CENG 3420 Lab 2-1 Report

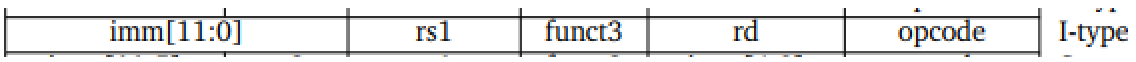
Name: Chan Chun Ming SID: 1155163257

Lab 2.1

The aim of this lab is to implement a RISC-V assembler, which convert assembly language code into machine code.

Integer register-immediate Instructions

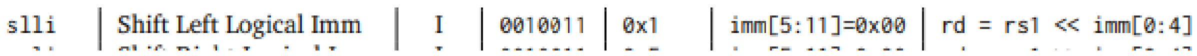
The assignment starts with processing integer registers-immediate instructions, which includes slli, xori, srli, srai, ori, andi, lui. They are all I-format instructions.



imm[11:0]	rs1	funct3	rd	opcode	I-type

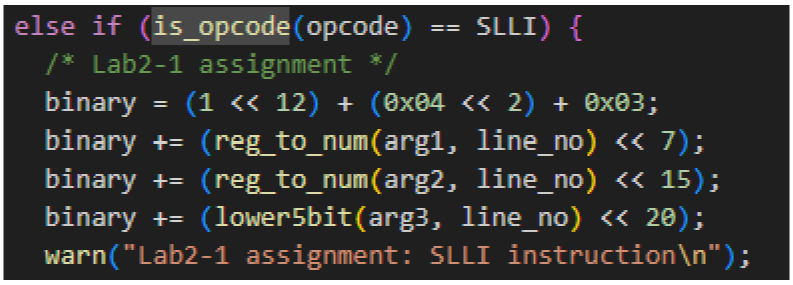
So, to implement these instructions, I followed the details of instruction to add the field to the binary stream.

Eg. SLLI



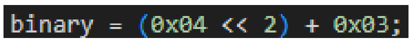
slli	Shift Left Logical Imm	I	0010011	0x1	imm[5:11]=0x00	rd = rs1 << imm[0:4]

My implementation:



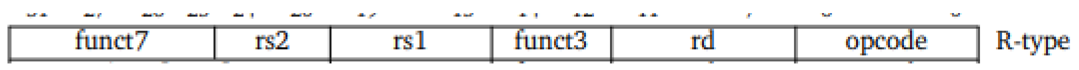
```
else if (is_opcode(opcode) == SLLI) {
  /* Lab2-1 assignment */
  binary = (1 << 12) + (0x04 << 2) + 0x03;
  binary += (reg_to_num(arg1, line_no) << 7);
  binary += (reg_to_num(arg2, line_no) << 15);
  binary += (lower5bit(arg3, line_no) << 20);
  warn("Lab2-1 assignment: SLLI instruction\n");
```

As shown in the image, I first add funct3 (0x1), and opcode (0x04 <<2) + 0x03 to the binary stream. Then rd to 7th bit, rs1 to 15th bit. For rs2, only lower 5 bit are used for binary coding since the position that should be shifted must be within 5 bits. (32 Position, this is because a RISC-V instruction should only contain 32-bits).

The implementation through all other I-format instructions is similar on rd, rs1, rs2, only differs in adding what funct3, For example, for ADDI, funct3 = 0x0, thus funct3 + opcode =  `binary = (0x04 << 2) + 0x03;`

Integer register-register instructions

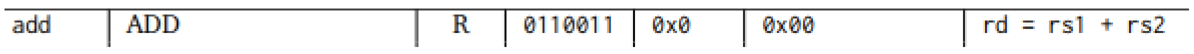
This includes the implementation of sub, sll, xor, srl, sra, or, and instructions, which they are all in R-format.



funct7	rs2	rs1	funct3	rd	opcode	R-type

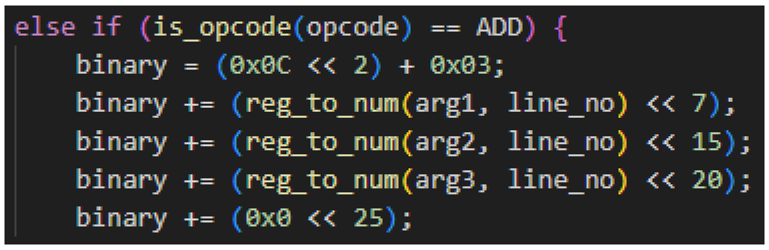
So, implementing these instructions will follow the structure of R-format to build.

