Lab 3 Report

CENG3420

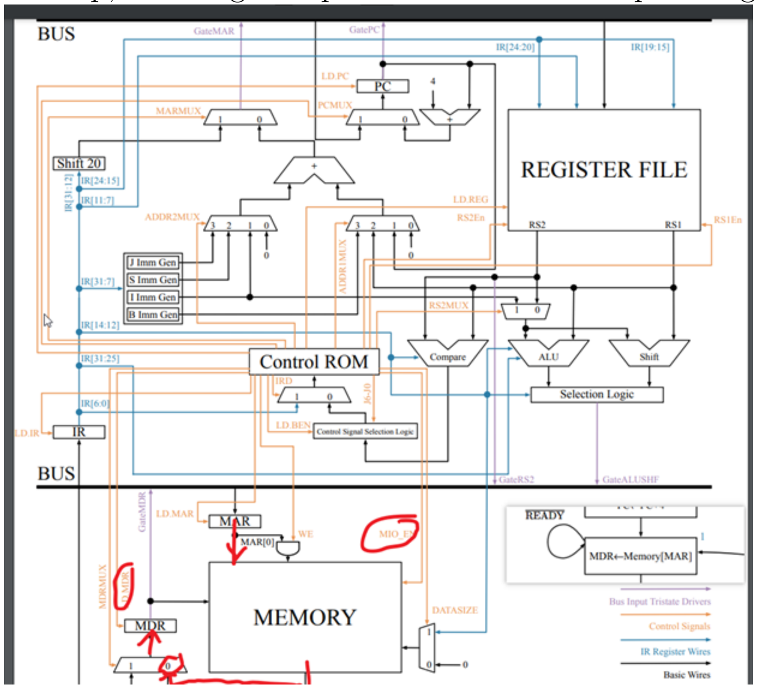
Lam Kin Ho

1155158095

6 May, 2022

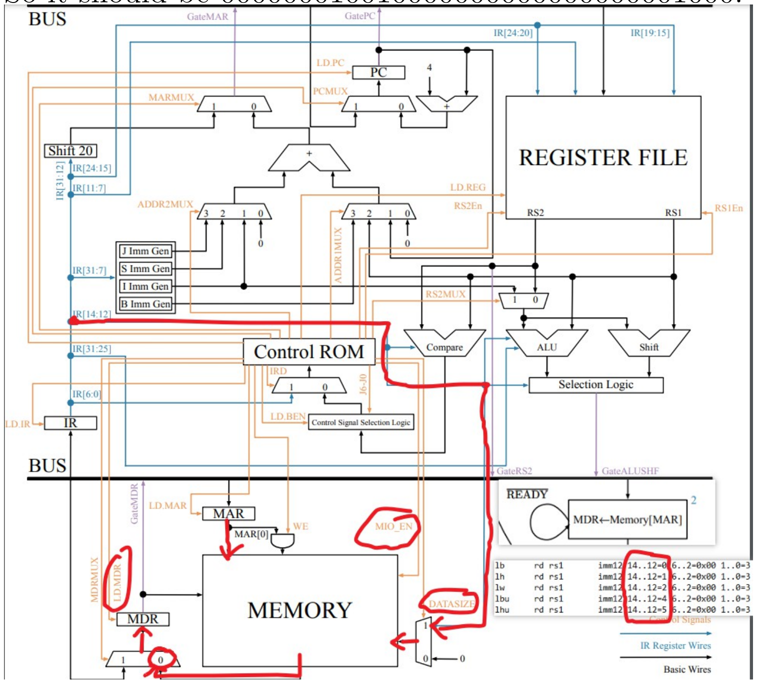
Lab 3.1

For uop, refer to graffle provided and enable required signal except J2 as waiting for READY.



its next state is 5 so I have to enable J0(8), LD.MDR(11), and MIO_EN(30).

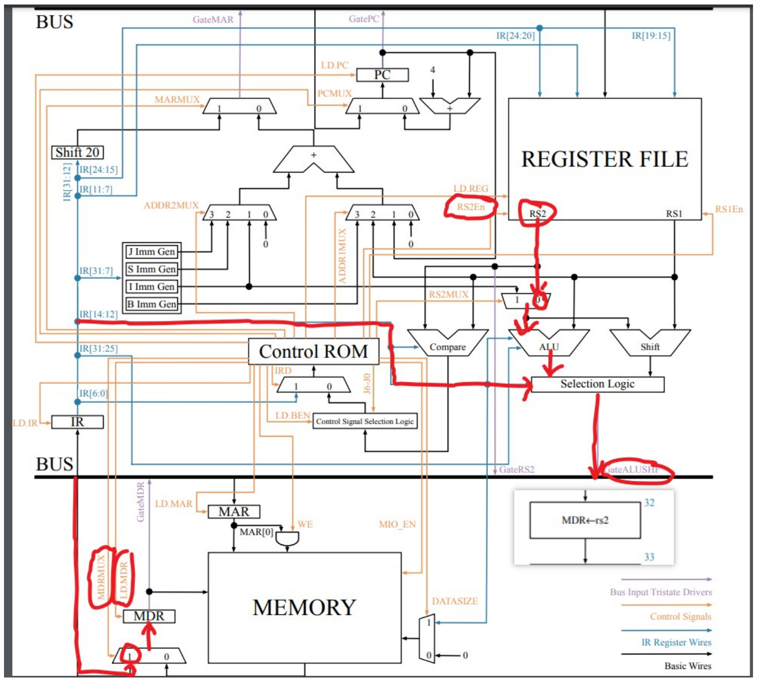
So it should be 0000000100100000000000000000001000.



```
lb     rd rs1     imm12 14..12=0 6..2=0x00 1..0=3
lh     rd rs1     imm12 14..12=1 6..2=0x00 1..0=3
lw     rd rs1     imm12 14..12=2 6..2=0x00 1..0=3
lbu    rd rs1     imm12 14..12=4 6..2=0x00 1..0=3
lhu    rd rs1     imm12 14..12=5 6..2=0x00 1..0=3
```

