Name: Steve Dexter B. Tamang

SID: 1155124868

Course: CENG 3420 Computer Organization

Lab 3 Report

Lab 3-1: Uop and eval_micro_sequencer

Implementation

```
int state_number;
// Get IR[6:0]
if(ird == 1){
    NEXT_LATCHES.STATE_NUMBER = mask_val(CURRENT_LATCHES.IR, 6, 0);
}else{

    NEXT_LATCHES.STATE_NUMBER = j | (_B << 3) | (CURRENT_LATCHES.READY << 2);

}
/*
 * Lab3-1 assignment
 */
printf("Lab3-1 assignment: the next state number\n");
```

This is part of the code of eval_micro_sequencer(). If IRD is 1, then the control will be the first 7 bits of the instruction register. Otherwise, when IRD is 0, first the value j3 depends on the logic of j3 or the value from the LD.BEN MUX. Then if the value of j2 depends on the value j2 or ready bit. For example in state 1, J bits = 0000001, when R is ready, the J bits becomes = 0000101, state 5.

Results

Count10.bin

After hitting md:

```
Current register/bus values :
-------------------------------------
Cycle Count  : 376
PC           : 0x00400000
IR           : 0x0000707f
STATE_NUMBER : 0x0000007f

BUS          : 0x00000000
MDR          : 0x0000707f
MAR          : 0x0000001c
MemOut       : 0x00000000
B            : 0x00000000
Registers:
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]:   0x00000000
s1      [x9]:   0x00000000
a0      [x10]:  0x00000000
a1      [x11]:  0x00000000
```

Swap.bin

Before running ‘go’, the addresses at 0x34 and 0x38 are abcd and 1234 respectively.

```
Read 60 words (240 bytes) from program into memory.

LC-RISCV-SIM> md 0x0 0x50


Memory content [0x00000000..0x00000050] :
-------------------------------------
  0x00000000 (0) : 0x000002b7
  0x00000004 (4) : 0x03428293
  0x00000008 (8) : 0x0002a283
  0x0000000c (12) : 0x00000337
  0x00000010 (16) : 0x03830313
  0x00000014 (20) : 0x00032303
  0x00000018 (24) : 0x000003b7
  0x0000001c (28) : 0x03438393
  0x00000020 (32) : 0x00000e37
  0x00000024 (36) : 0x038e0e13
  0x00000028 (40) : 0x005e2023
  0x0000002c (44) : 0x0063a023
  0x00000030 (48) : 0x0000707f
  0x00000034 (52) : 0x0000abcd
  0x00000038 (56) : 0x00001234
```

After hitting ‘go’, the addresses of 0x34 and 0x38 are 1234 and abcd respectively.

```
LC-RISCV-SIM> md 0x0 0x50


Memory content [0x00000000..0x00000050] :
-------------------------------------
  0x00000000 (0) : 0x000002b7
  0x00000004 (4) : 0x03428293
  0x00000008 (8) : 0x0002a283
  0x0000000c (12) : 0x00000337
  0x00000010 (16) : 0x03830313
  0x00000014 (20) : 0x00032303
  0x00000018 (24) : 0x000003b7
  0x0000001c (28) : 0x03438393
  0x00000020 (32) : 0x00000e37
  0x00000024 (36) : 0x038e0e13
  0x00000028 (40) : 0x005e2023
  0x0000002c (44) : 0x0063a023
  0x00000030 (48) : 0x0000707f
  0x00000034 (52) : 0x00001234
  0x00000038 (56) : 0x0000abcd
```

After hitting rd:

```
LC-RISCV-SIM> rd


Current register/bus values :
------------------------------------
Cycle Count  : 158
PC           : 0x00400000
IR           : 0x0000707f
STATE_NUMBER : 0x0000007f

BUS          : 0x00000000
MDR          : 0x0000707f
MAR          : 0x00000030
MemOut       : 0x00000000
B            : 0x00000000
Registers:
zero    [x0]:   0x00000000
ra      [x1]:   0x00000000
sp      [x2]:   0x00000000
gp      [x3]:   0x00000000
tp      [x4]:   0x00000000
t0      [x5]:   0x0000abcd
t1      [x6]:   0x00001234
t2      [x7]:   0x00000034
fp/s0   [x8]:   0x00000000
```

isa.bin

```
Current register/bus values :
-------------------------------------
Cycle Count  : 363
PC           : 0x00400000
IR           : 0x0000707f
STATE_NUMBER : 0x0000007f

BUS          : 0x00000000
MDR          : 0x0000707f
MAR          : 0x0000007c
MemOut       : 0x00000000
B            : 0x00000000
Registers:
zero    [x0]:   0x00000044
ra      [x1]:   0x00000040
sp      [x2]:   0x00000000
gp      [x3]:   0x00000000
tp      [x4]:   0x00000000
t0      [x5]:   0x00000000
t1      [x6]:   0x00000000
t2      [x7]:   0x00000000
fp/s0   [x8]:   0x00000000
s1      [x9]:   0x00000084
a0      [x10]:  0xfffffffe
a1      [x11]:  0xffffffff
a2      [x12]:  0xfffff800
a3      [x13]:  0xffffffee
a4      [x14]:  0xfffffff9
a5      [x15]:  0x0000000a
a6      [x16]:  0x0000000d
a7      [x17]:  0x00000068
```

