Note

Before I heard that we don't need to include the textual explanations in report, I have already finished the report of Lab3.1 and Lab3.2. I asked on Piazza and I found out that I can still include the textual report in here.

Lab 3.1 Report

1. Problem analysis

   In lab 3.1 we are implementing the simulator of RISCV-LC, especially the finite state machine for stage update. Different from previous code, this time we are separating each instruction into many different stages, and for each stage the 33 control signal bits will be different, and we have to implement how are the 128 states interconnected (which is the finite state machine) and in each state, what are each signal.

2. My understandings of the RISCV-LC simulator structure

   (a) My understanding of the instruction execution process:

   The program starts with the main function. In the main function, the program first get the binary file from the console and opens it, then it starts to initialize like allocating space for memory, loading the control state into the code etc.

   Then it goes into a while loop in which the get_command function is executed and will get command from the user interface. It can show the values in specific memories or registers, and it can also execute instructions.

   The execution process is handled by the run functions. It runs a specific numbers of instructions by calling the cycle() function a specific amount of times, where each time the cycle() function executes exactly one instruction.

   In the cycle instruction, the core steps of the simulator are executed. The core steps are separated into 5 parts: eval_micro_sequencer, cycle_memory, eval_bus_drivers, drive_bus, datapath_values. Which are basically to update each parts including PC, memory, BUS and so on in the microarchitecture for each instruction. These steps are determined by the control signals in the file uop.

   (b) My understanding of the finite state machine

   There are 22 states in the finite state machine in total, but these ID (state number) are from 0 to 127, this is first good for future extensions, and also it is because the IR[6:0] and J6~J0 are 7 bits and is easy to map to 0 to 127. In the finite state machine graph there are 21 states, and the HALT state(127) is ignored. From the graph we can know that for each state, what processes are done and what are the next state for each state, so we can implement the uop file.

   (c) My understanding of the uop file

   uop file is the file where we store the exact numbers of each state. It has 128

rows in total, which corresponds to the 128 states with state numbers 0 to 127. For each row, there are 33 columns, which correspond to 33 specific control signals for each states. Control signals include:

(i) IRD: it determines if the state number of next state is from the instruction bits [6:0] (when IRD is 1) or from J6~J0 (together with B, READY, etc.) in ROM (when IRD is 0).
(ii) J6~J0: when IRD is 0, they partly determine the state transition.
(iii) LD: they basically enable whether each structure (PC, MAR, etc.) read and update their values in the current stage or not.
(iv) Gate: gate is enable signals for bus tristate driver. If they are enabled (1), it means that on the bus we can read its value, otherwise we disable it (0).
(v) MUX: just the selection value of each multiplexer.
(vi) En: the enable value of output or input for register file and memory
(vii) RESET: a special signal to indicate whether to reset the whole RISCV-LC.

3. My understanding of some key implementations in the code
   (a) Some commonly used functions and macros
      (i) handle_get_element_of_error(x, id):
      By masking and shifting, we can get specific digits of the input, so we can use MACROS to easily get the IRD, J, B etc. values.
      (ii) MUX function:
      MUX functions are the simulator of real multiplexers. For example, int blockBMUX(int, int, int): represent the B mux: return B or 0 according to LD.BEN.
   (b) Some key variables
      (i) struct_system_latches:
      The latch is kind of the set that stores all the values for state, including PC, registers, MAR, IR, B, READY, MICROINDTRUCTION, and STATE_NUMBER. We have 2 struct_system_latches which are CURRENT_LATCHES and NEXT_LATCHES that stores the value of the current and next stage.

4. Handle hard-wired 0
   In the code, we need to handle the hard-wired 0: which means that no matter what operation is done to REG[0], in the next stage, the REG[0] is still zero. This can be done by setting NEXT_LATCHES.REGS[0] to 0 at the eval_micro_sequencer() function:

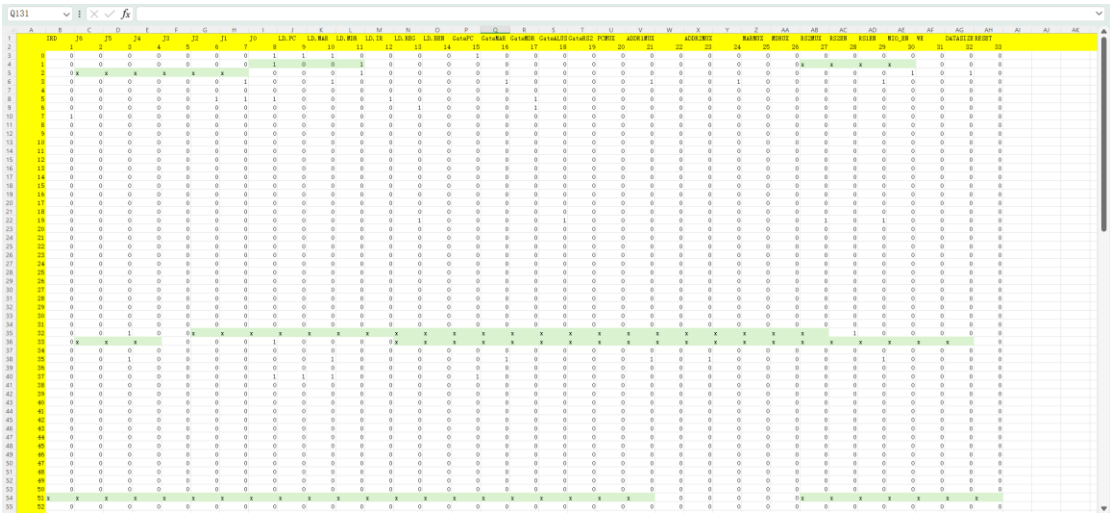
```
    // error("Lab3-1 assignment: x0 is hard-wired to zero\n");
    // hard-wired 0
    NEXT_LATCHES.REGS[0] = 0;
```

5. Handle the uop file
   (a) Preprocessing of the uop file

   The uop matrix is really hard to read, so I decide to convert it to an excel file so that I can quickly locate specific rows and columns. So I wrote the to_csv.c file to convert it to csv file by adding comma in-between:

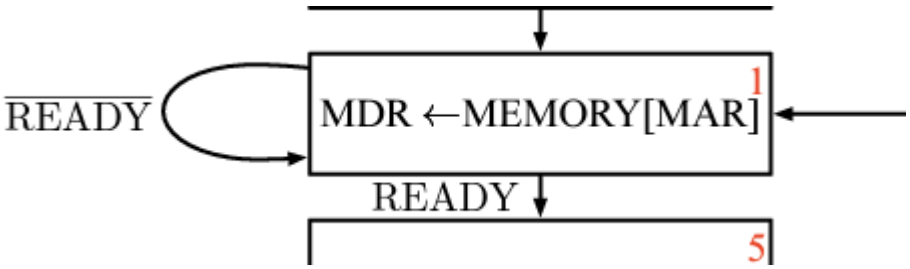
```c
#include <stdio.h>

int main(int argc, char* argv[]) {
    if (argc < 3) {
        printf("Usage: program_name input_file output_file\n");
        return 1;
    }

    FILE* file_in = fopen(argv[1], "r");
    if (file_in == NULL) {
        printf("Error opening input file.\n");
        return 1;
    }

    FILE* file_out = fopen(argv[2], "w");
    if (file_out == NULL) {
        printf("Error opening output file.\n");
        fclose(file_in);
        return 1;
    }

    int ch;
    while ((ch = fgetc(file_in)) != EOF) {
        if (ch == '\n') {
            fputc('\n', file_out);
            continue;
        }

        fputc(ch, file_out);
        fputc(',', file_out);
    }

    fclose(file_in);
    fclose(file_out);

    return 0;
}
```

   And after converting, we can open it in Excel with clearer layout (with other labels in yellow and undetermined values in green):

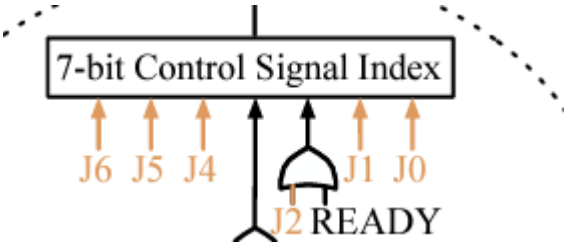


Then, we can modify each undetermined value.

(b) State 1:



In state 1, the next state depends on the value of READY, if READY is 0, then next state is 1 (0000001if READY is 1, the next state is 5 (0000101).



Looking at the control part we can see, if we set J0 to 1, and all others to 0, we can achieve this purpose since if J2 is 0, it is actually READY that determines bit2.

In this stage we update MDR to the value of memory at the address of MAR. Therefore, no PC is updated, so LD.PC is 0;

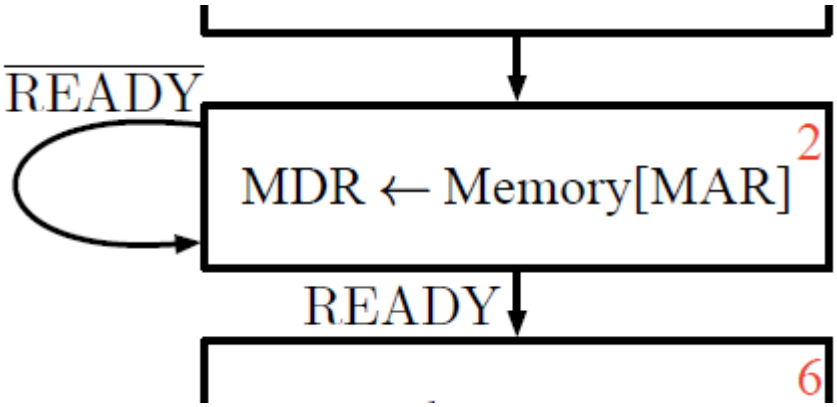
No MAR is updated, so LD.MAR is 0;

MDR is updated, so LD.MDR is 1;

In this stage, we don't need to read the values from register file, so RS1EN and RS2EN should both be disabled, set to 0. The RS2MUX should be set to 0 to choose the output of rs2 because in this stage, we don't want to calculate with ALU and the 2 inputs of the ALU should be disabled. But there is no control for disabling the I immediate generator, so we can only let the RS2MUX choose the disabled RS2 to make the input disabled.

Since we want to read the data from memory to MDR, we should make the memory IO enabled, by setting MIO_EN to 1.