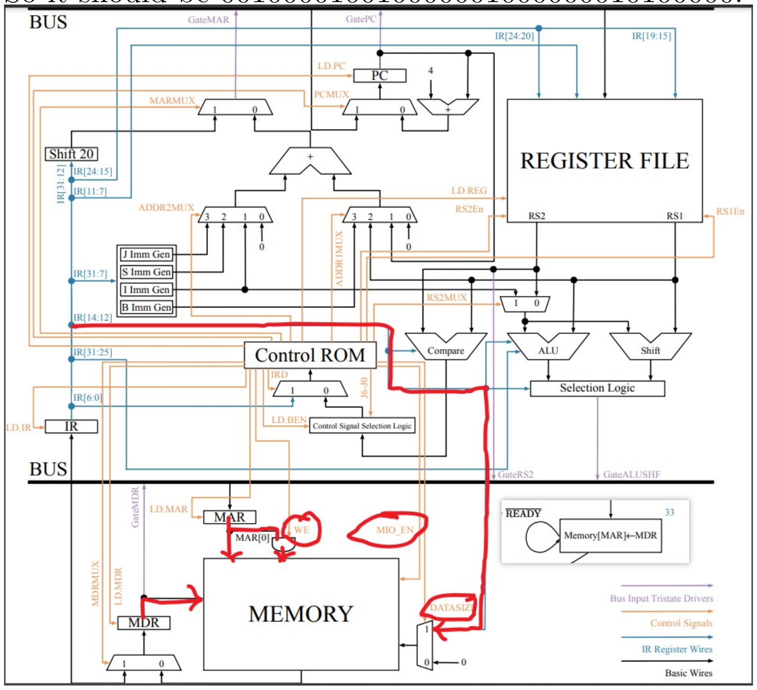
its next state is 6 so I have to enable J1(7), LD.MDR(11), MIO_EN(30), and DATASIZE(32).

So it should be 0000001000100000000000000000001010.



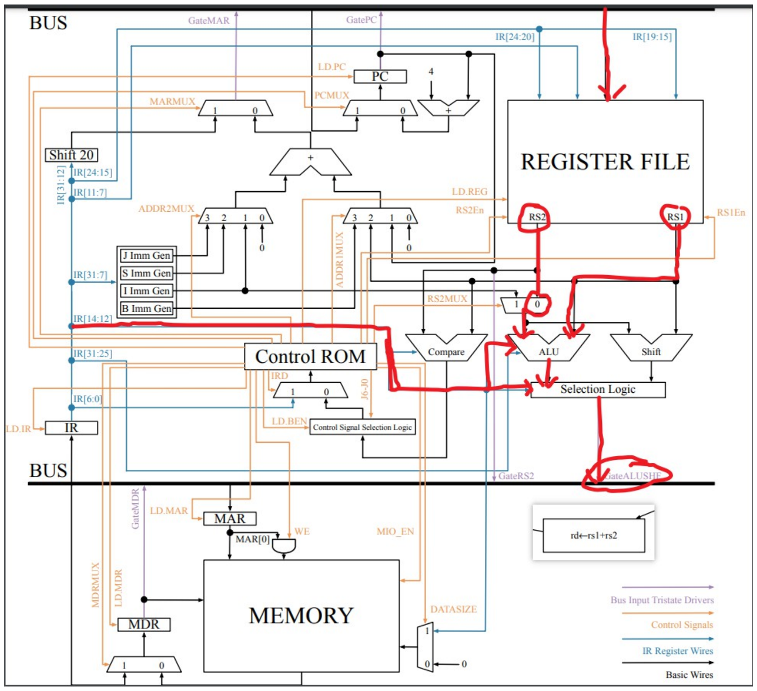
its next state is 33 so I have to enable J5(3), J0(8), LD.MDR(11), GateALUSHF(18), MDR-MUX(26), RS2En(28). MIO_En(30)

So it should be 00100000100100000001000000010100000.

