mostly do not use data memory. So, the R type instructions use the result of ALU instead of the data from the memory. As it can be seen, readdata isn't zero when the instruction is 0x8c020050 which is a "lw" instruction. This time, the value from the data memory is needed. However, other than that, the other instructions given in the instruction memory does not use data memory.

- **IV)** dataadr corresponds to the output of the ALU. However, it is useless in R type instructions as they do not use data memory. However, it is useful for lw instructions.
- **V)** memwrite becomes 1 when there is need to write to memory. So, in sw instructions it becomes 1.

```
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'b110: result = a << b;
                               3'b111: result = (a < b)? 1: 0;
                              default: result = {32{1'bx}};
                       endcase
               assign zero = (result == 0)? 1'b1: 1'b0;
```

Part 2.)

endmodule

a.)

For subi

Im [PC]

RF [rt] <-- RF [rs] - SignExt(immed)

PC <— PC + 4

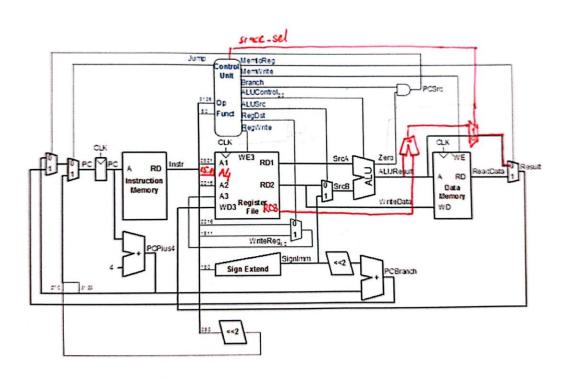
For sracc

Im [PC]

 $RF[rd] \leftarrow RF[rs] \Rightarrow RF[rt] + RF[rd]$

PC <-- PC + 4

b.)



For the addition of datapath, subi instruction does not need any components as it can
be implemented within the ALU. However, for sracc, more components were needed.
First of all, another port to read from rd was needed. Then, with the help of an adder,
the result of ALU and the result of RF [rd] was added. Finally, this was sent to another
multiplexer which has a new select signal from controller.

c.)

Main Decoder

Instruction	Opcode	RegWrite	RegDst	ALUSrc	Branch	MemWrite	MemToReg	ALUOp	Jump	Sracc_sel
R-type	000000	1	1	0	0	0	0	10	0	0
lw	100011	1	0	1	0	0	1	00	0	0
SW	101011	0	X	1	0	1	X	00	0	0
beq	000100	0	X	0	1	0	X	01	0	0
addi	001000	1	0	1	0	0	0	00	0	0
j	000010	0	X	X	X	0	X	XX	1	0
subi	010100	1	0	1	0	0	0	01	0	0
sracc	000000	1	1	0	0	0	0	10	0	1

• For subi instruction, I chose the opcode to be 010100 as it does not conflict with any other opcodes. Also, for sracc, I used 000000 since it can be classified as an R type instruction.

ALU Decoder

ALUOp	Funct	ALUControl
00	X	010 (add)
01	X	110 (subtract)
1X	100000 (add)	010 (add)
1X	100010 (sub)	110 (subtract)
1X	100100 (and)	000 (and)
1X	100101 (or)	001 (or)
1X	101010 (slt)	111 (set less than)
1X	000011(sra)	100(shift right arithmetic)

• I used shift right arithmetic which is 000011 because sracc operation needs shift right. Also, I assigned it to 100 as it was not used.