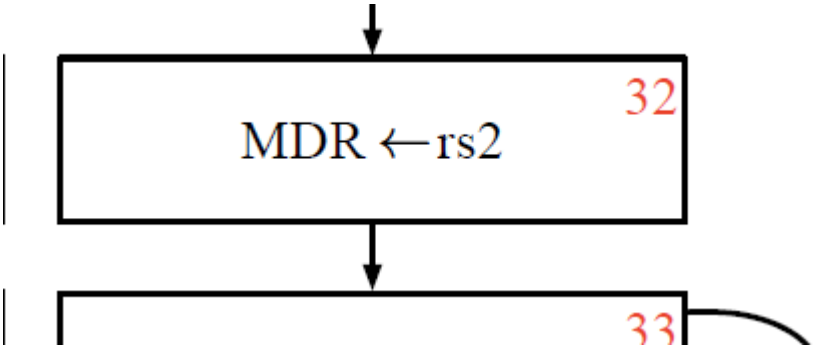
(c) State 2



State 2 is almost the same with state 1, the only difference is that if READY is 0, then next state is 2 (0000010if READY is 1, the next state is 6 (0000110).

Looking at the control part we can see, if we set J1 to 1, and all others to 0, we can achieve this purpose since if J2 is 0, it is actually READY that determines bit2.

(d) State 32



The next state is 33 (0b 0100001) so J0 and J5 should be set to 1, others 0.

This state, we load the value of rs2 into MDR, therefore,

No PC is updated, so LD.PC is 0;

No MAR is updated, so LD.MAR is 0;

MDR is updated, so LD.MDR is 1;

No instruction is updated, so LD.IR is 0;

No register is updated, so LD.REG is 0.

Since the next state number doesn't depend on the value of B, so in the multiplexer we set LD.BEN to 0 meaning that we won't choose the B value to determine the control signal.

For the tristate driver gates, since in this stage the value on the bus is the value of rs2, so only the gate that controls RS2 value should be enabled and all other values are disabled. Meaning that GatePC, GateMAR, GateMDR, GateALUSHF are set to 0, and GateRS2 is set to 1.

Since PC is not updated at this stage, so in theory we don't need to determine the values of PCMUX, ADDR1MUX, ADDR2MUX because the LD.PC is disabled and these values are not used. But in practice, we set them to the default value, which is 0, meaning that PCMUX=0, ADDR1MUX, ADDR2MUX = 00.

Since the GateMAR is disabled, actually the value of MARMUX is unused and in theory we don't need to use the value determined by MARMUX, but in practice, we set it to the default value, which is 0, meaning that MARMUX=0.
Since the value of MDR comes from the value of rs2 in the BUS, so that the input of MDR should be from the bus, so that MDRMUX should be 1 to choose from the BUS.
In this stage, we don't need the value of Selection Logic part, so in theory the input of the ALU determined by RS2MUX doesn't have to be determined, but in practice, we set it to default 0.

(e) State 33

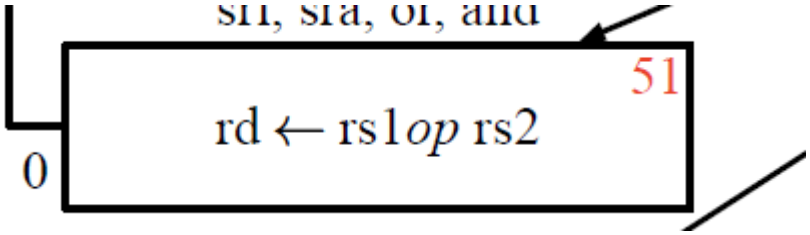
In state 33, the next stage determination is almost the same with state 1, the only difference is that if READY is 0, then next state is 33 (0100001) if READY is 1, the next state is 37 (0100101).
Looking at the control part we can see, if we set J1 and J5 to 1, and all others to 0, we can achieve this purpose since if J2 is 0, it is actually READY that determines bit2.
In this state, we load the value of MDR to the memory at the address MAR, therefore, no register is updated, so LD.REG is 0;
Since the next state number doesn't depend on the value of B, so in the multiplexer we set LD.BEN to 0 meaning that we won't choose the B value to determine the control signal.
For the tristate driver gates, since in this stage no value on the bus is needed to be transfered, so all gates should be disabled. Meaning that GatePC, GateMAR, GateMDR, GateALUSHF and GateRS2 are set to 0.
Since PC is not updated at this stage, so in theory we don't need to determine the values of PCMUX, ADDR1MUX, ADDR2MUX because the LD.PC is disabled and these values are not used. But in practice, we set them to the default value, which is 0, meaning that PCMUX=0, ADDR1MUX, ADDR2MUX = 00.
Since the GateMAR is disabled, actually the value of MARMUX is unused and in theory we don't need to use the value determined by MARMUX, but in practice, we set it to the default value, which is 0, meaning that MARMUX=0.
Since we don't need to load anything to the MDR, so in theory we don't need to determine MDRMUX, but in practice, we set it to the default value, which is 0, meaning that MDRMUX=0.
Since we don't need to output anything from the register file, so we disable the output of rs1 and rs2, meaning that RS2EN and RS1EN are set to 0.

The RS2MUX should be set to 0 to choose the output of rs2 because in this stage, we don't want to calculate with ALU and the 2 inputs of the ALU should be disabled. But there is no control for disabling the I immediate generator, so we can only let the RS2MUX choose the disabled RS2 to make the input disabled.

Since we need to modify the memory, so that MIO is enabled, meaning that MIO_EN is set to 1, and since we need to write to memory, WE is enabled, set to 1.

Since the bit width should be selected by the control signal, the DATASIZE multiplexer should be set to 1.

(f) State 51



Since the next state of this state does not depend on the instruction, we don't need to read from the IR[6:0] and the IRD is set to 0.

Since the next state number is 0, then we should set all J6~J0 to 0 to represent 0.

In this state, we update the rd to be the value calculated by the operation on rs1 and rs2, therefore,

No PC is updated, so LD.PC is 0;

No MAR is updated, so LD.MAR is 0;

No MDR is updated, so LD.MDR is 0;

No instruction is updated, so LD.IR is 0;

Register file is updated, so LD.REG is 1.

Since the next state number doesn't depend on the value of B, so in the multiplexer we set LD.BEN to 0 meaning that we won't choose the B value to determine the control signal.

For the tristate driver gates, since in this stage the value on the bus is the value of the ALU, so only the gate that controls Selection logic value should be enabled and all other values are disabled. Meaning that GatePC, GateMAR, GateMDR, GateRS2 are set to 0, and GateALUSHF is set to 1.

Since PC is not updated at this stage, so in theory we don't need to determine the values of PCMUX, ADDR1MUX because the LD.PC is disabled and these values are not used. But in practice, we set them to the default value, which is 0, meaning that PCMUX=0, ADDR1MUX=00.

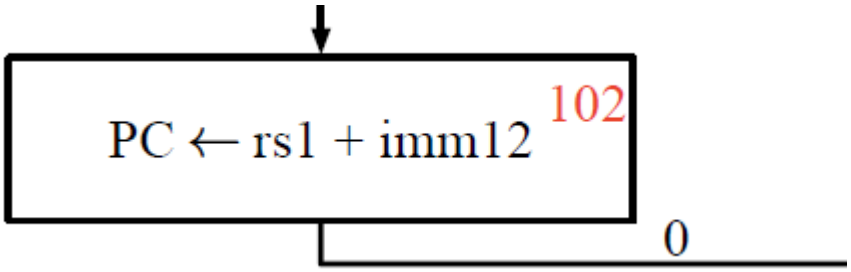
Since in this stage we need to calculate the values by ALU based on the output of rs1 and rs2, so we should enable RS2EN and RS1EN, setting them to 1.

Since the input value of ALU is rs1 and rs2, so in the RS2MUX we should choose the value from RS2, setting the value of RS2MUX to 0.

Since in this stage, no memory operation is needed, so we should set MIO_E and WE to 0. Since we don't select the bitwidth because no memory operation is performed, we set it to 0 by making the multiplexer choose 0 (setting the DATASIZE to 0)

Since in this state we don't reset the RISCV-LC, we do not enable reset, setting RESET to 0.

(g) State 102



Since the next state of this state does not depend on the instruction, we don't need to read from the IR[6:0] and the IRD is set to 0.

Since the next state number is 0, then we should set all J6~J0 to 0 to represent 0.

In this state, we update the value of PC to the value of the sum of rs1 + imm12, therefore,

PC is updated, so LD.PC is 1;

No MAR is updated, so LD.MAR is 0;

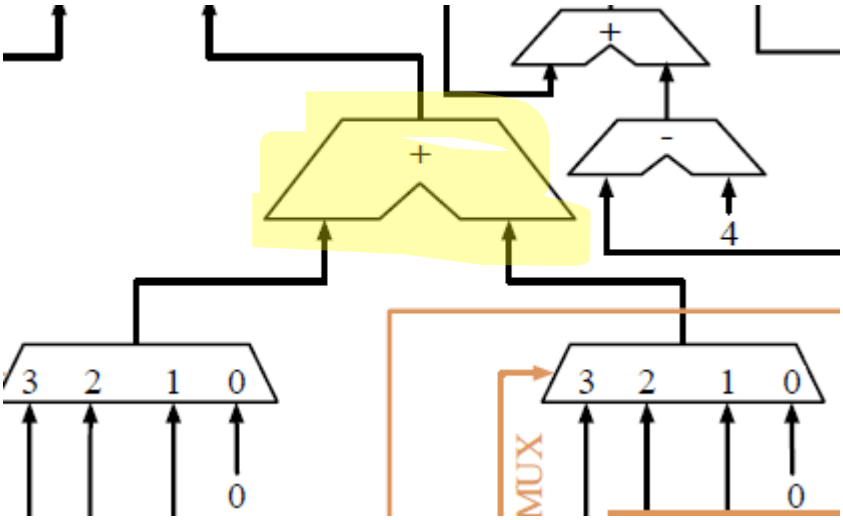
No MDR is updated, so LD.MDR is 0;

No instruction is updated, so LD.IR is 0;

No Register file is updated, so LD.REG is 0.

Since the next state number doesn't depend on the value of B, so in the multiplexer we set LD.BEN to 0 meaning that we won't choose the B value to determine the control signal.

For the tristate driver gates, since in this stage the PC value are calculated by the ALU with input immediate generator and rs1, so it should be the ALU below (highlighted in yellow):



Since the output of this ALU should be transferred to the BUS by GateMAR then to the PC, so only the gate that controls GateMAR value should be enabled and all other values are disabled. Meaning that GatePC, GateALUSHF, GateMDR, GateRS2 are set to 0, and GateMAR is set to 1.