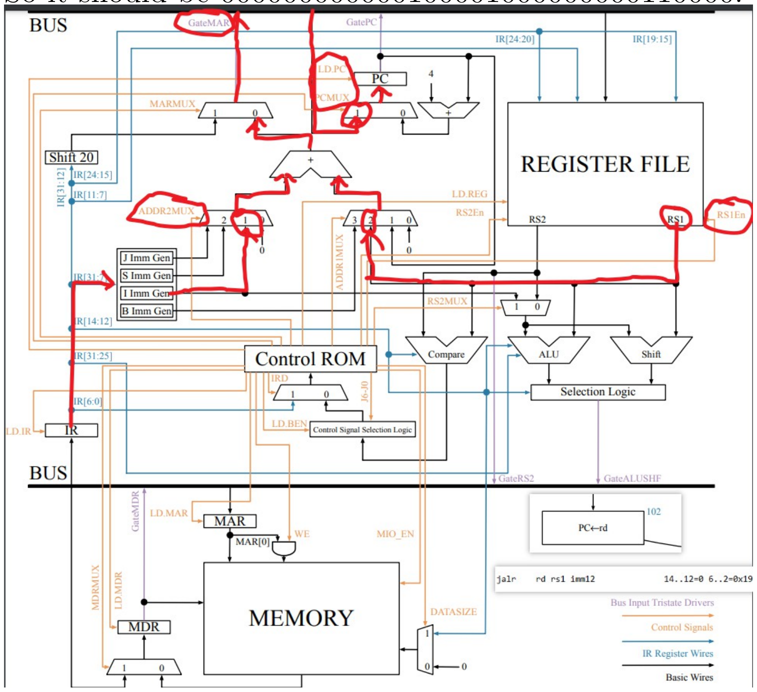


its next state is 37 so I have to enable J5(3), J0(8), MIO_EN(30), WE(31), DATASIZE(32).

So it should be 00100000100000000000000000000001110.



its next state is 0 so I have to enable LD.REG(13), GateALUSHF(18), RS2En(28), RS2En(29).
So it should be 00000000000010000100000000000110000.

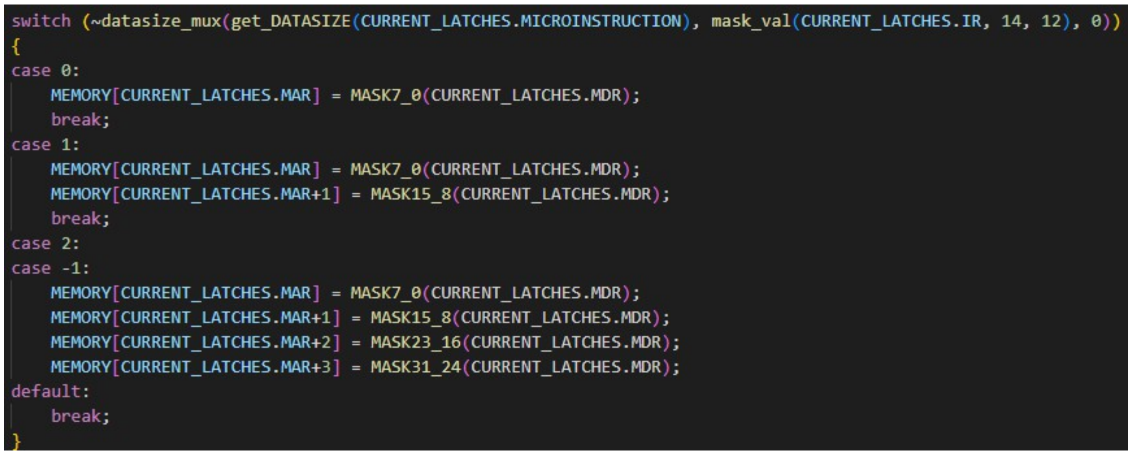


its next state is 0 so I have to enable LD.PC(9), GateMAR(16), PCMUX(20), ADDR1MUX(21), ADDR2MUX(24), RS1En(29).
So it should be 00000000100000010001100100001000.

