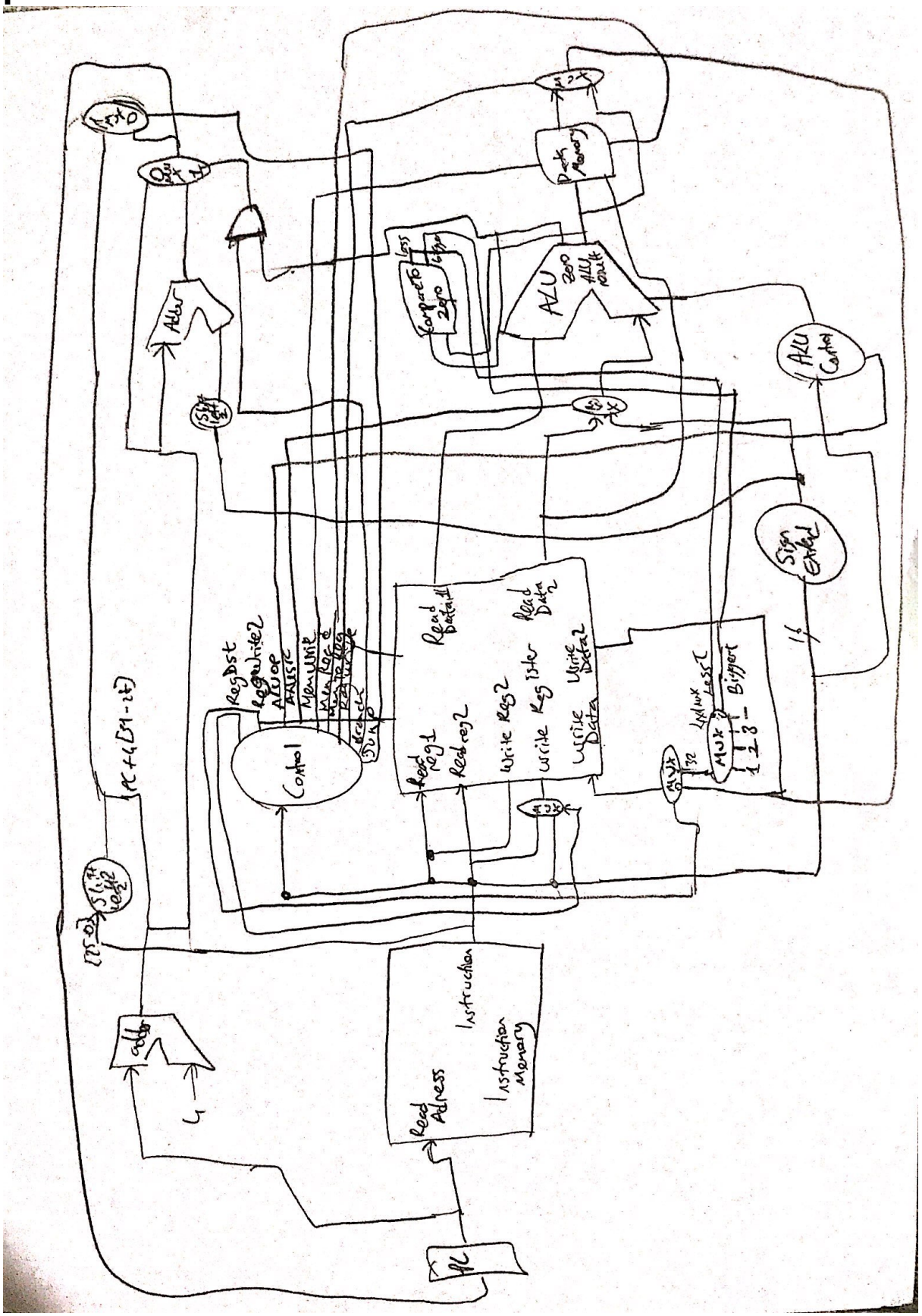
Computer Organization Report
Homework #4

Mustafa Tokgöz
171044077

My Datapath for different version of 32-bit MIPS processor.