Eg. ADD



add	ADD	R	0110011	0x0	0x00	rd = rs1 + rs2

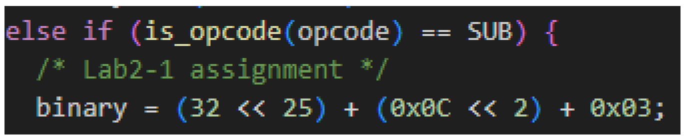
My implementation:



```
else if (is_opcode(opcode) == ADD) {
    binary = (0x0C << 2) + 0x03;
    binary += (reg_to_num(arg1, line_no) << 7);
    binary += (reg_to_num(arg2, line_no) << 15);
    binary += (reg_to_num(arg3, line_no) << 20);
    binary += (0x0 << 25);
```

Similar to I-format instruction, I first setup the ADD opcode (which is 0110011), then put rd, rs1, rs2 to the relative position in the bitstream. At last, set the funct7 value (0x00 for add instruction)

As the structure similar between same format, I implemented other R-format instruction in a similar way except changing the funct3 and funct7 value.



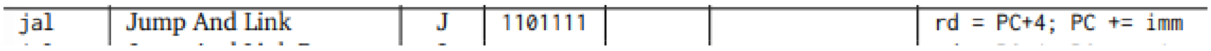
```
else if (is_opcode(opcode) == SUB) {
  /* Lab2-1 assignment */
  binary = (32 << 25) + (0x0C << 2) + 0x03;
```

For example, for SUB, funct7 = 0x20, So I need to also update the 29th bit to 1.

Unconditional Jumps

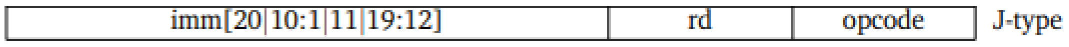
Unconditional jumps include JAL and JALR instructions, they are J and I format respectively.

Eg. JAL



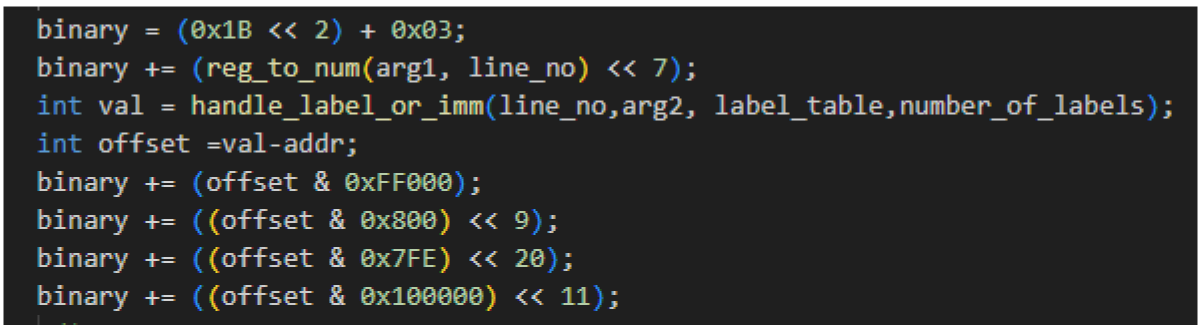
jal	Jump And Link	J	1101111			rd = PC+4; PC += imm

To implement JAL, J-format is the format to be followed.



imm[20|10:1|11|19:12]	rd	opcode	J-type

My implementation:



```
binary = (0x1B << 2) + 0x03;
binary += (reg_to_num(arg1, line_no) << 7);
int val = handle_label_or_imm(line_no,arg2, label_table,number_of_labels);
int offset =val-addr;
binary += (offset & 0xFF000);
binary += ((offset & 0x800) << 9);
binary += ((offset & 0x7FE) << 20);
binary += ((offset & 0x100000) << 11);
```

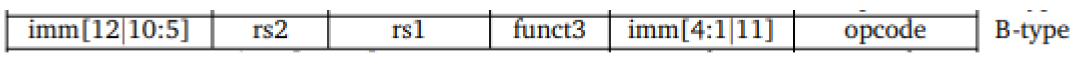
Firstly, set the opcode to (110111), then save rd value. Afterwards, handle_label_or_imm handle the jump target specified by arg2, if arg2 is a label, the function look up the table and return the corresponding address. The offset is used to for calculating the offset between jump target address and the current address. Which then store the address value to the immediate field.