For hard wired x0=0, just

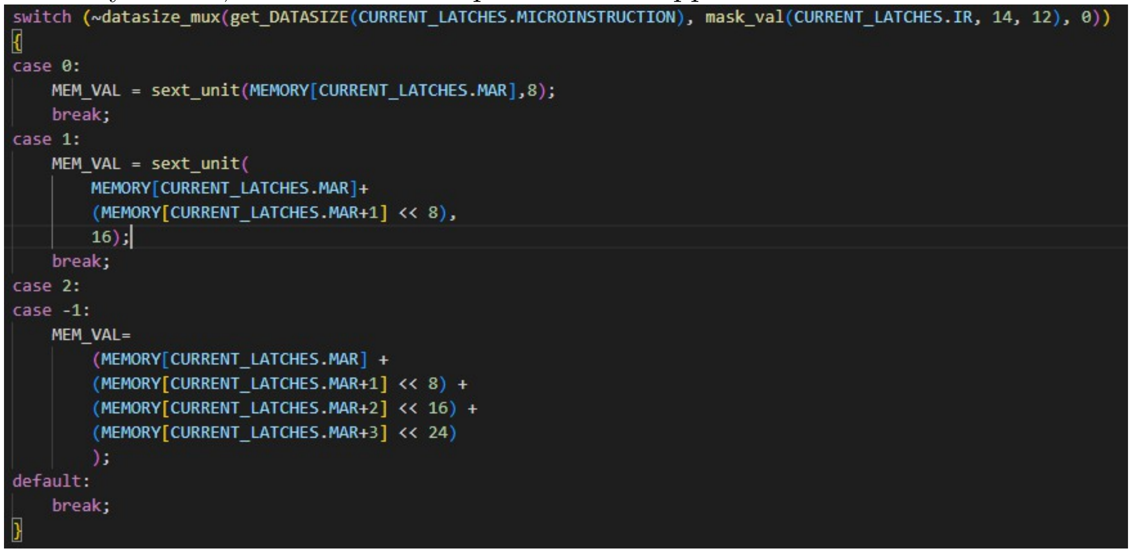
```
const CURRENT_LATCHES.REGS[0] = 0;
const NEXT_LATCHES.REGS[0] = 0;
```

Lab 3.2



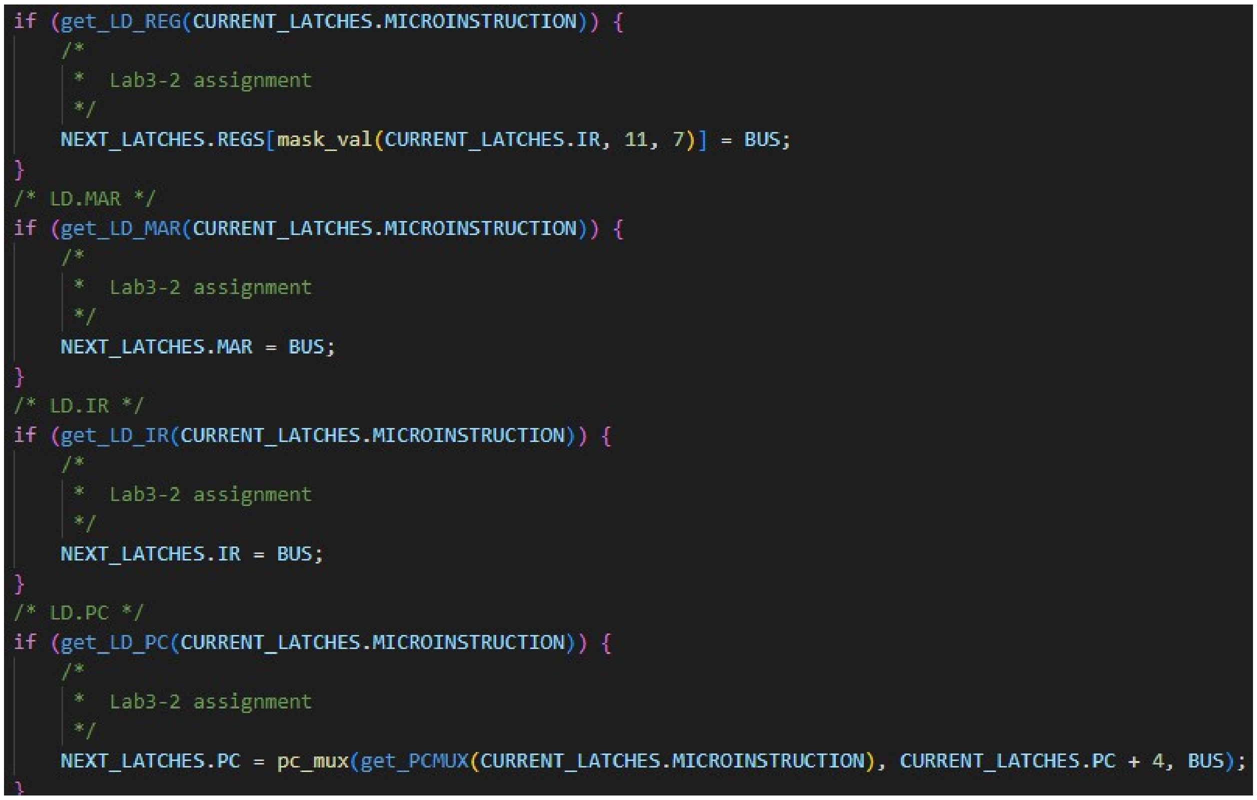
```
switch (~datasize_mux(get_DATASIZE(CURRENT_LATCHES.MICROINSTRUCTION), mask_val(CURRENT_LATCHES.IR, 14, 12), 0))
{
case 0:
    MEMORY[CURRENT_LATCHES.MAR] = MASK7_0(CURRENT_LATCHES.MDR);
    break;
case 1:
    MEMORY[CURRENT_LATCHES.MAR] = MASK7_0(CURRENT_LATCHES.MDR);
    MEMORY[CURRENT_LATCHES.MAR+1] = MASK15_8(CURRENT_LATCHES.MDR);
    break;
case 2:
case -1:
    MEMORY[CURRENT_LATCHES.MAR] = MASK7_0(CURRENT_LATCHES.MDR);
    MEMORY[CURRENT_LATCHES.MAR+1] = MASK15_8(CURRENT_LATCHES.MDR);
    MEMORY[CURRENT_LATCHES.MAR+2] = MASK23_16(CURRENT_LATCHES.MDR);
    MEMORY[CURRENT_LATCHES.MAR+3] = MASK31_24(CURRENT_LATCHES.MDR);
default:
    break;
}
```