Since the PC is receiving the value from the addition result of imm and rs1 instead of from PC+4, so the PCMUX should be set to 1.

Since the immediate value is the 12 bits immediate from I instruction (for example, jalr), the immediate value should be generated from I Imm Gen, so the ADDR2MUX should select the value of 1, or 01. Since the other value fed into the ALU is rs1, so that the ADDR1MUX should choose the value of 2, or 10.

Since the calculation result is from the ALU highlighted in yellow, the MARMUX should be set to 0.

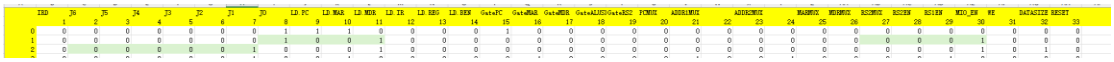
Since we don't need to load anything to the MDR, so in theory we don't need to determine MDRMUX, but in practice, we set it to the default value, which is 0, meaning that MDRMUX=0.

Since we only need the output of rs1 and don't need the output of rs2, we only need to set RS1EN to 1 and set RS2EN to 0.

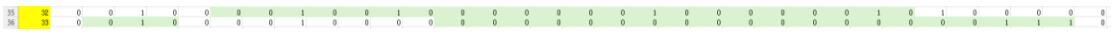
Since in this stage, no memory operation is needed, so we should set MIO_E and WE to 0.

(h) Final result of uop:

State 1 and 2



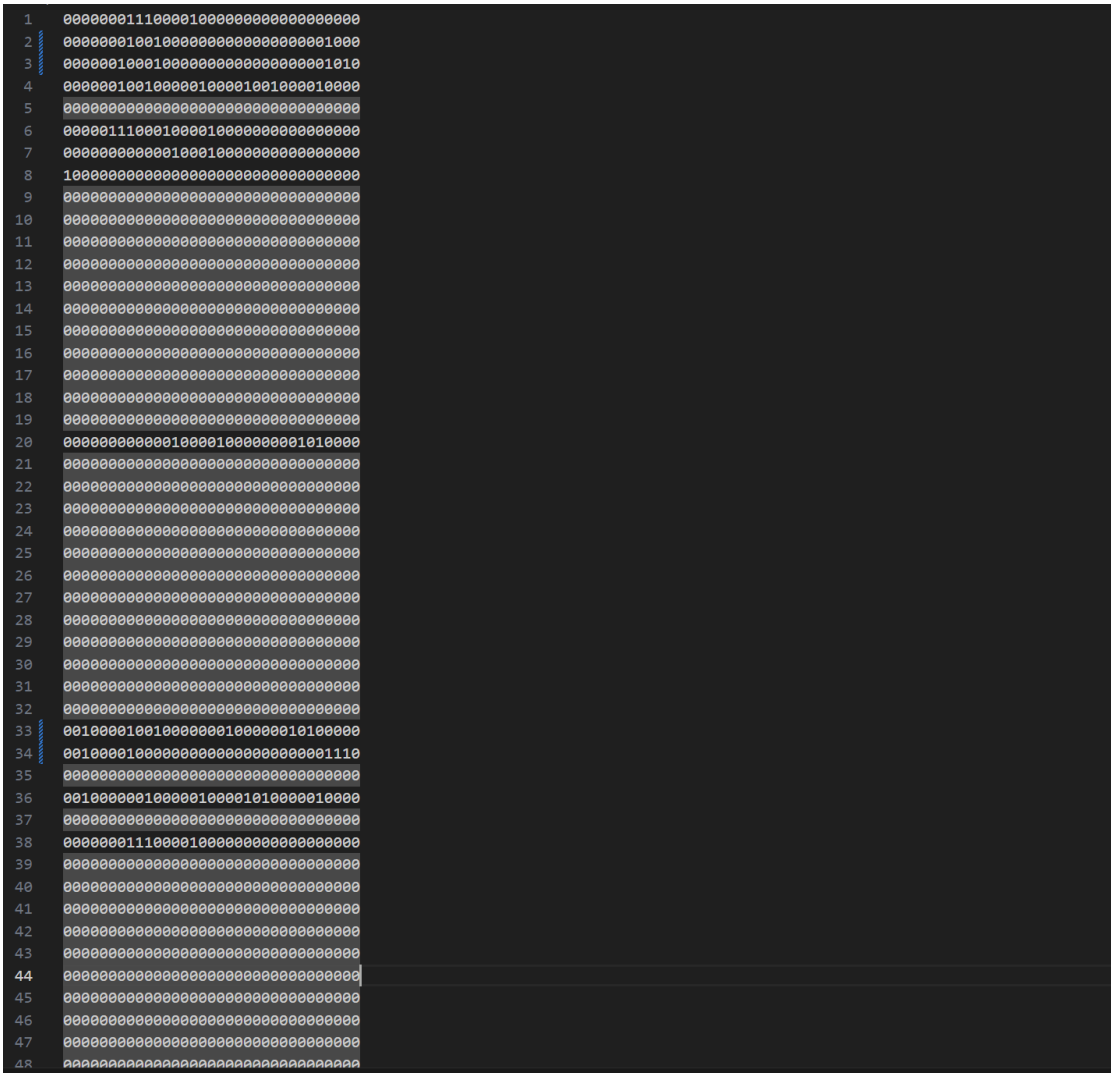
State 32 and 33



State 51

State 102

The final uop file:


00000001110000100000000000000000
00000001001000000000000000001000
00000010001000000000000000001010
00000100100000100001001000010000
00000000000000000000000000000000
00000111000100001000000000000000
00000000000010001000000000000000
10000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000010000100000001010000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00100001001000000010000010100000
00100001000000000000000000001110
00000000000000000000000000000000
00100000010000010000101000010000
00000000000000000000000000000000
00000001110000100000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000

```
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000010000100000000110000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000010010000000100000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
```

```
00000000000000000000000000000000
00000001110000100000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
01100100000010000000000000110000
01100000000010000000000000000000
00000000000000000000000000000000
00000000100000010001100100010000
01100110100010100000000000000000
00000000100000010001110000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
01110000100010100000000000000000
00000000100000010001001100000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
01111111000000000000000000000001

```

6. My mistakes and understandings
   When compiling, I got this mistake:

```
cc: error: ./tools/linux/ubuntu/20.04/libriscv-lc.a: No such file or directory
make: *** [Makefile:39: riscv-lc] Error 1
```

I checked on the piazza, and I found that it is the problem that the version of my ubuntu isn't correct. So I followed this instruction and modified my path and Makefile, and the problem is solved.

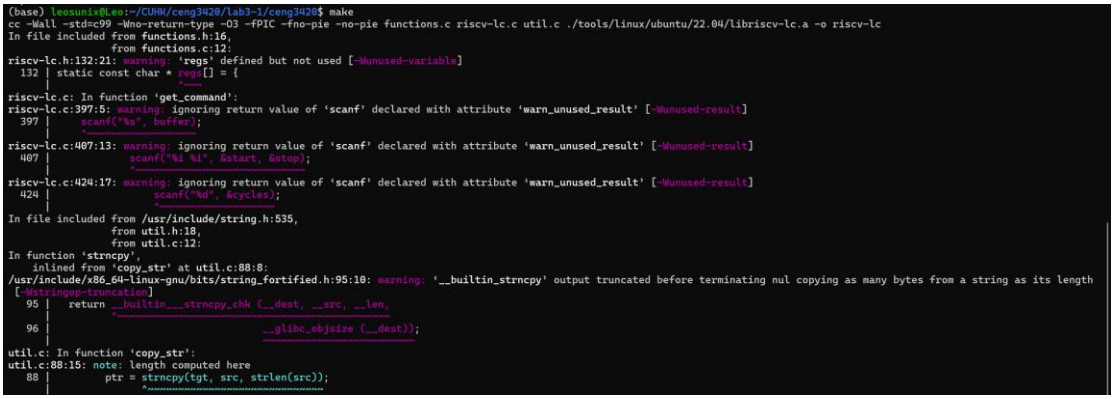
the instructors' answer, *where instructors collectively construct a single answer*

You can copy ./tools/linux/ubuntu/18.04 and rename it to ./tools/linux/ubuntu/22.04

Vincent 5 days ago
I think I have no compile error now. Let me share the file so that it can avoid some mistakes during converting. Thx a lot from my bro for sharing the file. Oh yeah be sure to follow the steps except the Makefile. I think the Makefile is a bit different with what I got from my fri.
riscv-lc.h
Makefile

7. Console results

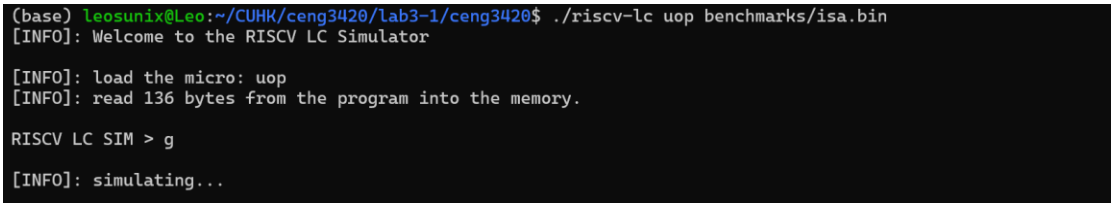
(a) Make:

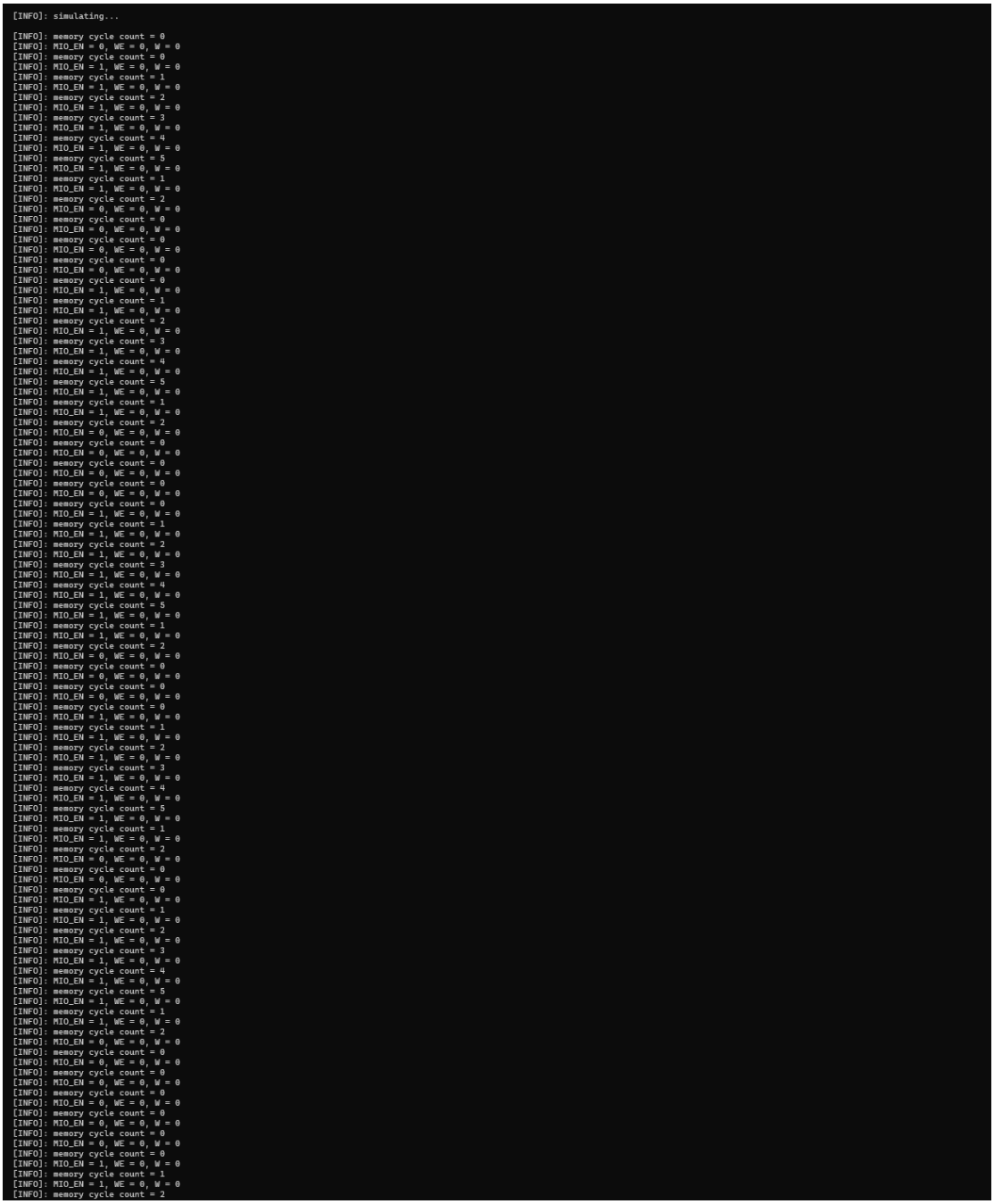


There are some warnings, but it turns out to be some version issues and is about the unsafe use of some string manipulation functions, so we can ignore it in this assignment.

(b) Isa.bin

Running:





```
[INFO]: simulating...

[INFO]: memory cycle count = 0
[INFO]: MIO_EN = 0, WE = 0, W = 0
[INFO]: memory cycle count = 0
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 1
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 2
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 3
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 4
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 5
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 1
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 2
[INFO]: MIO_EN = 0, WE = 0, W = 0
[INFO]: memory cycle count = 0
[INFO]: MIO_EN = 0, WE = 0, W = 0
[INFO]: memory cycle count = 0
[INFO]: MIO_EN = 0, WE = 0, W = 0
[INFO]: memory cycle count = 0
[INFO]: MIO_EN = 0, WE = 0, W = 0
[INFO]: memory cycle count = 0
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 1
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 2
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 3
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 4
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 5
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 1
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 2
[INFO]: MIO_EN = 0, WE = 0, W = 0
[INFO]: memory cycle count = 0
[INFO]: MIO_EN = 0, WE = 0, W = 0
[INFO]: memory cycle count = 0
[INFO]: MIO_EN = 0, WE = 0, W = 0
[INFO]: memory cycle count = 0
[INFO]: MIO_EN = 0, WE = 0, W = 0
[INFO]: memory cycle count = 0
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 1
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 2
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 3
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 4
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 5
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 1
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 2
[INFO]: MIO_EN = 0, WE = 0, W = 0
[INFO]: memory cycle count = 0
[INFO]: MIO_EN = 0, WE = 0, W = 0
[INFO]: memory cycle count = 0
[INFO]: MIO_EN = 0, WE = 0, W = 0
[INFO]: memory cycle count = 0
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 1
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 2
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 3
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 4
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 5
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 1
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 2
[INFO]: MIO_EN = 0, WE = 0, W = 0
[INFO]: memory cycle count = 0
[INFO]: MIO_EN = 0, WE = 0, W = 0
[INFO]: memory cycle count = 0
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 1
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 2
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 3
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 4
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 5
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 1
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 2
[INFO]: MIO_EN = 0, WE = 0, W = 0
[INFO]: memory cycle count = 0
[INFO]: MIO_EN = 0, WE = 0, W = 0
[INFO]: memory cycle count = 0
[INFO]: MIO_EN = 0, WE = 0, W = 0
[INFO]: memory cycle count = 0
[INFO]: MIO_EN = 0, WE = 0, W = 0
[INFO]: memory cycle count = 0
[INFO]: MIO_EN = 0, WE = 0, W = 0
[INFO]: memory cycle count = 0
[INFO]: MIO_EN = 0, WE = 0, W = 0
[INFO]: memory cycle count = 0
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 1
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 2
```

(rest is omitted)

Result:

```
RISCV LC SIM > rd


current register/bus values:
-----------------------------------
cycle count  : 364
PC           : 0x00400000
IR           : 0x0000707f
STATE_NUMBER : 0x0000007f

BUS          : 0x00000000
MDR          : 0x0000707f
MAR          : 0x0000007c
Mem. content : 0x00000000
B            : 0x00000000
READY        : 0x00000000
registers:
zero    [x0]:   0x00000000
ra      [x1]:   0x00000000
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
a3      [x13]:  0xffffffee
a5      [x15]:  0x0000000a
a6      [x16]:  0x0000000d
a7      [x17]:  0x00000068
s2      [x18]:  0x00000000
s3      [x19]:  0x00000000
s4      [x20]:  0x00000000
s5      [x21]:  0x00000000
s6      [x22]:  0x00000000
s7      [x23]:  0x00000000
s8      [x24]:  0x00000000
s9      [x25]:  0x00000000
s10     [x26]:  0x00000000
s11     [x27]:  0x00000000
t3      [x28]:  0x00000000
t4      [x29]:  0x00000000
t5      [x30]:  0x00000000
t6      [x31]:  0x00000000
```

```
RISCV LC SIM > md 0x84 0x94


memory content [0x00000084..0x00000094]:
-----------------------------------
  0x00000084 (132) : 0xfffffff7
  0x00000088 (136) : 0x00000000
  0x0000008c (140) : 0x00000017
  0x00000090 (144) : 0x00000000
  0x00000094 (148) : 0xffffffee

RISCV LC SIM >
```

(c) Count10

Running

```
(base) leosunix@Leo:~/CUHK/ceng3420/lab3-1/ceng3420$ ./riscv-lc uop benchmarks/count10.bin
[INFO]: Welcome to the RISCV LC Simulator

[INFO]: load the micro: uop
[INFO]: read 36 bytes from the program into the memory.

RISCV LC SIM > g

[INFO]: simulating...

[INFO]: memory cycle count = 0
```

```
[INFO]: memory cycle count = 0
[INFO]: MIO_EN = 0, WE = 0, W = 0
[INFO]: memory cycle count = 0
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 1
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 2
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 3
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 4
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 5
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 1
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 2
[INFO]: MIO_EN = 0, WE = 0, W = 0
[INFO]: memory cycle count = 0
[INFO]: MIO_EN = 0, WE = 0, W = 0
[INFO]: memory cycle count = 0
[INFO]: MIO_EN = 0, WE = 0, W = 0
[INFO]: memory cycle count = 0
[INFO]: MIO_EN = 0, WE = 0, W = 0
[INFO]: memory cycle count = 0
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 1
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 2
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 3
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 4
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 5
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 1
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 2
[INFO]: MIO_EN = 0, WE = 0, W = 0
[INFO]: memory cycle count = 0
[INFO]: MIO_EN = 0, WE = 0, W = 0
[INFO]: memory cycle count = 0
[INFO]: MIO_EN = 0, WE = 0, W = 0
[INFO]: memory cycle count = 0
[INFO]: MIO_EN = 0, WE = 0, W = 0
[INFO]: memory cycle count = 0
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 1
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 2
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 3
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 4
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 5
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 1
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 2
[INFO]: MIO_EN = 0, WE = 0, W = 0
[INFO]: memory cycle count = 0
[INFO]: MIO_EN = 0, WE = 0, W = 0
[INFO]: memory cycle count = 0
[INFO]: MIO_EN = 0, WE = 0, W = 0
[INFO]: memory cycle count = 0
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 1
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 2
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 3
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 4
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 5
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 1
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 2
[INFO]: MIO_EN = 0, WE = 0, W = 0
[INFO]: memory cycle count = 0
[INFO]: MIO_EN = 0, WE = 0, W = 0
[INFO]: memory cycle count = 0
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 1
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 2
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 3
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 4
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 5
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 1
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 2
[INFO]: MIO_EN = 0, WE = 0, W = 0
[INFO]: memory cycle count = 0
[INFO]: MIO_EN = 0, WE = 0, W = 0
[INFO]: memory cycle count = 0
[INFO]: MIO_EN = 0, WE = 0, W = 0
[INFO]: memory cycle count = 0
[INFO]: MIO_EN = 0, WE = 0, W = 0
[INFO]: memory cycle count = 0
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 1
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 2
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 3
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 4
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 5
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 1
[INFO]: MIO_EN = 1, WE = 0, W = 0
```

(rest is omitted)

Result

```
RISCV LC SIM > rd


current register/bus values:
-------------------------------------
cycle count  : 412
PC           : 0x00400000
IR           : 0x0000707f
STATE_NUMBER : 0x0000007f

BUS          : 0x00000000
MDR          : 0x0000707f
MAR          : 0x0000001c
Mem. content : 0x00000000
B            : 0x00000000
READY        : 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
a2      [x12]:  0x00000000
a3      [x13]:  0x00000000
a4      [x14]:  0x00000000
a5      [x15]:  0x00000000
a6      [x16]:  0x00000000
a7      [x17]:  0x00000000
s2      [x18]:  0x00000000
s3      [x19]:  0x00000000
s4      [x20]:  0x00000000
s5      [x21]:  0x00000000
s6      [x22]:  0x00000000
s7      [x23]:  0x00000000
s8      [x24]:  0x00000000
s9      [x25]:  0x00000000
s10     [x26]:  0x00000000
s11     [x27]:  0x00000000
t3      [x28]:  0x00000000
t4      [x29]:  0x00000000
t5      [x30]:  0x00000000
t6      [x31]:  0x00000000
```