Lab 3-2

Implementation

Cycle Memory

```
if (W) {
    /* Write */
    /*
     * Lab3-2 ass
     */
    int funct3 = mask_val(CURRENT_LATCHES.IR, 14, 12);
    int dataSize = blockDATASIZEMUX(get_DATASIZE(CURRENT_LATCHES.MICROINSTRUCTION), funct3, 0);
    write_memory(dataSize, CURRENT_LATCHES.MAR, CURRENT_LATCHES.MDR);
    printf("Lab3-2 write_memory");
    //exit(1);
} else {
    /* Read */
    /*
     * Lab3-2 assignment
     */
    int funct3 = mask_val(CURRENT_LATCHES.IR, 14, 12);
    int dataSize = blockDATASIZEMUX(get_DATASIZE(CURRENT_LATCHES.MICROINSTRUCTION), funct3, 0);
    MemOut = read_memory(dataSize, CURRENT_LATCHES.MAR);
    printf("Lab3-2 read_memory");
    //exit(1);
```

unsigned int mask_val(int, int, int)
Functions declarations

For write memory: Check what size of data to write by using the function blockDATASIZEMUX, and passing the funct3 value (IR[14:12]), which decides whether it is sw, sb or sh. After that, use the write_memory function.

For read memory: Check what size of data to read by using the function blockDATASIZEMUX, and passing funct3 value (IR[14:12]), which decides whether it is lw, lb or lh. After that, use the read_memory function and assign it to the global variable MemOut.

Eval Bus Drivers

Value Of Gate MAR:

```
int sextImm = blockSEXT(mask_val(ir, 31, 20), 12);
int block_s_type = blockSEXT(blockStypeImm(mask_val(ir, 11, 7), mask_val(ir, 31, 25)), 12);
int block_j_type = blockSEXT(blockJtypeImm(mask_val(ir, 31, 31), mask_val(ir, 30, 21),
    (ir, 20, 20), mask_val(ir, 19, 12)), 20);
adder = blockADDR2MUX(get_ADDR2MUX(microInstr), 0, sextImm, block_s_type, block_j_type);
```

First get the value of blockADDR2MUX, if 0, then the value is 0. If 1, then the value is the IR[31:20]. If 2 then, the value is S-type imm value. If 3, then the value is J-Type imm value. After that, the value from blockADDR2MUX is assigned to addr2.

```
int _rs1 = blockRS1En(get_RS1En(microInstr), 0, CURRENT_LATCHES.REGS[mask_val(ir, 19, 15)]);

int block_b_type = blockSEXT(blockBtypeImm(mask_val(ir, 7, 7), mask_val(ir, 11, 8),
    mask_val(ir, 30, 25), mask_val(ir, 31, 31)), 12);
adder += blockADDR1MUX(get_ADDR1MUX(microInstr), 0, CURRENT_LATCHES.PC, _rs1, block_b_type);
```

Next, get the value of ADDR1MUX, if 0, then the value is 0. If 1, then the value is PC. If 2, then the value is rs1. If 3, then the value is the B-Type imm. After that, the value is assigned to the adder added with the previous value.

```
int lshf = blockLSHF20(mask_val(CURRENT_LATCHES.IR, 31, 20));
valueOfGateMAR = blockMARMUX(get_MARMUX(microInstr), adder, lshf);
```

After that, the value of addr1 and addr2 are added together. Then the value to be stored to valueOfGateMAR is decided by blockMARMUX. If the MARMUX is 0, then the valueOfGateMAR is (addr1 + addr2), otherwise if MARMUX is 1, then get the IR[31:20] shifted by 20.

Value of Gate RS2

```
int _rs2 = blockRS2En(get_RS2En(microInstr), 0, CURRENT_LATCHES.REGS[mask_val(ir, 24, 20)]);
valueOfGateRS2 = _rs2;
```

The valueOfGateRS2 is determined by RS2En. If RS2En is 1, then the value of rs2 will be assigned to valueOfGateRS2. Otherwise, if RS2En is 0, then the value of rs2 is 0.