When writing to memory, call the datasize_mux to identify whether it is lw, lh or lb. The returned value need to bitwise not as it is bitwise not it datasize_mux. The memory block is one-byte each, so similar technique in lab2 is applied.



```
switch (~datasize_mux(get_DATASIZE(CURRENT_LATCHES.MICROINSTRUCTION), mask_val(CURRENT_LATCHES.IR, 14, 12), 0))
{
case 0:
    MEM_VAL = sext_unit(MEMORY[CURRENT_LATCHES.MAR],8);
    break;
case 1:
    MEM_VAL = sext_unit(
        MEMORY[CURRENT_LATCHES.MAR]+
        (MEMORY[CURRENT_LATCHES.MAR+1] << 8),
        16);
    break;
case 2:
case -1:
    MEM_VAL=
        (MEMORY[CURRENT_LATCHES.MAR] +
        (MEMORY[CURRENT_LATCHES.MAR+1] << 8) +
        (MEMORY[CURRENT_LATCHES.MAR+2] << 16) +
        (MEMORY[CURRENT_LATCHES.MAR+3] << 24)
        );
default:
    break;
}
```

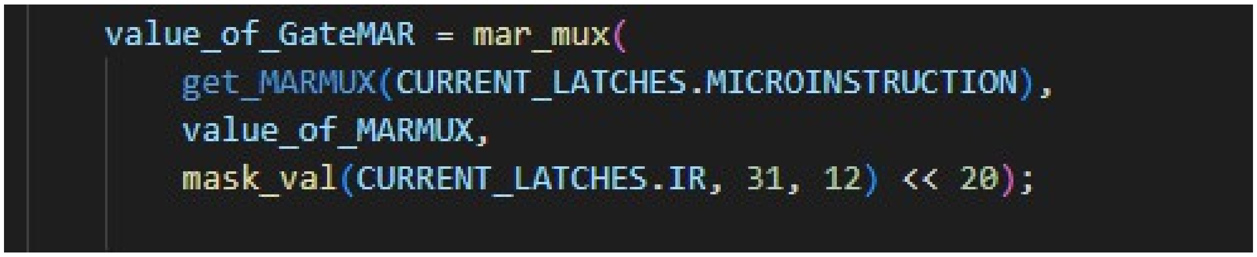
When reading from memory, similarly, call datasize_mux and bitwise not it. For lh and lb case, it have to signed extend. Other technique is same with write_mem and lab2.



```
if (get_LD_REG(CURRENT_LATCHES.MICROINSTRUCTION)) {
    /*
     *  Lab3-2 assignment
     */
    NEXT_LATCHES.REGS[mask_val(CURRENT_LATCHES.IR, 11, 7)] = BUS;
}
/* LD.MAR */
if (get_LD_MAR(CURRENT_LATCHES.MICROINSTRUCTION)) {
    /*
     *  Lab3-2 assignment
     */
    NEXT_LATCHES.MAR = BUS;
}
/* LD.IR */
if (get_LD_IR(CURRENT_LATCHES.MICROINSTRUCTION)) {
    /*
     *  Lab3-2 assignment
     */
    NEXT_LATCHES.IR = BUS;
}
/* LD.PC */
if (get_LD_PC(CURRENT_LATCHES.MICROINSTRUCTION)) {
    /*
     *  Lab3-2 assignment
     */
    NEXT_LATCHES.PC = pc_mux(get_PCMUX(CURRENT_LATCHES.MICROINSTRUCTION), CURRENT_LATCHES.PC + 4, BUS);
}
```