(d) Swap

Before

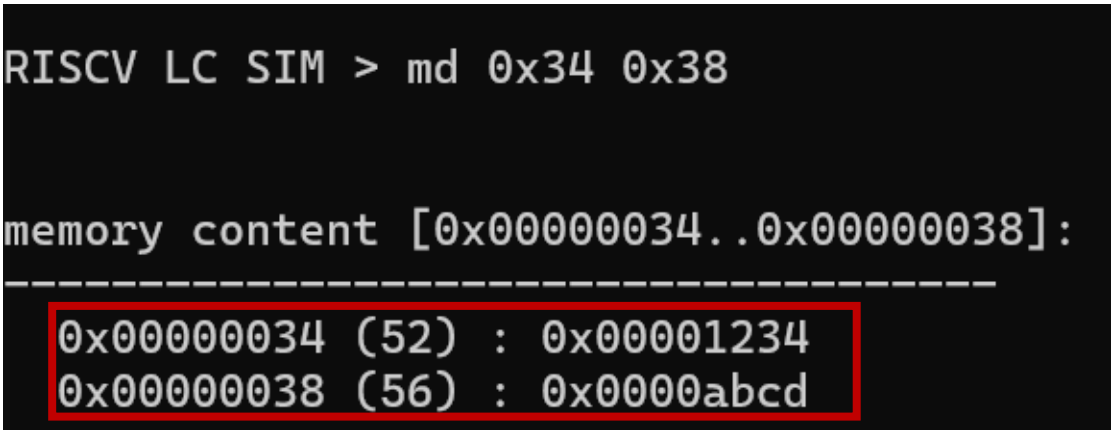
```
(base) leosunix@Leo:~/CUHK/ceng3420/lab3-1/ceng3420$ ./riscv-lc uop benchmarks/swap.bin
[INFO]: Welcome to the RISCV LC Simulator

[INFO]: load the micro: uop
[INFO]: read 60 bytes from the program into the memory.

RISCV LC SIM > mdump 0x34 0x38


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

Running

```
[INFO]: memory cycle count = 0
[INFO]: MIO_EN = 0, WE = 0, W = 0
[INFO]: memory cycle count = 0
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 1
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 2
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 3
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 4
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 5
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 1
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 2
[INFO]: MIO_EN = 0, WE = 0, W = 0
[INFO]: memory cycle count = 0
[INFO]: MIO_EN = 0, WE = 0, W = 0
[INFO]: memory cycle count = 0
[INFO]: MIO_EN = 0, WE = 0, W = 0
[INFO]: memory cycle count = 0
[INFO]: MIO_EN = 0, WE = 0, W = 0
[INFO]: memory cycle count = 0
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 1
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 2
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 3
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 4
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 5
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 1
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 2
[INFO]: MIO_EN = 0, WE = 0, W = 0
[INFO]: memory cycle count = 0
[INFO]: MIO_EN = 0, WE = 0, W = 0
[INFO]: memory cycle count = 0
[INFO]: MIO_EN = 0, WE = 0, W = 0
[INFO]: memory cycle count = 0
[INFO]: MIO_EN = 0, WE = 0, W = 0
[INFO]: memory cycle count = 0
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 1
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 2
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 3
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 4
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 5
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 1
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 2
[INFO]: MIO_EN = 0, WE = 0, W = 0
[INFO]: memory cycle count = 0
[INFO]: MIO_EN = 0, WE = 0, W = 0
[INFO]: memory cycle count = 0
[INFO]: MIO_EN = 0, WE = 0, W = 0
[INFO]: memory cycle count = 0
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 1
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 2
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 3
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 4
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 5
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 1
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 2
[INFO]: MIO_EN = 0, WE = 0, W = 0
[INFO]: memory cycle count = 0
[INFO]: MIO_EN = 0, WE = 0, W = 0
[INFO]: memory cycle count = 0
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 1
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 2
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 3
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 4
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 5
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 1
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 2
[INFO]: MIO_EN = 0, WE = 0, W = 0
[INFO]: memory cycle count = 0
[INFO]: MIO_EN = 0, WE = 0, W = 0
[INFO]: memory cycle count = 0
[INFO]: MIO_EN = 0, WE = 0, W = 0
[INFO]: memory cycle count = 0
[INFO]: MIO_EN = 0, WE = 0, W = 0
[INFO]: memory cycle count = 0
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 1
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 2
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 3
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 4
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 5
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 1
[INFO]: MIO_EN = 1, WE = 0, W = 0
```

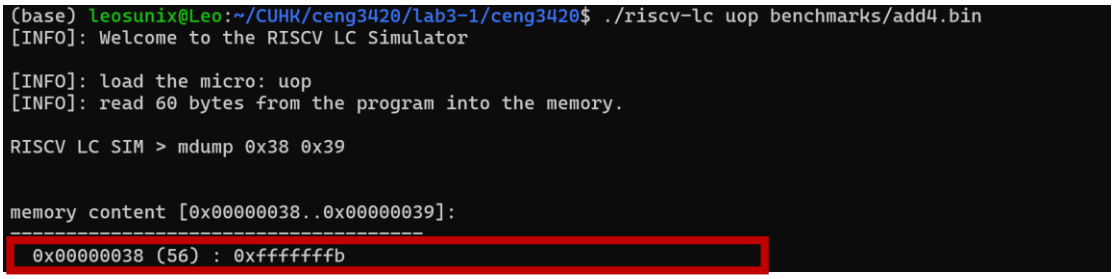
(rest is omitted)

Result:



(e) Add4

Before:

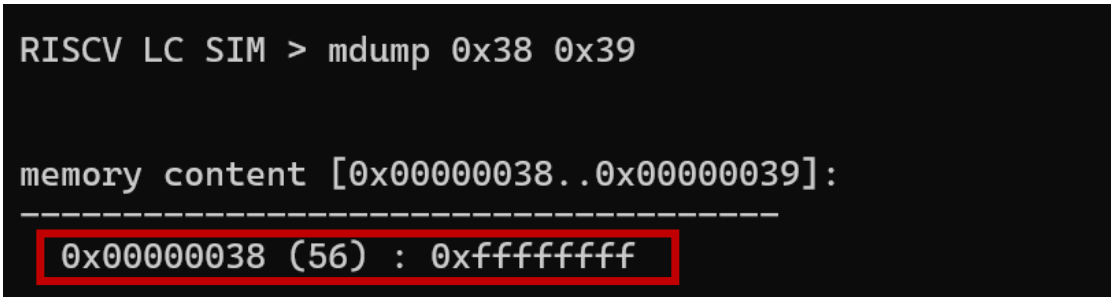


Running:

```
[INFO]: memory cycle count = 0
[INFO]: MIO_EN = 0, WE = 0, W = 0
[INFO]: memory cycle count = 0
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 1
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 2
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 3
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 4
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 5
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 1
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 2
[INFO]: MIO_EN = 0, WE = 0, W = 0
[INFO]: memory cycle count = 0
[INFO]: MIO_EN = 0, WE = 0, W = 0
[INFO]: memory cycle count = 0
[INFO]: MIO_EN = 0, WE = 0, W = 0
[INFO]: memory cycle count = 0
[INFO]: MIO_EN = 0, WE = 0, W = 0
[INFO]: memory cycle count = 0
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 1
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 2
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 3
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 4
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 5
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 1
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 2
[INFO]: MIO_EN = 0, WE = 0, W = 0
[INFO]: memory cycle count = 0
[INFO]: MIO_EN = 0, WE = 0, W = 0
[INFO]: memory cycle count = 0
[INFO]: MIO_EN = 0, WE = 0, W = 0
[INFO]: memory cycle count = 0
[INFO]: MIO_EN = 0, WE = 0, W = 0
[INFO]: memory cycle count = 0
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 1
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 2
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 3
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 4
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 5
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 1
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 2
[INFO]: MIO_EN = 0, WE = 0, W = 0
[INFO]: memory cycle count = 0
[INFO]: MIO_EN = 0, WE = 0, W = 0
[INFO]: memory cycle count = 0
[INFO]: MIO_EN = 0, WE = 0, W = 0
[INFO]: memory cycle count = 0
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 1
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 2
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 3
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 4
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 5
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 1
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 2
[INFO]: MIO_EN = 0, WE = 0, W = 0
[INFO]: memory cycle count = 0
[INFO]: MIO_EN = 0, WE = 0, W = 0
[INFO]: memory cycle count = 0
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 1
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 2
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 3
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 4
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 5
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 1
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 2
[INFO]: MIO_EN = 0, WE = 0, W = 0
[INFO]: memory cycle count = 0
[INFO]: MIO_EN = 0, WE = 0, W = 0
[INFO]: memory cycle count = 0
[INFO]: MIO_EN = 0, WE = 0, W = 0
[INFO]: memory cycle count = 0
[INFO]: MIO_EN = 0, WE = 0, W = 0
[INFO]: memory cycle count = 0
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 1
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 2
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 3
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 4
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 5
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 1
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 2
```

(rest is omitted)

Result:



Lab 3.2 Report

1. Problem analysis

   In lab 3.1 we are implementing the simulator of RISCV-LC, especially the memory operation part and the part that sends the value via bus.

2. My understandings of the RISCV-LC simulator structure

   (a) My understanding of the instruction execution process:

   Just as in lab 3.1, in the cycle instruction, the core steps of the simulator are executed. In lab 3.2, we assume that other parts are already implemented, and we mainly focus on cycle_memory and latch_datapath_values.

3. My understanding of some key implementations in the code

   (a) Some commonly used functions and macros

   (ii) datasize_mux ():

   This function emulates the data size multiplexer. The goal of it is to tell what the size of the data that the memory operation should operate on.

   If the multiplexer chooses 0 (DATASIZE==0), then it will directly return 0. Else it will return the value based on the func3 of the instructions.

   For memory operations. lb, lbu and sb are the instructions that are operating on bytes. Their func3 are 0x0, 0x4, 0x0 (0b000, 0b100, 0b000) respectively. So after the operation ~(funct3 & 0x3), the return value is always 0b 1111 1111 1111 1111 1111 1111 1111 1111, or -1.

   For memory operations, lh, lhu, sh are the instructions that are operating on 2 bytes. Their func3 are 0x1, 0x5, 0x1 (0b001, 0x101, 0b001) respectively. So after the operation ~(funct3 & 0x3), the return value is always 0b 1111 1111 1111 1111 1111 1111 1111 1110, or -2.

   For memory operations, lw, sw are the instructions that are operating on 4 bytes. Their func3 are 0x2 (0b10) respectively. So after the operation ~(funct3 & 0x3), the return value is always 0b 1111 1111 1111 1111 1111 1111 1111 1101, or -3.

   (iii) ADDRnMUX

   Functions addr1_mux and addr2_mux emulates the multiplexer ADDR1MUX and ADDR2MUX. The input is the potential output values of each multiplexers and the control value. The control value can be retrieved by the MACRO get_ADDR1MUX and get_ADDR2MUX.

   (c) Some key variables

MEM_VAL: the temporary variable that stores the memory after each cycle finishes.

W: the indicator for writing enabled. If mio & W then it means we should write to the memory; if mio & !W then it means that we should do memory IO but we shouldn't do memory write, which means we should read the memory.

BUS: the value of bus.

(d) Handle cycle_memory()

cycle_memory() is the function to handle the operations for writing or reading to the main memory.

(e) Writing

As discussed above, when W is true, we should write.

First, we should get the value of funct3, because the funct3 decides how many bytes we should manipulate. This can be done by using the mask_val function to retrieve the 14..12 bits of the instructions.

Then by utilizing the datasize_mux discussed above, with control signal get_DATASIZE(CURRENT_LATCHES.MICROINSTRUCTION), and input funct3, 0, we can map the result to how many bytes we should work on. Since it is little endian, the higher bits should be placed in higher addresses.

(i) If the result is -1, we should write only one byte.

The first byte is the bits 7..0 of MDR, and the destination address is CURRENT_LATCHES.MAR, so we can use MASK7_0() to retrieve it, and assign it to MEMORY[CURRENT_LATCHES.MAR].

(ii) If the result is -2, we should write the first 2 bytes.

The first byte is the bits 7..0 of MDR, and the destination address is CURRENT_LATCHES.MAR, so we can use MASK7_0() to retrieve it, and assign it to MEMORY[CURRENT_LATCHES.MAR].

The second byte is the bits 15..8 of MDR, and the destination address is CURRENT_LATCHES.MAR + 1, so we can use MASK15_8() to retrieve it, and assign it to MEMORY[CURRENT_LATCHES.MAR+1].

(iii) Otherwise, we should write all 4 bytes

The first byte is the bits 7..0 of MDR, and the destination address is CURRENT_LATCHES.MAR, so we can use MASK7_0() to retrieve it, and assign it to MEMORY[CURRENT_LATCHES.MAR].

The second byte is the bits 15..8 of MDR, and the destination address is CURRENT_LATCHES.MAR + 1, so we can use MASK15_8() to retrieve it, and assign it to MEMORY[CURRENT_LATCHES.MAR+1].

The third byte is the bits 23..16 of MDR, and the destination address is CURRENT_LATCHES.MAR+2, so we can use MASK23_116() to retrieve it, and assign it to MEMORY[CURRENT_LATCHES.MAR+2].

The fourth byte is the bits 31..24 of MDR, and the destination address is CURRENT_LATCHES.MAR + 3, so we can use MASK31_24() to retrieve it, and assign it to MEMORY[CURRENT_LATCHES.MAR+3].

The implementation is shown below:

```
/* write */
/*
 * Lab3-2 assignment DONE
 */
// error("Lab3-2 assignment: write to the main memory");
int funct3 = mask_val(CURRENT_LATCHES.IR, 14, 12);
switch(datasize_mux(get_DATASIZE(CURRENT_LATCHES.MICROINSTRUCTION), funct3, 0)){
    case -1:        // byte
        MEMORY[CURRENT_LATCHES.MAR] = MASK7_0(CURRENT_LATCHES.MDR);
        break;
    case -2:        // half
        MEMORY[CURRENT_LATCHES.MAR] = MASK7_0(CURRENT_LATCHES.MDR);
        MEMORY[CURRENT_LATCHES.MAR + 1] = MASK15_8(CURRENT_LATCHES.MDR);
        break;
    default:        // word
        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);
}
```

(f) Reading

If W is false, then we should do reading. Same as writing, there are 3 situations of the output of datasize_mux. Since it is little endian, the higher bits should be retrieved in higher addresses.

(i) If the result is -1, we should read only one byte.

The byte is MEMORY[CURRENT_LATCHES.MAR], and we can use sext_unit() to sign extend it.

Then we can assign this value to MEM_VAL.

(ii) If the result is -2, we should read 2 bytes.

The first byte is MEMORY[CURRENT_LATCHES.MAR], the second byte is MEMORY[CURRENT_LATCHES.MAR+1], so the value is (MEMORY[CURRENT_LATCHES.MAR]) + (MEMORY[CURRENT_LATCHES.MAR + 1] << 8. And we can use sext_unit() to sign extend it.

Then we can assign this value to MEM_VAL.

(iii) Otherwise, we should read 4 bytes.

The first byte is MEMORY[CURRENT_LATCHES.MAR], the second byte is MEMORY[CURRENT_LATCHES.MAR+1], The third byte is MEMORY[CURRENT_LATCHES.MAR+2], the fourth byte is MEMORY[CURRENT_LATCHES.MAR+3], so the value is MEMORY[CURRENT_LATCHES.MAR]) + (MEMORY[CURRENT_LATCHES.MAR + 1] << 8) + (MEMORY[CURRENT_LATCHES.MAR + 2] << 16) + (MEMORY[CURRENT_LATCHES.MAR + 3] << 24. And we can use sext_unit() to sign extend it.

Then we can assign this value to MEM_VAL.

```
/* read */
/*
 * Lab3-2 assignment DONE
 * Tips: assign the read value to `MEM_VAL`
 */
// error("Lab3-2 assignment: read from the main memory");
int funct3 = mask_val(CURRENT_LATCHES.IR, 14, 12);

switch(datasize_mux(get_DATASIZE(CURRENT_LATCHES.MICROINSTRUCTION), funct3, 0)){
    case -1:    // byte
        MEM_VAL = sext_unit(MEMORY[CURRENT_LATCHES.MAR], 8);
        break;
    case -2:    // half
        MEM_VAL = sext_unit((MEMORY[CURRENT_LATCHES.MAR]) + (MEMORY[CURRENT_LATCHES.MAR + 1] << 8), 16);
        break;
    default:    // word
        MEM_VAL = (MEMORY[CURRENT_LATCHES.MAR]) + (MEMORY[CURRENT_LATCHES.MAR + 1] << 8) + (MEMORY[CURRENT_LATCHES.MAR + 2] << 16) + (MEMORY[CURRENT_LATCHES.MAR + 3] << 24);
}
```

(g) Handle latch_datapath_values()

In this function, we update all the blocks with values from the bus according to LD signals.

(iv) Update register file

First we need to retrieve the register index. Since rd is in bits 11..7, we can use mask_val(CURRENT_LATCHES.IR, 11, 7) to get the index.

Then, we can set the corresponding register to the bus value, by NEXT_LATCHES.REGS[reg_idx] = BUS.

The implementation is shown below:

```
    /*
     *  Lab3-2 assignment DONE
     */
    // error("Lab3-2 assignment: handle LD_REG");
    int reg_idx = mask_val(CURRENT_LATCHES.IR, 11, 7);
    NEXT_LATCHES.REGS[reg_idx] = BUS;
```

(v) Update MAR

The update of MAR is simple, as long as the LD.MAR bit is true, we just assign the bus value to MAR.

The implementation is shown below:

```
/* LD.MAR */
if (get_LD_MAR(CURRENT_LATCHES.MICROINSTRUCTION)) {
    /*
     *  Lab3-2 assignment DONE
     */
    // error("Lab3-2 assignment: handle LD_MAR");
    NEXT_LATCHES.MAR = BUS;
}
```

(vi) Update IR

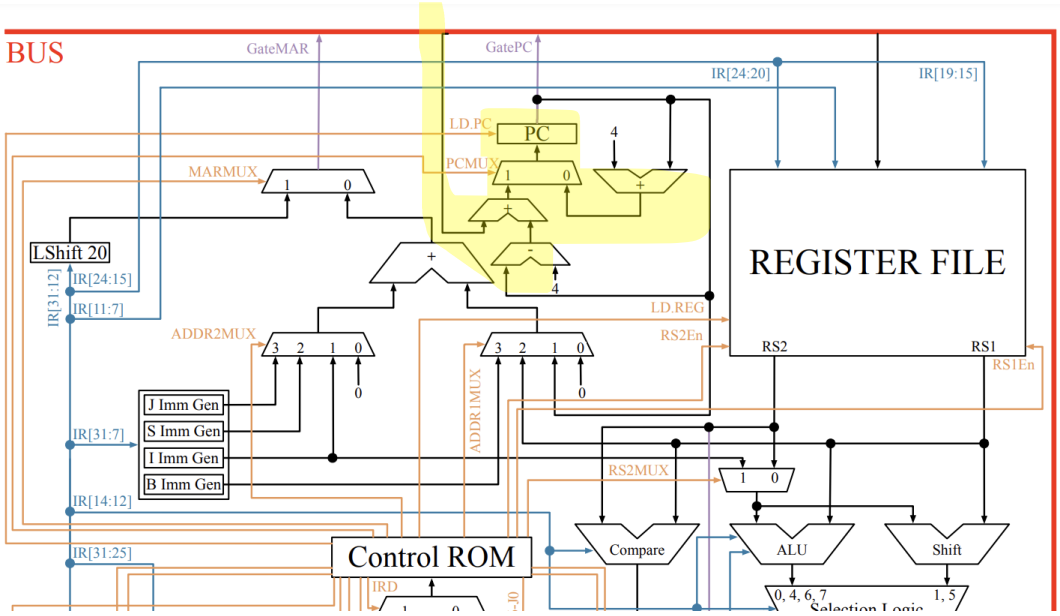
The update of IR is simple, as long as the LD.IR bit is true, we just assign the bus value to IR.

The implementation is shown below:

```
/* LD.IR */
if (get_LD_IR(CURRENT_LATCHES.MICROINSTRUCTION)) {
    /*
     *  Lab3-2 assignment DONE
     */
    // error("Lab3-2 assignment: handle LD_IR");
    NEXT_LATCHES.IR = BUS;
}
```

(vii) Update PC

The update of PC is more complex. From the graph we can see that, there are 2 sources from the PC MUX:



The first source is the value of PC+4, since it is read when PCMUX is 0, I call it pc_mux_0_value. The other value is the sum of the BUS value and PC-4, since it is read when PCMUX is 1, I call it pc_mux_1_value. These 2 values are chosen depending on the control value of LD.PC.

So the PC value can be retrieved by the function pc_mux, with parameters get_PCMUX(CURRENT_LATCHES.MICROINSTRUCTION), pc_mux_0_valueand pc_mux_1_value.

The implementation is shown below:

```
if (get_LD_PC(CURRENT_LATCHES.MICROINSTRUCTION)) {
    /*
     *  Lab3-2 assignment DONE
     */
    // error("Lab3-2 assignment: handle LD_PC");

    // cal pc_mux_0_value
    int pc_mux_0_value = CURRENT_LATCHES.PC + 4;
    int pc_minus_4 = CURRENT_LATCHES.PC - 4;
    int pc_mux_1_value = BUS + pc_minus_4;

    // calculate the final value
    NEXT_LATCHES.PC = pc_mux(get_PCMUX(CURRENT_LATCHES.MICROINSTRUCTION), pc_mux_0_value, pc_mux_1_value);

}
```

(h) My mistakes and understandings

One mistake I had when doing the lab is that I missed considering the endianness of the data, by little endian rule, the higher bits should be placed in higher addresses, but I got it reversed. Resulting in mistakes.

When compiling, I got this mistake:

```
cc: error: ./tools/linux/ubuntu/20.04/libriscv-lc.a: No such file or directory
make: *** [Makefile:39: riscv-lc] Error 1
```

I checked on the piazza, and I found that it is the problem that the version of my ubuntu isn't correct. So I followed this instruction and modified my path and Makefile, and the problem is solved.

the instructors' answer, *where instructors collectively construct a single answer*

You can copy ./tools/linux/ubuntu/18.04 and rename it to ./tools/linux/ubuntu/22.04

Vincent 5 days ago

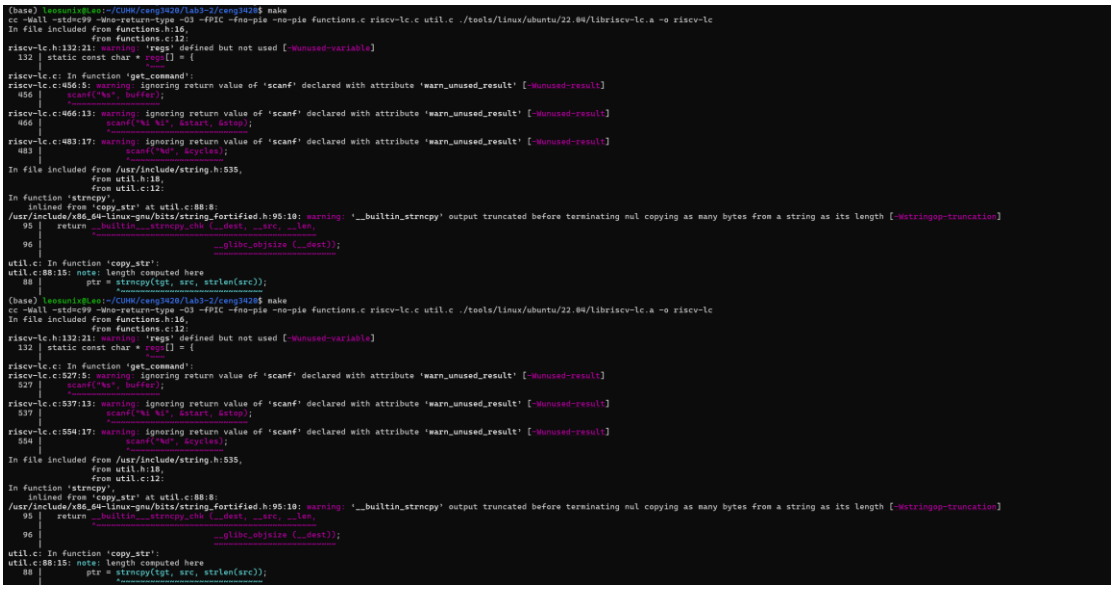
I think I have no compile error now. Let me share the file so that it can avoid some mistakes during converting. Thx a lot from my bro for sharing the file. Oh yeah be sure to follow the steps except the Makefile. I think the Makefile is a bit different with what I got from my fri.

riscv-lc.h

Makefile

4. Console results

(a) Make:

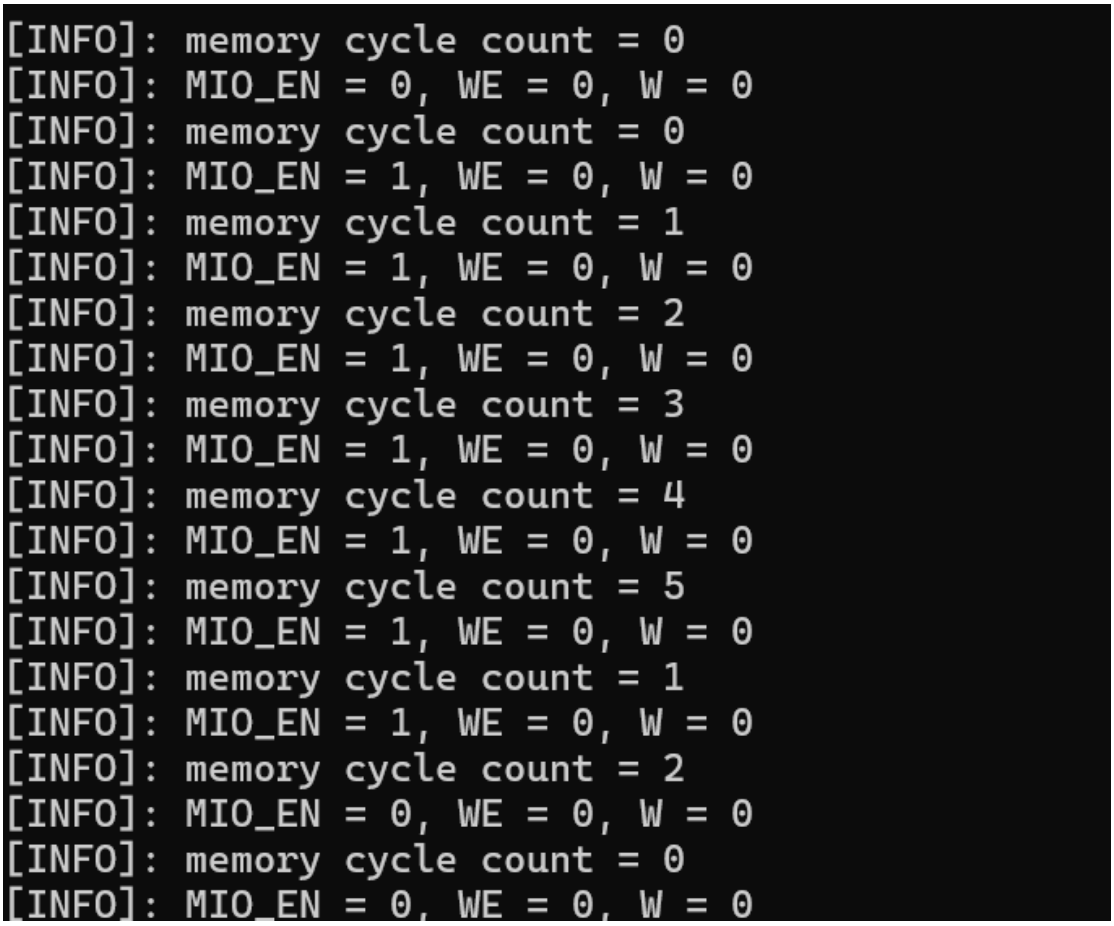


There are some warnings, but it turns out to be some version issues and is about the unsafe use of some string manipulation functions, so we can ignore it in this assignment.

(f) Isa.bin

Running:

```
(base) leosunix@Leo:~/CUHK/ceng3420/lab3-2/ceng3420$ ./riscv-lc uop benchmarks/isa.bin
[INFO]: Welcome to the RISCV LC Simulator

[INFO]: load the micro: uop
[INFO]: read 136 bytes from the program into the memory.

RISCV LC SIM > g

[INFO]: simulating...
```



(rest is omitted)

Result:

```
RISCV LC SIM > rd


current register/bus values:
-----------------------------------
cycle count  : 364
PC           : 0x00400000
IR           : 0x0000707f
STATE_NUMBER : 0x0000007f

BUS          : 0x00000000
MDR          : 0x0000707f
MAR          : 0x0000007c
Mem. content : 0x00000000
B            : 0x00000000
READY        : 0x00000000
registers:
zero    [x0]:   0x00000050
ra      [x1]:   0x00000000
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
s3      [x19]:  0x00000000
s4      [x20]:  0x00000000
s5      [x21]:  0x00000000
s6      [x22]:  0x00000000
s7      [x23]:  0x00000000
s8      [x24]:  0x00000000
s9      [x25]:  0x00000000
s10     [x26]:  0x00000000
s11     [x27]:  0x00000000
t3      [x28]:  0x00000000
t4      [x29]:  0x00000000
t5      [x30]:  0x00000000
t6      [x31]:  0x00000000
```

```
RISCV LC SIM > md 0x84 0x94


memory content [0x00000084..0x00000094]:
-------------------------------------
  0x00000084 (132) : 0xfffffff7
  0x00000088 (136) : 0x00000000
  0x0000008c (140) : 0x00000017
  0x00000090 (144) : 0x00000000
  0x00000094 (148) : 0xffffffee
```

(g) Count10

Running:

```
(base) leosunix@Leo:~/CUHK/ceng3420/lab3-2/ceng3420$ ./riscv-lc uop benchmarks/count10.bin
[INFO]: Welcome to the RISCV LC Simulator

[INFO]: load the micro: uop
[INFO]: read 36 bytes from the program into the memory.

RISCV LC SIM > g

[INFO]: simulating...