Explanation and Results :

-Firstly, I drew a datapath for different version of 32 bit Mips processor. Then I did modules for this datapath. First I did 2x1 Mux 1 bit.

```
# Time: 0 input1:1, input2:0, s:0, out:1
# Time: 5 input1:1, input2:0, s:1, out:0
```

if select bit is 0 then output is 1 else output is 0

-After that I did 2x1 Mux 5 bit by using 2x1 Mux 1 bit for selecting rt or rd .

```
# Time: 0 input1:11111, input2:00000, s:0, out:11111
# Time: 5 input1:11111, input2:00000, s:1, out:00000
```

If select bit is 0 then output is 11111 else output is 00000

-After that I did 2x1 Mux 32 bit by using ex1 Mux 1 bit for selecting Read data 2 result or sign extend result.

```
# Time: 0 input1:11111111111111111111111111111111, input2:00000000000000000000000000000000, s:0, out:11111111111111111111111111111111
# Time: 5 input1:11111111111111111111111111111111, input2:00000000000000000000000000000000, s:1, out:00000000000000000000000000000000
```

If select bit is 0 then output is 32 bit 1 else output is 32bit 0

-After that I did 4x1 Mux 1 bit for 1 bit alu.

```
# Time: 0 input1:1, input2:0, input3:0, input4:0, s1:0, s0:0, out:1
# Time: 5 input1:0, input2:1, input3:0, input4:0, s1:0, s0:1, out:1
# Time:10 input1:0, input2:0, input3:1, input4:0, s1:1, s0:0, out:1
# Time:15 input1:0, input2:0, input3:0, input4:1, s1:1, s0:1, out:1
```

It selects first input1 then input2 then input3 then input4 with select bits.

-After that I did 4x1 Mux 32 bit for selecting 1,2,3 to write rd.

```
# Time: 0 input1:11111111111111111111111111111111, input2:00000000000000000000000000000001, input3:00000000000000000000000000000111, input4:11000000000000000000000000000000, s1:0, s0:0, out:
11111111111111111111111111111111
# Time: 5 input1:11111111111111111111111111111111, input2:00000000000000000000000000000001, input3:00000000000000000000000000000111, input4:11000000000000000000000000000000, s1:0, s0:1, out:
00000000000000000000000000000001
# Time:10 input1:11111111111111111111111111111111, input2:00000000000000000000000000000001, input3:00000000000000000000000000000111, input4:11000000000000000000000000000000, s1:1, s0:0, out:
00000000000000000000000000000111
# Time:15 input1:11111111111111111111111111111111, input2:00000000000000000000000000000001, input3:00000000000000000000000000000111, input4:11000000000000000000000000000000, s1:1, s0:1, out:
11000000000000000000000000000000
```

It selects first input1 then input2 then input3 then input4 with select bits.

-After that I did 1 bit alu. After that I did 32 bit alu by using 1 bit alu. In alu I select the opcodes as 000=andn, 001 = orn, 010 = addn, 110 = subn, 111 = xorn. This alu also check result is 0 or not.

```
# Time: 0 , opcode:000 , a:00000000000000000000000000000101, b:00000000000000000000000000000010 , result:00000000000000000000000000000000 , zero:1 , c_out:0
# Time: 5 , opcode:001 , a:00000000000000000000000000000101, b:00000000000000000000000000000010 , result:00000000000000000000000000000111 , zero:0 , c_out:0
# Time:10 , opcode:010 , a:00000000000000000000000000000101, b:00000000000000000000000000000010 , result:00000000000000000000000000000111 , zero:0 , c_out:0
# Time:15 , opcode:110 , a:00000000000000000000000000000101, b:00000000000000000000000000000010 , result:11111111111111111111111111111101 , zero:0 , c_out:1
# Time:20 , opcode:111 , a:00000000000000000000000000000101, b:00000000000000000000000000000010 , result:00000000000000000000000000000111 , zero:0 , c_out:1
```

Time 0 => 000 andn> b and a => 000..0010 and 000..0101= 000..0000

Zero = 1

Time 5 => 001 orn > b or a =>000..0010 or 000..0101 = 000…0111

Zero =0

Time 10 => 010 addn > b + a => 000..0010 + 000..0101 = 000… 0111. 2 + 5 = 7

Zero =0

Time 15 => 110 subn > b - a => 000..0010 - 000..0101 = 111… 1101.

2 - 5 = -3

Zero =0

Time 20 => 111 xorn > b xor a => 000..0010 xor 000..0101 = 000… 0111.