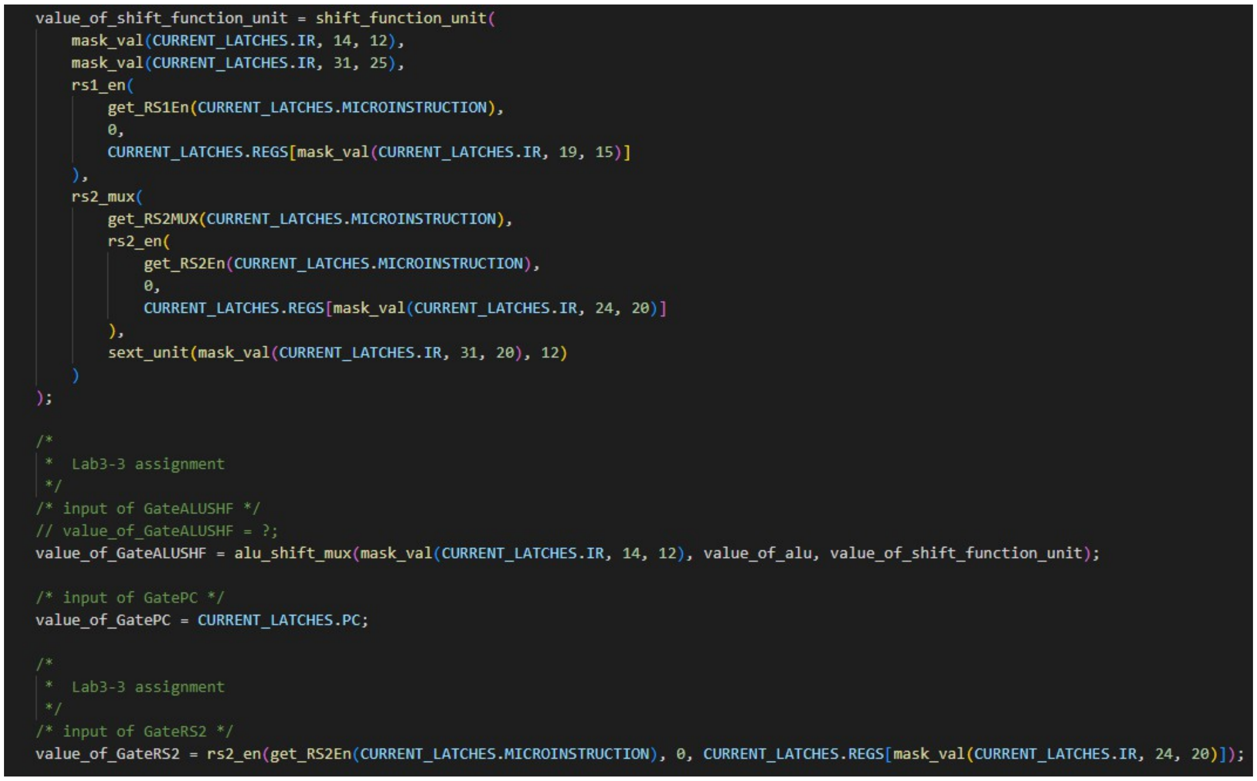
for reg/mar/ir, if its LD signal is enable, just set it as BUS value. for pc, call the pc_mux function, if LD.PC is disable, it return pc+4. if LD.PC is enable, it return BUS value.

Lab 3.3



```
value_of_GateMAR = mar_mux(
    get_MARMUX(CURRENT_LATCHES.MICROINSTRUCTION),
    value_of_MARMUX,
    mask_val(CURRENT_LATCHES.IR, 31, 12) << 20);
```

For GateMAR, call mar_mux, if MARMUX is 1, GateMAR = shift 20. if MARMUX is 0, GateMAR = value_of_MARMUX calculated above.