[INFO]: memory cycle count = 0
[INFO]: MIO_EN = 0, WE = 0, W = 0
[INFO]: memory cycle count = 0
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 1
[INFO]: MIO_EN = 1, WE = 0, W = 0
```

(rest is omitted)

Result:

```
RISCV LC SIM > rd


current register/bus values:
-------------------------------------
cycle count  : 412
PC           : 0x00400000
IR           : 0x0000707f
STATE_NUMBER : 0x0000007f

BUS          : 0x00000000
MDR          : 0x0000707f
MAR          : 0x0000001c
Mem. content : 0x00000000
B            : 0x00000000
READY        : 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
a2      [x12]:  0x00000000
a3      [x13]:  0x00000000
a4      [x14]:  0x00000000
a5      [x15]:  0x00000000
a6      [x16]:  0x00000000
a7      [x17]:  0x00000000
s2      [x18]:  0x00000000
s3      [x19]:  0x00000000
s4      [x20]:  0x00000000
s5      [x21]:  0x00000000
s6      [x22]:  0x00000000
s7      [x23]:  0x00000000
s8      [x24]:  0x00000000
s9      [x25]:  0x00000000
s10     [x26]:  0x00000000
s11     [x27]:  0x00000000
t3      [x28]:  0x00000000
t4      [x29]:  0x00000000
t5      [x30]:  0x00000000
t6      [x31]:  0x00000000
```

(h) Swap

Before:

```
(base) leosunix@Leo:~/CUHK/ceng3420/lab3-2/ceng3420$ ./riscv-lc uop benchmarks/swap.bin
[INFO]: Welcome to the RISCV LC Simulator

[INFO]: load the micro: uop
[INFO]: read 60 bytes from the program into the memory.

RISCV LC SIM > md 0x34 0x38


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

Running:

```
[INFO]: simulating...

[INFO]: memory cycle count = 0
[INFO]: MIO_EN = 0, WE = 0, W = 0
[INFO]: memory cycle count = 0
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 1
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 2
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 3
[INFO]: MIO_EN = 1, WE = 0, W = 0
```

(rest is omitted)

Result:

```
RISCV LC SIM > md 0x34 0x38


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

(i) Add4

Before

```
(base) leosunix@Leo:~/CUHK/ceng3420/lab3-2/ceng3420$ ./riscv-lc uop benchmarks/add4.bin
[INFO]: Welcome to the RISCV LC Simulator

[INFO]: load the micro: uop
[INFO]: read 60 bytes from the program into the memory.

RISCV LC SIM > md 0x38 0x39


memory content [0x00000038..0x00000039]:
--------------------------------------
  0x00000038 (56) : 0xfffffffb
```

running:

```
RISCV LC SIM > g

[INFO]: simulating...

[INFO]: memory cycle count = 0
[INFO]: MIO_EN = 0, WE = 0, W = 0
[INFO]: memory cycle count = 0
[INFO]: MIO_EN = 1, WE = 0, W = 0
```

(rest is omitted)

Result:

```
memory content [0x00000038..0x00000039]:
--------------------------------------
  0x00000038 (56) : 0xffffffff
```

Lab 3.3

1. Console results
   (a) Make:

```
(base) leosunix@Leo:/mnt/d/OneDrive/OneDrive - The Chinese University of Hong Kong/CUHK/y2s2/CENG3420/lab3/lab3-3/lab3-3_checked$ make
cc -Wall -std=c99 -Wno-return-type -O3 -fPIC -fno-pie -no-pie functions.c riscv-lc.c util.c ./tools/linux/ubuntu/22.04/libriscv-lc.a -o riscv-lc
In file included from functions.h:16,
                 from functions.c:12:
riscv-lc.h:132:21: warning: 'regs' defined but not used [-Wunused-variable]
  132 | static const char * regs[] = {
      |                     ^~~~
riscv-lc.c: In function 'get_command':
riscv-lc.c:525:5: warning: ignoring return value of 'scanf' declared with attribute 'warn_unused_result' [-Wunused-result]
  525 |     scanf("%s", buffer);
      |     ^~~~~~~~~~~~~~~~~~~
riscv-lc.c:535:13: warning: ignoring return value of 'scanf' declared with attribute 'warn_unused_result' [-Wunused-result]
  535 |             scanf("%i %i", &start, &stop);
      |             ^~~~~~~~~~~~~~~~~~~~~~~~~~~~~
riscv-lc.c:552:17: warning: ignoring return value of 'scanf' declared with attribute 'warn_unused_result' [-Wunused-result]
  552 |                 scanf("%d", &cycles);
      |                 ^~~~~~~~~~~~~~~~~~~~
In file included from /usr/include/string.h:535,
                 from util.h:18,
                 from util.c:12:
In function 'strncpy',
    inlined from 'copy_str' at util.c:88:8:
/usr/include/x86_64-linux-gnu/bits/string_fortified.h:95:10: warning: '__builtin_strncpy' output truncated before terminating nul copying as many bytes from a string as its length
 [-Wstringop-truncation]
   95 |   return __builtin___strncpy_chk (__dest, __src, __len,
      |          ^~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
   96 |                                   __glibc_objsize (__dest));
      |                                   ~~~~~~~~~~~~~~~~~~~~~~~~~
util.c: In function 'copy_str':
util.c:88:15: note: length computed here
   88 |         ptr = strncpy(tgt, src, strlen(src));
      |               ^~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

   (b) lsa.bin
   Running:

```
/y2s2/CENG3420/lab3/lab3-2/ceng3420_checked$ ./riscv-lc uop benchmarks/swap.bin
[INFO]: Welcome to the RISCV LC Simulator

[INFO]: load the micro: uop
[INFO]: read 60 bytes from the program into the memory.

RISCV LC SIM > md 0x34 0x38


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

RISCV LC SIM > g

[INFO]: simulating...

[INFO]: memory cycle count = 0
[INFO]: MIO_EN = 0, WE = 0, W = 0
[INFO]: memory cycle count = 0
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 1
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 2
```

   (rest is omitted)
   Result:

```
RISCV LC SIM > rd


current register/bus values:
-------------------------------------
cycle count  : 364
PC           : 0x00400000
IR           : 0x0000707f
STATE_NUMBER : 0x0000007f

BUS          : 0x00000000
MDR          : 0x0000707f
MAR          : 0x0000007c
Mem. content : 0x00000000
B            : 0x00000000
READY        : 0x00000000
registers:
zero    [x0]:   0x00000050
ra      [x1]:   0x00000000
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
s3      [x19]:  0x00000000
s4      [x20]:  0x00000000
s5      [x21]:  0x00000000
s6      [x22]:  0x00000000
s7      [x23]:  0x00000000
s8      [x24]:  0x00000000
s9      [x25]:  0x00000000
s10     [x26]:  0x00000000
s11     [x27]:  0x00000000
t3      [x28]:  0x00000000
t4      [x29]:  0x00000000
t5      [x30]:  0x00000000
t6      [x31]:  0x00000000
```

```
RISCV LC SIM > md 0x84 0x94


memory content [0x00000084..0x00000094]:
-----------------------------------
  0x00000084 (132) : 0xfffffff7
  0x00000088 (136) : 0x00000000
  0x0000008c (140) : 0x00000017
  0x00000090 (144) : 0x00000000
  0x00000094 (148) : 0xffffffee
```

(c) Count10:

Running:

```
RISCV LC SIM > g

[INFO]: simulating...

[INFO]: memory cycle count = 0
[INFO]: MIO_EN = 0, WE = 0, W = 0
[INFO]: memory cycle count = 0
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 1
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 2
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 3
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 4
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 5
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 1
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 2
[INFO]: MIO_EN = 0, WE = 0, W = 0
[INFO]: memory cycle count = 0
[INFO]: MIO_EN = 0, WE = 0, W = 0
[INFO]: memory cycle count = 0
[INFO]: MIO_EN = 0, WE = 0, W = 0
[INFO]: memory cycle count = 0
[INFO]: MIO_EN = 0, WE = 0, W = 0
[INFO]: memory cycle count = 0
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 1
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 2
```

(rest is omitted)

Result:

```
RISCV LC SIM > rd


current register/bus values:
-----------------------------------
cycle count  : 412
PC           : 0x00400000
IR           : 0x0000707f
STATE_NUMBER : 0x0000007f

BUS          : 0x00000000
MDR          : 0x0000707f
MAR          : 0x0000001c
Mem. content : 0x00000000
B            : 0x00000000
READY        : 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
a2      [x12]:  0x00000000
a3      [x13]:  0x00000000
a4      [x14]:  0x00000000
a5      [x15]:  0x00000000
a6      [x16]:  0x00000000
a7      [x17]:  0x00000000
s2      [x18]:  0x00000000
s3      [x19]:  0x00000000
s4      [x20]:  0x00000000
s5      [x21]:  0x00000000
s6      [x22]:  0x00000000
s7      [x23]:  0x00000000
s8      [x24]:  0x00000000
s9      [x25]:  0x00000000
s10     [x26]:  0x00000000
s11     [x27]:  0x00000000
t3      [x28]:  0x00000000
t4      [x29]:  0x00000000
t5      [x30]:  0x00000000
t6      [x31]:  0x00000000
```

(a) Swap

Before:

```
RISCV LC SIM > md 0x34 0x38


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

Running:

```
RISCV LC SIM > g

[INFO]: simulating...

[INFO]: memory cycle count = 0
[INFO]: MIO_EN = 0, WE = 0, W = 0
[INFO]: memory cycle count = 0
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 1
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 2
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 3
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 4
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 5
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 1
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 2
```

(rest is omitted)

Result:

```
RISCV LC SIM > md 0x34 0x38


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

(b) Add4

Before:

```
RISCV LC SIM > md 0x38 0x39


memory content [0x00000038..0x00000039]:
-------------------------------------
  0x00000038 (56) : 0xfffffffb
```

Running:

```
RISCV LC SIM > g

[INFO]: simulating...

[INFO]: memory cycle count = 0
[INFO]: MIO_EN = 0, WE = 0, W = 0
[INFO]: memory cycle count = 0
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 1
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 2
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 3
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 4
[INFO]: MIO_EN = 1, WE = 0, W = 0
[INFO]: memory cycle count = 5
```

(rest is omitted)

Result:

```
RISCV LC SIM > md 0x38 0x39


memory content [0x00000038..0x00000039]:
-------------------------------------
  0x00000038 (56) : 0xffffffff
```

Reference:

TextBook -Computer Organization and Design_ The Hardware Software Interface [RISC-V Edition]

opcodes-rv32i reference document

risc-v-asm-manual.pdf

riscv-spec-20191213.pdf

fsm.pdf

riscv-lc.pdf