Value of Gate PC

```
valueOfGatePC = CURRENT_LATCHES.PC;
```

The valueOfGatePC is just the current PC.

Value of Gate MDR

```
valueOfGateMDR = CURRENT_LATCHES.MDR;
```

The valueOfGateMDR is equal to the current memory data register.

Value of Gate ALUSHF

```
valueOfGateRS2 = blockRS2En(get_RS2En(microInstr), 0, _rs2);

int _imm12 = mask_val(CURRENT_LATCHES.IR, 31, 20);
int funct7 = mask_val(CURRENT_LATCHES.IR, 31, 25);

// RS2MUX, returns rs2 or imm12 depending on the RS2MUX
int operand = blockRS2MUX(get_RS2MUX(microInstr), _rs2, blockSEXT(_imm12, 12));



// valueOfGateALUSHF
int valueOfALU = blockALU(funct3, funct7, _rs1, operand);
int valueOfSHF = blockSHF(funct3, funct7,_rs1, operand);
valueOfGateALUSHF = blockALUSHFMUX(funct3, valueOfALU, valueOfSHF);
```

First, the value of operand is determined by the blockRS2MUX. The blockRS2MUX can be either the imm value or rs2. Then use the function valueOfALU and valueOfSHF. After that, the blockALUSHF with funct3, determines whether the valueOfALU or valueOfSHF is assigned to valueOfGateALUSHF.

Drive Bus

```
void drive_bus() {
    /*
     *  Lab3-2 assignment
     */
    int* micro_instr = CURRENT_LATCHES.MICROINSTRUCTION;

    int gate_mdr = get_GateMDR(micro_instr);
    int gate_rs2 = get_GateRS2(micro_instr);
    int gate_pc = get_GatePC(micro_instr);
    int gate_alushf = get_GateALUSHF(micro_instr);
    int gate_mar = get_GateMAR(micro_instr);

    int choose_bus = (gate_mdr << 4) + (gate_rs2 << 3) + (gate_pc << 2) + (gate_alushf << 1) + gate_mar;

    switch (choose_bus){
        case 0:
            BUS = 0;
            break;
        case 1:
            BUS = valueOfGateMAR;
            break;
        case 2:
            BUS = valueOfGateALUSHF;
            break;
        case 4:
            BUS = valueOfGatePC;
            break;
        case 8:
            BUS = valueOfGateRS2;
            break;
        case 16:
            BUS = valueOfGateMDR;
            break;
        default:
            BUS = 0;
            break;
    }
```

First, we get the gates of MDR, MAR, ALUSHF, RS2, and, PC. Then assigning it to choose_bus and combining the values of the gates into a 16 bit value.

If the value is 0, BUS = 0. If the value is 1, BUS = valueOfGateMAR. If the value is 2, then BUS = valueOfGateALUSHF. If the value is 4, BUS = valueOfGatePC. If the value is 8, then BUS = valueOfGateRS2. If the value is 16, then the BUS = valueOfGateMDR.

Latch Datapath Values

LD_REG

```
/* LD.REG */
if (get_LD_REG(CURRENT_LATCHES.MICROINSTRUCTION)) {
    /*
     *  Lab3-2 assignment
     */
    // [11:7]
    NEXT_LATCHES.REGS[mask_val(CURRENT_LATCHES.IR, 11, 7)] = BUS;
    printf("Lab3-2 LD_REG");
    //exit(1);
}
```

The next latches regs (rd) will be the value of the BUS.

MAR

```
/* LD.MAR */
if (get_LD_MAR(CURRENT_LATCHES.MICROINSTRUCTION)) {
    /*
     *  Lab3-2 assignment
     */
    NEXT_LATCHES.MAR = BUS;
    printf("Lab3-2 LD_MAR");
    //exit(1);
}
```

The next latches MAR value is equal to BUS.

LD_IR

```
/* LD.IR */
if (get_LD_IR(CURRENT_LATCHES.MICROINSTRUCTION)) {
    /*
     *  Lab3-2 assignment
     */
    printf("Lab3-2 LD_IR");
    NEXT_LATCHES.IR = BUS;
    //exit(1);
}
```

The NEXT_LATCHES.IR is equal to the BUS.

LD_PC

```
/* LD.PC */
if (get_LD_PC(CURRENT_LATCHES.MICROINSTRUCTION)) {
    /*
     *  Lab3-2 assignment
     */
    printf("Lab3-2 LD_PC");
    int pc_val = get_PCMUX(CURRENT_LATCHES.MICROINSTRUCTION);
    int PCAdd4 = CURRENT_LATCHES.PC + 4;
    NEXT_LATCHES.PC = blockPCMUX(pc_val, PCAdd4, BUS);
    //NEXT_LATCHES.PC = BUS;
    //exit(1);
}
```

The NEXT_LATCHES.PC is determined by the blockPCMUX. First check the PCMUX bit in the microInstruction. If it is 1, then the next PC is the BUS, otherwise, the next PC is PC + 4.