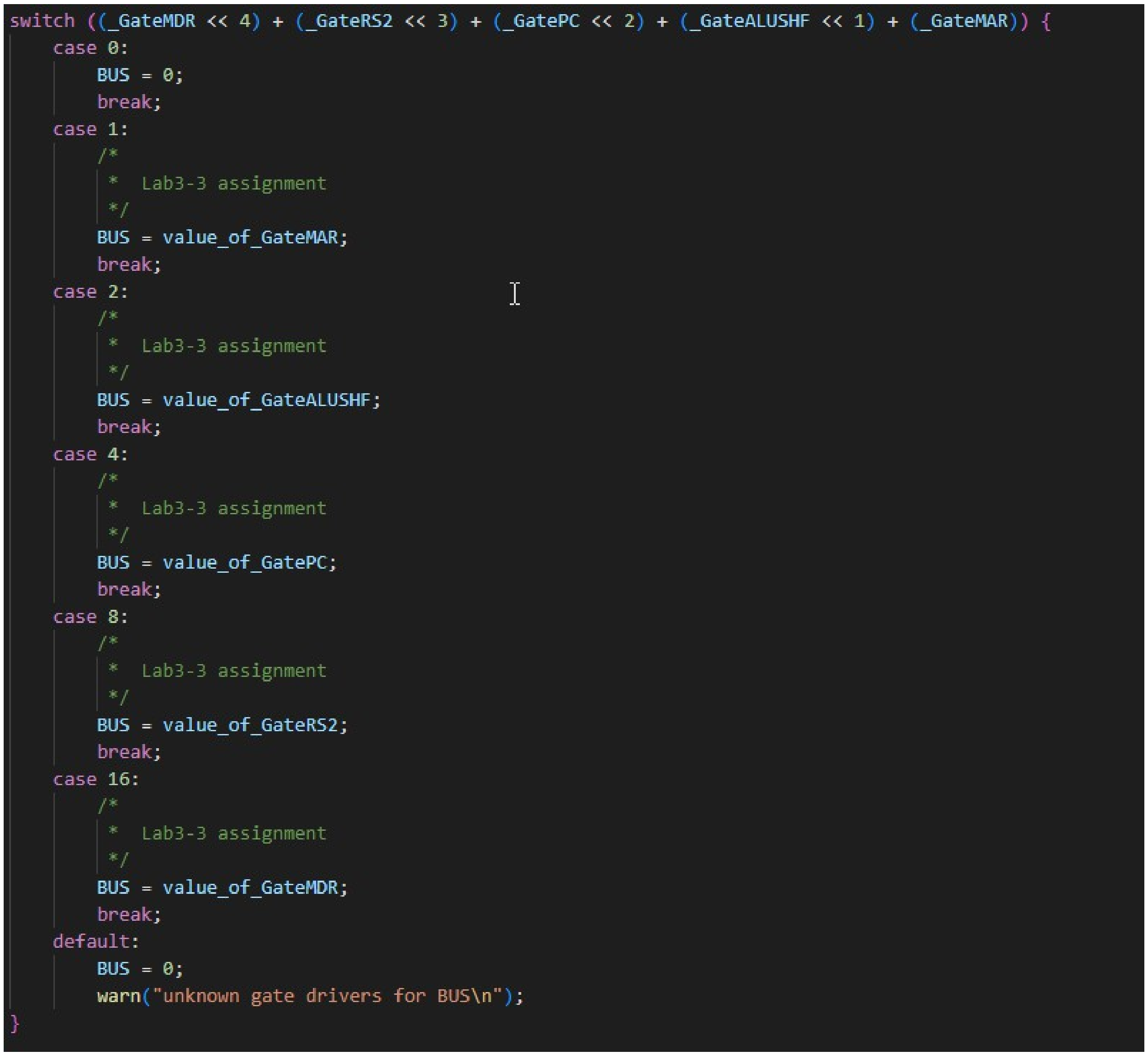


```
value_of_shift_function_unit = shift_function_unit(
    mask_val(CURRENT_LATCHES.IR, 14, 12),
    mask_val(CURRENT_LATCHES.IR, 31, 25),
    rs1_en(
        get_RS1En(CURRENT_LATCHES.MICROINSTRUCTION),
        0,
        CURRENT_LATCHES.REGS[mask_val(CURRENT_LATCHES.IR, 19, 15)]
    ),
    rs2_mux(
        get_RS2MUX(CURRENT_LATCHES.MICROINSTRUCTION),
        rs2_en(
            get_RS2En(CURRENT_LATCHES.MICROINSTRUCTION),
            0,
            CURRENT_LATCHES.REGS[mask_val(CURRENT_LATCHES.IR, 24, 20)]
        ),
        sext_unit(mask_val(CURRENT_LATCHES.IR, 31, 20), 12)
    )
);

/*
 *  Lab3-3 assignment
 */
/* input of GateALUSHF */
// value_of_GateALUSHF = ?;
value_of_GateALUSHF = alu_shift_mux(mask_val(CURRENT_LATCHES.IR, 14, 12), value_of_alu, value_of_shift_function_unit);

/* input of GatePC */
value_of_GatePC = CURRENT_LATCHES.PC;

/*
 *  Lab3-3 assignment
 */
/* input of GateRS2 */
value_of_GateRS2 = rs2_en(get_RS2En(CURRENT_LATCHES.MICROINSTRUCTION), 0, CURRENT_LATCHES.REGS[mask_val(CURRENT_LATCHES.IR, 24, 20)]);
```

For value_of_shift_function_unit, call shift_function_unit, pass (func3,func7,(if Rs1_en is enable pass RS1, else pass 0),(if RS2MUX is 1, and Rs2_en is enable, pass RS2, else pass 0, if RS2MUX is 0, pass value from I immGen))

For value_of_GateALUSHF, choose value_of_alu or value_of_shift_function_unit based on funct3.

For value_of_GateRS2, if RS2_en is enable, it equal rs2, else, it equal 0



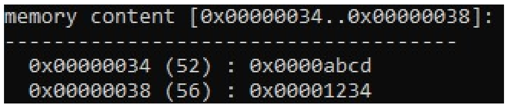
```
switch ((_GateMDR << 4) + (_GateRS2 << 3) + (_GatePC << 2) + (_GateALUSHF << 1) + (_GateMAR)) {
    case 0:
        BUS = 0;
        break;
    case 1:
        /*
         *  Lab3-3 assignment
         */
        BUS = value_of_GateMAR;
        break;
    case 2:
        /*
         *  Lab3-3 assignment
         */
        BUS = value_of_GateALUSHF;
        break;
    case 4:
        /*
         *  Lab3-3 assignment
         */
        BUS = value_of_GatePC;
        break;
    case 8:
        /*
         *  Lab3-3 assignment
         */
        BUS = value_of_GateRS2;
        break;
    case 16:
        /*
         *  Lab3-3 assignment
         */
        BUS = value_of_GateMDR;
        break;
    default:
        BUS = 0;
        warn("unknown gate drivers for BUS\n");
}
```

Finally, just assign BUS to corresponding value_of_Gate* based on control signal.