Zero =0

-After that I did compareTozero module. This module take alu output zero and result as parameter and gives us result is bigger than 0 or less than 0 . If output biggerT and lessT are 0 then result is equal to 0. This part for new instructions.

```
# Time: 0 input1:11111111111100111111111111111111, biggerT:0 ,lessT:1
# Time: 5 input1:00011111111111111111111111111111, biggerT:1 ,lessT:0
```

If input is bigger than 0 then biggerT = 1 else 0

If input is less than 0 then lessT = 1 else 0

-After that I did alp-control module for R-types and it gives alu_control and takes function and alu-op code.

```
# Time: 0 funct:100100 alu_op:10, alu_control:000, expectation:000
# Time: 5 funct:100101 alu_op:10, alu_control:001, expectation:001
# Time:10 funct:100000 alu_op:10, alu_control:010, expectation:010
# Time:15 funct:100010 alu_op:10, alu_control:110, expectation:110
# Time:20 funct:101010 alu_op:10, alu_control:111, expectation:111
```

I select function codes like this. I choose 101010 for xor instead sit because my processor doesn't include slt.

Func:100100 for and

Func:100101 for or

Func:100000 for add

Func:100010 for sub

Func:101010 for xor

It shows alu control codes and expectations on the screenshot.

-After that I did sign extend for 16 to 32 bit.

```
# Time: 0 input1:1111000111111101, out:11111111111111111111000111111101
# Time: 5 input1:0000000000010011, out:00000000000000000000000000010011
```

It extend with respect to msb.

-After that I did shift left 2 and add pc[31:28] to instruction jump.

This is for jump instructions.

```
# Time: 0 input1:      11111111111111111111111111, pc:0000, out:00001111111111111111111111111100
# Time: 5 input1:      11111111111111111111111111, pc:1000, out:10001111111111111111111111111100
```

It shifts left with 0 then add 4 bit of pc to top of the output.

-After that I did 1 bit adder. After that I did 32 bit adder by using 1 bit adder for branch instructions.

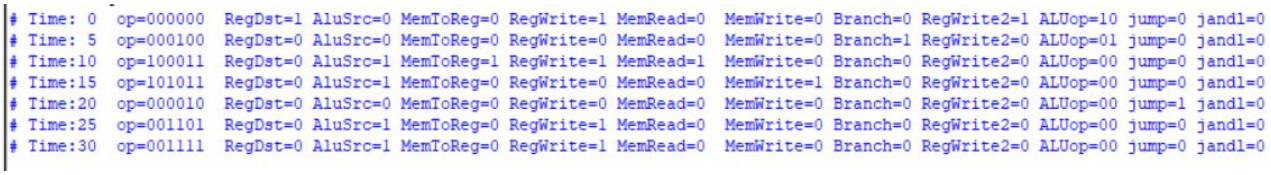
```
# Time: 0 input1:00000000000000000000000000000001, input2:00000000000000000000000000000111, cin:0, cout:0, out:00000000000000000000000000001000
# Time: 5 input1:00000000000000000000000000000001, input2:00000000000000000000000000000111, cin:1, cout:0, out:00000000000000000000000000001001
```

It adds input1 to input2 = 00..0001 + 00..0111 = 00..1000

If carry in is 1 then it adds 1 bit also to show adder is correct.

It adds input1 to input2 = 00..0001 + 00..0111 + cin (00…0001)= 00..1001

- After that I did control unit. I did this control unit with outputs as RegDst , AluSrc, MemtoReg, RegWrite , MemRead , MemWrite, Branch , RegWrite2 (this is for writing to rs for new instructions) , ALUop, jump (this is for jump instruction), jandl (this is for jandl instruction).



For test Op codes => 000000 = R-type

000100 = Beq instruction

100011 = lw instruction

101011 = sw instruction

000010 = jump instruction

001101 = ori instruction

001111 = lui instruction

Outputs is shown on the screenshot.

Implemented Parts :

2x1 Mux 1bit, 2x1 Mux 32 bit, 2x1 Mux 5 bit, 4x1 Mux 1 bit, 4x1 Mux 32 bit, 1 bit ALU , 32 bit ALU, Sign extend, CompareTozero, 1 bit Adder , 32 bit Adder , jump (shift 2 bit left and add pc most significant 4 bit), Control Unit , ALU Control.

Unimplemented Parts :

Instruction Memory, Pc, Register, Data Memory

I spend lots of time to doing these but I can’t do instruction memory , pc , register and data memory. But I did all module excepts these and showed simulations on the upper side. Thank you so much.

Mustafa Tokgöz 171044077