Results

Count10.bin

After pressing go and then rd

```
Current register/bus values :
-----------------------------------
Cycle Count  : 376
PC           : 0x00400000
IR           : 0x0000707f
STATE_NUMBER : 0x0000007f

BUS          : 0x00000000
MDR          : 0x0000707f
MAR          : 0x0000001c
MemOut       : 0x00000000
B            : 0x00000000
Registers:
zero    [x0]:   0x00000000
ra      [x1]:   0x00000000
sp      [x2]:   0x00000000
gp      [x3]:   0x00000000
tp      [x4]:   0x00000000
t0      [x5]:   0x00000020
t1      [x6]:   0x00000000
t2      [x7]:   0x00000037
```

Isa.bin

After entering go and then rd

```
Current register/bus values :
-----------------------------------
Cycle Count  : 363
PC           : 0x00400000
IR           : 0x0000707f
STATE_NUMBER : 0x0000007f

BUS          : 0x00000000
MDR          : 0x0000707f
MAR          : 0x0000007c
MemOut       : 0x00000000
B            : 0x00000000
Registers:
zero    [x0]:   0x00000044
ra      [x1]:   0x00000040
sp      [x2]:   0x00000000
gp      [x3]:   0x00000000
tp      [x4]:   0x00000000
t0      [x5]:   0x00000000
t1      [x6]:   0x00000000
t2      [x7]:   0x00000000
fp/s0   [x8]:   0x00000000
s1      [x9]:   0x00000084
a0      [x10]:  0xfffffffe
a1      [x11]:  0xffffffff
a2      [x12]:  0xfffff800
a3      [x13]:  0xffffffee
a4      [x14]:  0xfffffff9
a5      [x15]:  0x0000000a
a6      [x16]:  0x0000000d
a7      [x17]:  0x00000068
s2      [x18]:  0x00000000
```

Swap.bin

After entering md 0x0 0x50, before entering go.

```
Memory content [0x00000000..0x00000050] :
------------------------------------
  0x00000000 (0) : 0x000002b7
  0x00000004 (4) : 0x03428293
  0x00000008 (8) : 0x0002a283
  0x0000000c (12) : 0x00000337
  0x00000010 (16) : 0x03830313
  0x00000014 (20) : 0x00032303
  0x00000018 (24) : 0x000003b7
  0x0000001c (28) : 0x03438393
  0x00000020 (32) : 0x00000e37
  0x00000024 (36) : 0x038e0e13
  0x00000028 (40) : 0x005e2023
  0x0000002c (44) : 0x0063a023
  0x00000030 (48) : 0x0000707f
  0x00000034 (52) : 0x0000abcd
  0x00000038 (56) : 0x00001234
```

After pressing go and then enter md 0x0 0x50.

```
Memory content [0x00000000..0x00000050] :
------------------------------------
  0x00000000 (0) : 0x000002b7
  0x00000004 (4) : 0x03428293
  0x00000008 (8) : 0x0002a283
  0x0000000c (12) : 0x00000337
  0x00000010 (16) : 0x03830313
  0x00000014 (20) : 0x00032303
  0x00000018 (24) : 0x000003b7
  0x0000001c (28) : 0x03438393
  0x00000020 (32) : 0x00000e37
  0x00000024 (36) : 0x038e0e13
  0x00000028 (40) : 0x005e2023
  0x0000002c (44) : 0x0063a023
  0x00000030 (48) : 0x0000707f
  0x00000034 (52) : 0x00001234
  0x00000038 (56) : 0x0000abcd
```

The register of 0x34 and 0x38 are swapped from 0x34 = abcd, 0x38 = 1234 to 0x34 = 1234 and 0x38 = abcd.

After pressing rd

```
LC-RISCV-SIM> rd


Current register/bus values :
-------------------------------------
Cycle Count  : 158
PC           : 0x00400000
IR           : 0x0000707f
STATE_NUMBER : 0x0000007f

BUS          : 0x00000000
MDR          : 0x0000707f
MAR          : 0x00000030
MemOut       : 0x00000000
B            : 0x00000000
Registers:
zero    [x0]:   0x00000000
ra      [x1]:   0x00000000
sp      [x2]:   0x00000000
gp      [x3]:   0x00000000
tp      [x4]:   0x00000000
t0      [x5]:   0x0000abcd
t1      [x6]:   0x00001234
t2      [x7]:   0x00000034
fp/s0   [x8]:   0x00000000
```