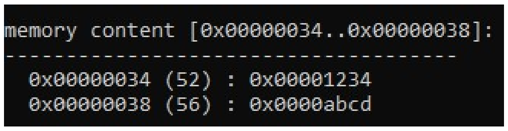
1 Result

The result of lab3.1 3.2 3.3 is same:

For swap:

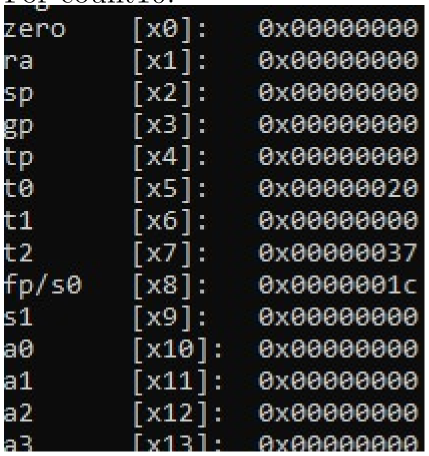


```
memory content [0x00000034..0x00000038]:
-------------------------------------
  0x00000034 (52) : 0x0000abcd
  0x00000038 (56) : 0x00001234
```



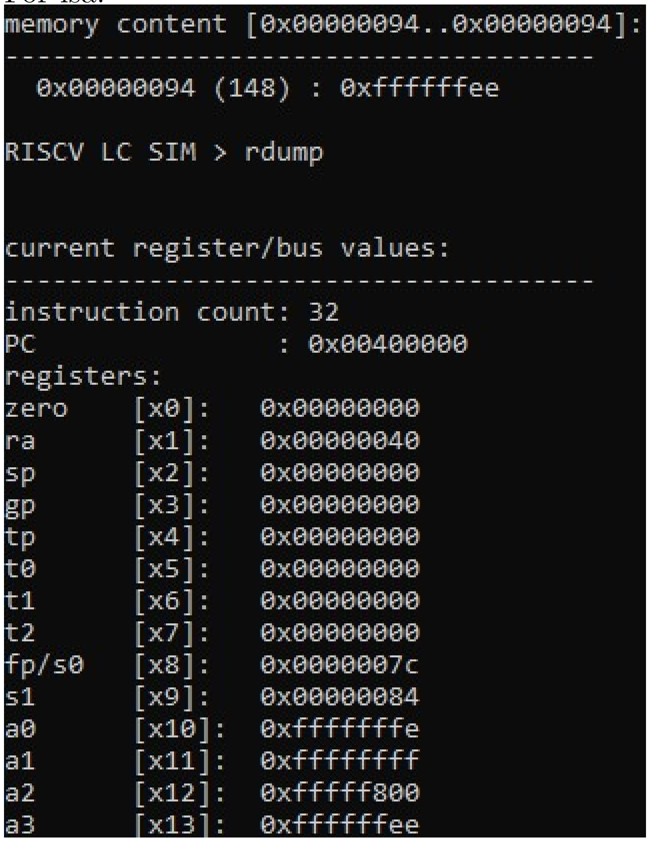
```
memory content [0x00000034..0x00000038]:
-------------------------------------
  0x00000034 (52) : 0x00001234
  0x00000038 (56) : 0x0000abcd
```

For count10:



```
zero    [x0]:   0x00000000
ra      [x1]:   0x00000000
sp      [x2]:   0x00000000
gp      [x3]:   0x00000000
tp      [x4]:   0x00000000
t0      [x5]:   0x00000020
t1      [x6]:   0x00000000
t2      [x7]:   0x00000037
fp/s0   [x8]:   0x0000001c
s1      [x9]:   0x00000000
a0      [x10]:  0x00000000
a1      [x11]:  0x00000000
a2      [x12]:  0x00000000
a3      [x13]:  0x00000000
```

For isa:



```
memory content [0x00000094..0x00000094]:
-------------------------------------
  0x00000094 (148) : 0xffffffee

RISCV LC SIM > rdump


current register/bus values:
-------------------------------------
instruction count: 32
PC                : 0x00400000
registers:
zero    [x0]:   0x00000000
ra      [x1]:   0x00000040
sp      [x2]:   0x00000000
gp      [x3]:   0x00000000
tp      [x4]:   0x00000000
t0      [x5]:   0x00000000
t1      [x6]:   0x00000000
t2      [x7]:   0x00000000
fp/s0   [x8]:   0x0000007c
s1      [x9]:   0x00000084
a0      [x10]:  0xfffffffe
a1      [x11]:  0xffffffff
a2      [x12]:  0xfffff800
a3      [x13]:  0xffffffee
```