Eg. JALR

The implementation is similar to the I instruction above. The only changes is that rd stores the return address, and also rs1 field is used to calculate the target address for the jump.

Conditional Branches

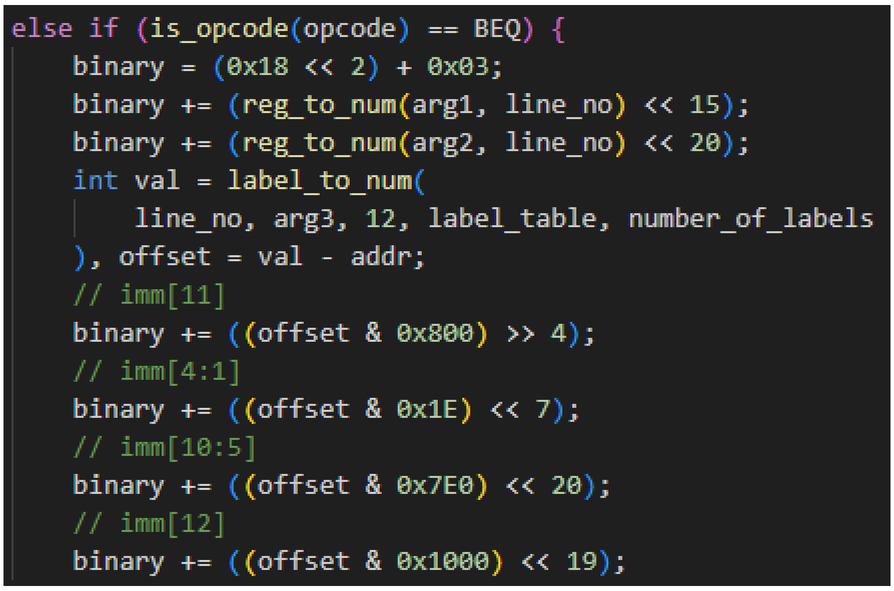
For conditional branches, bne, blt, bge is required for the implementation. Which all are in B-format instructions.



imm[12|10:5]	rs2	rs1	funct3	imm[4:1|11]	opcode	B-type

The implementation of branches is basically a mix of I-format and J-format instructions.

Eg. BEQ



```
else if (is_opcode(opcode) == BEQ) {
    binary = (0x18 << 2) + 0x03;
    binary += (reg_to_num(arg1, line_no) << 15);
    binary += (reg_to_num(arg2, line_no) << 20);
    int val = label_to_num(
        line_no, arg3, 12, label_table, number_of_labels
    ), offset = val - addr;
    // imm[11]
    binary += ((offset & 0x800) >> 4);
    // imm[4:1]
    binary += ((offset & 0x1E) << 7);
    // imm[10:5]
    binary += ((offset & 0x7E0) << 20);
    // imm[12]
    binary += ((offset & 0x1000) << 19);
```

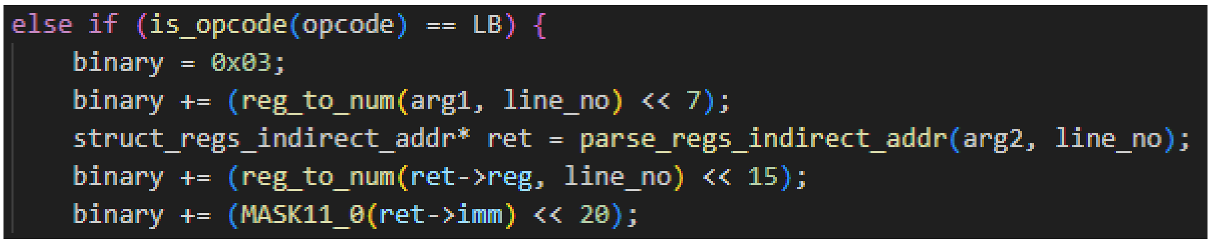
Similar to other instructions, the method to store rs2, rs1 and opcode are same. While to get the offset, similar to J-format instructions (JAL), label_to_num function is called to convert the label to an address value. The method of storing the offset is similar to J-format, only the number of bit and position of the bit are slightly different.

Load and Store instructions

This includes lb, lh, lw, sb, sh, sw that need to be implemented. lb, lh, lw are I format, sb, sh, sw are in S format.

Eg. LB (I-format)

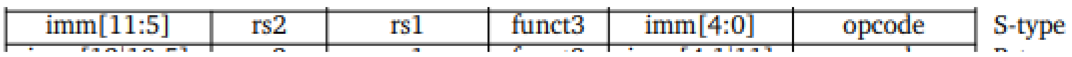
My implementation



```
else if (is_opcode(opcode) == LB) {
    binary = 0x03;
    binary += (reg_to_num(arg1, line_no) << 7);
    struct_regs_indirect_addr* ret = parse_regs_indirect_addr(arg2, line_no);
    binary += (reg_to_num(ret->reg, line_no) << 15);
    binary += (MASK11_0(ret->imm) << 20);
```

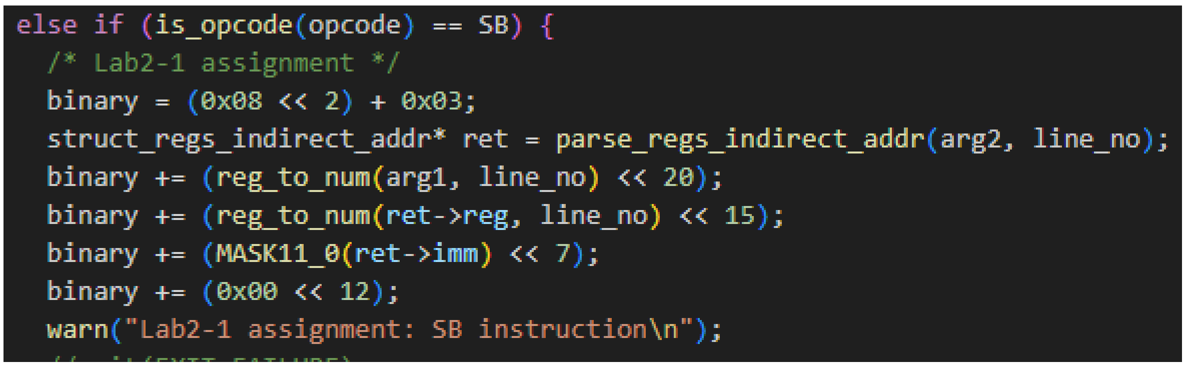
The method of implementing Load instructions is similar to other I format instruction like ADDI, SUBI. Only that the usage of the register is different. For example in LB instruction, reg2 (rs1) is used to specifies the base register and immediate offset for memory address location, the value is retrieved from parse_regs_indirect_addr which returns a structure containing the base register and the offset value.

Eg. SB (S-format)



imm[11:5]	rs2	rs1	funct3	imm[4:0]	opcode	S-type

My implementation:



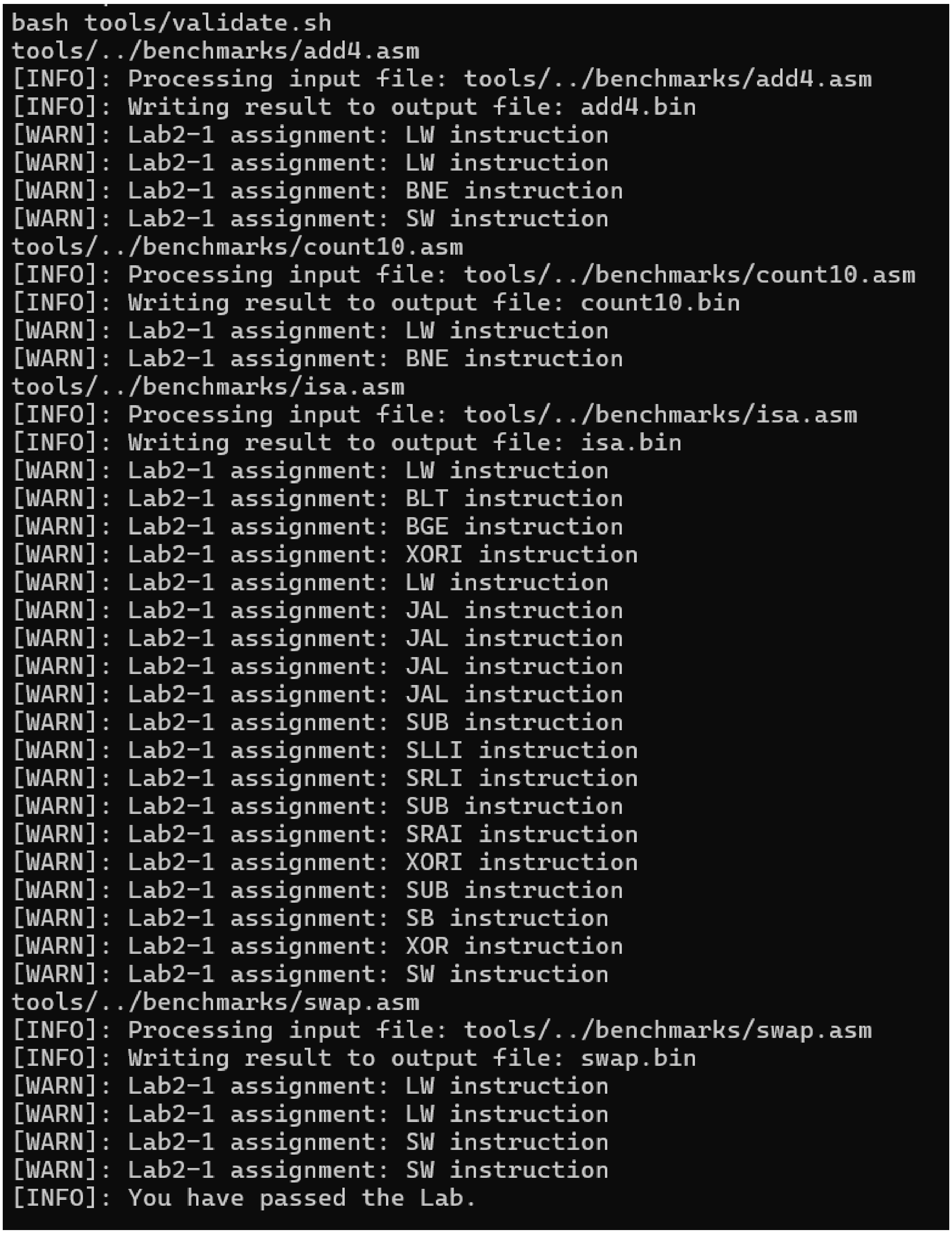
```
else if (is_opcode(opcode) == SB) {
  /* Lab2-1 assignment */
  binary = (0x08 << 2) + 0x03;
  struct_regs_indirect_addr* ret = parse_regs_indirect_addr(arg2, line_no);
  binary += (reg_to_num(arg1, line_no) << 20);
  binary += (reg_to_num(ret->reg, line_no) << 15);
  binary += (MASK11_0(ret->imm) << 7);
  binary += (0x00 << 12);
  warn("Lab2-1 assignment: SB instruction\n");
```

For SB instruction, first receiving the base value and offset value from ret, by parsing arg2.   arg1 is set to be the stored register. The base value and offset value that retrieved from the ret would be saved to the immediate area of S-format. MASK11_0 is used to store the immediate value to prevent exceeding 12-bits.

Executing asm.c

I have used ubuntu for the testing my code.

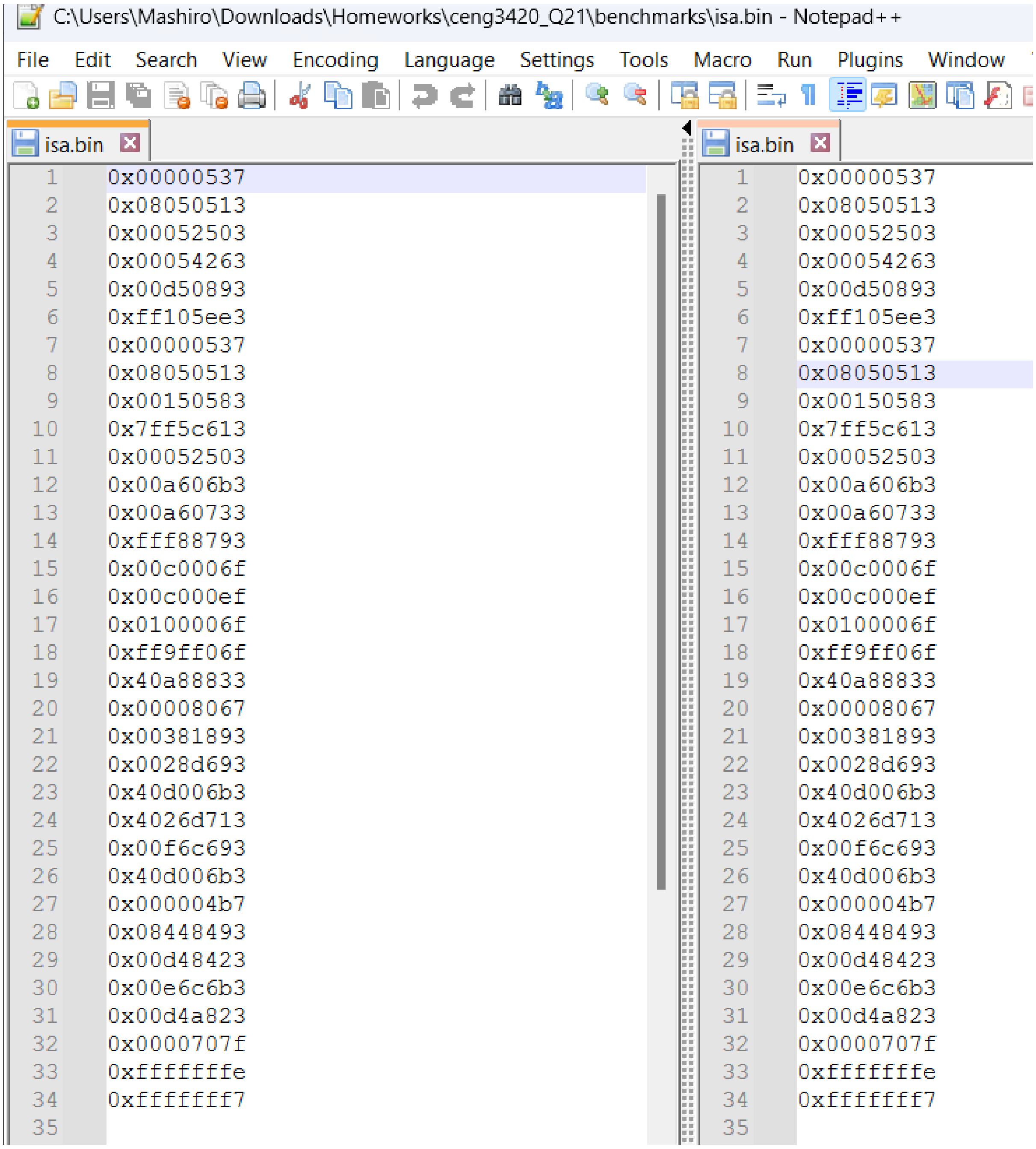
Command: make validate



```
bash tools/validate.sh
tools/../benchmarks/add4.asm
[INFO]: Processing input file: tools/../benchmarks/add4.asm
[INFO]: Writing result to output file: add4.bin
[WARN]: Lab2-1 assignment: LW instruction
[WARN]: Lab2-1 assignment: LW instruction
[WARN]: Lab2-1 assignment: BNE instruction
[WARN]: Lab2-1 assignment: SW instruction
tools/../benchmarks/count10.asm
[INFO]: Processing input file: tools/../benchmarks/count10.asm
[INFO]: Writing result to output file: count10.bin
[WARN]: Lab2-1 assignment: LW instruction
[WARN]: Lab2-1 assignment: BNE instruction
tools/../benchmarks/isa.asm
[INFO]: Processing input file: tools/../benchmarks/isa.asm
[INFO]: Writing result to output file: isa.bin
[WARN]: Lab2-1 assignment: LW instruction
[WARN]: Lab2-1 assignment: BLT instruction
[WARN]: Lab2-1 assignment: BGE instruction
[WARN]: Lab2-1 assignment: XORI instruction
[WARN]: Lab2-1 assignment: LW instruction
[WARN]: Lab2-1 assignment: JAL instruction
[WARN]: Lab2-1 assignment: JAL instruction
[WARN]: Lab2-1 assignment: JAL instruction
[WARN]: Lab2-1 assignment: JAL instruction
[WARN]: Lab2-1 assignment: SUB instruction
[WARN]: Lab2-1 assignment: SLLI instruction
[WARN]: Lab2-1 assignment: SRLI instruction
[WARN]: Lab2-1 assignment: SUB instruction
[WARN]: Lab2-1 assignment: SRAI instruction
[WARN]: Lab2-1 assignment: XORI instruction
[WARN]: Lab2-1 assignment: SUB instruction
[WARN]: Lab2-1 assignment: SB instruction
[WARN]: Lab2-1 assignment: XOR instruction
[WARN]: Lab2-1 assignment: SW instruction
tools/../benchmarks/swap.asm
[INFO]: Processing input file: tools/../benchmarks/swap.asm
[INFO]: Writing result to output file: swap.bin
[WARN]: Lab2-1 assignment: LW instruction
[WARN]: Lab2-1 assignment: LW instruction
[WARN]: Lab2-1 assignment: SW instruction
[WARN]: Lab2-1 assignment: SW instruction
[INFO]: You have passed the Lab.
```

Back-to-back comparison of isa.bin

Left: benchmarks Right: generated



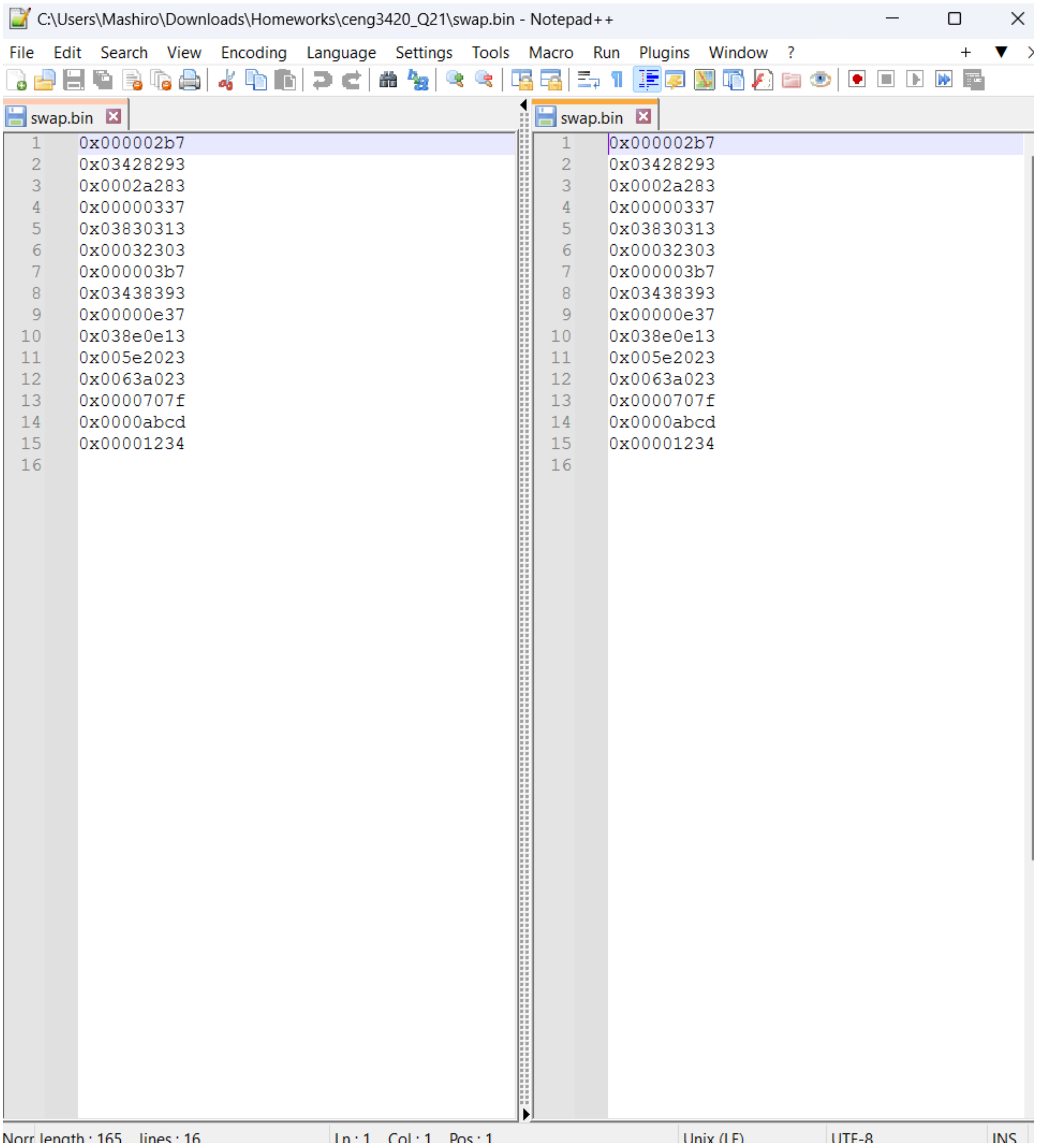
C:\Users\Mashiro\Downloads\Homeworks\ceng3420_Q21\benchmarks\isa.bin - Notepad++

```
0x00000537
0x08050513
0x00052503
0x00054263
0x00d50893
0xff105ee3
0x00000537
0x08050513
0x00150583
0x7ff5c613
0x00052503
0x00a606b3
0x00a60733
0xfff88793
0x00c0006f
0x00c000ef
0x0100006f
0xff9ff06f
0x40a88833
0x00008067
0x00381893
0x0028d693
0x40d006b3
0x4026d713
0x00f6c693
0x40d006b3
0x000004b7
0x08448493
0x00d48423
0x00e6c6b3
0x00d4a823
0x0000707f
0xfffffffe
0xfffffff7
```

```
0x00000537
0x08050513
0x00052503
0x00054263
0x00d50893
0xff105ee3
0x00000537
0x08050513
0x00150583
0x7ff5c613
0x00052503
0x00a606b3
0x00a60733
0xfff88793
0x00c0006f
0x00c000ef
0x0100006f
0xff9ff06f
0x40a88833
0x00008067
0x00381893
0x0028d693
0x40d006b3
0x4026d713
0x00f6c693
0x40d006b3
0x000004b7
0x08448493
0x00d48423
0x00e6c6b3
0x00d4a823
0x0000707f
0xfffffffe
0xfffffff7
```

Back-to-back comparison of swap.bin

Left: benchmarks Right: generated


C:\Users\Mashiro\Downloads\Homeworks\ceng3420_Q21\swap.bin - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
swap.bin
swap.bin
Norr length : 165 lines : 16
Ln : 1 Col : 1 Pos : 1
Unix (LF)
UTF-8
INS

Left:

```
0x000002b7
0x03428293
0x0002a283
0x00000337
0x03830313
0x00032303
0x000003b7
0x03438393
0x00000e37
0x038e0e13
0x005e2023
0x0063a023
0x0000707f
0x0000abcd
0x00001234

```

Right:

```
0x000002b7
0x03428293
0x0002a283
0x00000337
0x03830313
0x00032303
0x000003b7
0x03438393
0x00000e37
0x038e0e13
0x005e2023
0x0063a023
0x0000707f
0x0000abcd
0x00001234

```