CENG 3420 Lab 2-2 Report

Name: Chan Chun Ming SID: 1155163257

Lab 2.2

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

To complete the Lab. Firstly, need to add the code to handle with 24 instructions by getting the current instruction, and get the registers in the bit to perform different operations like arithmetic, branches, or immediate operations.

For example, of handling **ADDI** instructions:

```
void handle_addi(unsigned int cur_inst) {
    unsigned int rd = MASK11_7(cur_inst), rs1 = MASK19_15(cur_inst);
    int imm12 = sext(MASK31_20(cur_inst), 12);
    NEXT_LATCHES.REGS[rd] = CURRENT_LATCHES.REGS[rs1] + imm12;
    warn("Lab2-2 assignment: ADDI\n");
}
```

The function extracts the destination register (rd) numbers from 11-7th bit, source register from 19-15th bit. The value of this register will be added with the 12 bits immediate value, thus performing the addition immediate action.

Simply said, this part basically is referencing the format of each instruction, to determine the segmentation of the bit stream from the current address.

Give one more example of handling **JAL** instructions:

```
void handle_jal(unsigned int cur_inst) {
    /*
     * Lab2-2 assignment
     */
    unsigned int rd = MASK11_7(cur_inst);
    int imm20= (MASK31(cur_inst) << 20) + \
        (MASK19_12(cur_inst) << 12) + \
        (MASK20(cur_inst) << 11) + \
        (MASK30_21(cur_inst) << 1);
    NEXT_LATCHES.PC = sext(imm20, 20)+CURRENT_LATCHES.PC;
    NEXT_LATCHES.REGS[rd] = CURRENT_LATCHES.PC +4;
    warn("Lab2-2 assignment: JAL\n");
    //exit(EXIT_FAILURE);
}
```

To handle JAL instructions, firstly extract the rd value from the bitstream, which would be storing the next instruction address. Then calculating the jump target address by combining back the immediate bit fields that is encoded across multiple but fields.

Finally, update the program counter and register that will be used to jump to the target in the next cycle and save the next current instruction address for return use.

The second part of the code assignment is to handle the instruction by decoding the instruction. That means I need to determine the instruction by checking the value of opcode, funct3, funct7 ...

Example of filtering arithmetic-arithmetic instructions:

```
case (0x0C << 2) + 0x03:
    /*
     * Integer Register-Register Instructions
     */
    switch(funct3) {
        case 0:
            if (MASK31_25(cur_inst) == 0)
                handle_add(cur_inst);
            else
                handle_sub(cur_inst);
            break;
        case 1:
            handle_sll(cur_inst);
            break;
        case 4:
            handle_xor(cur_inst);
            break;
        case 5:
            if (MASK31_25(cur_inst) == 0)
                handle_srl(cur_inst);
            else
                handle_sra(cur_inst);
            break;
        case 6:
            handle_or(cur_inst);
            break;
        case 7:
            handle_and(cur_inst);
            break;
        default:
            error("unknown opcode 0x%08x is captured.\n", cur_inst);
```

For sub, sll, xor, srl, sra, or, and instructions, they all have the same opcode (0110011 or (0x0X <<2) + 0x03, thus I established a case for the instructions with same opcode. Then depends on their funct3 difference (In addition using funct7 to diff ADD and SUB instruction) to determine which function is called to handle with the instruction.

This practice is same for other instructions, such as load instructions. Starting from opcode difference to build a case, then using funct3 or funct7 to distinguish the specific instructions within the same category.

Example of load instructions

Determine the specific instructions by opcode and funct3 value

```
case (0x00 << 2) + 0x03:
    switch(funct3) {
        case 0:
            handle_lb(cur_inst);
            break;
        case 1:
            handle_lh(cur_inst);
            break;
        case 2:
            handle_lw(cur_inst);
            break;
        default:
            error("unknown opcode 0x%08x is captured.\n", cur_inst);
    }
```

Verification

isa.bin

```
2$ ./sim benchmarks/isa.bin
```

Result:

Command: go

```
RISCV LC SIM > go
```

...

```
[WARN]: Lab2-2 assignment: SW
[INFO]: cur_inst = 0x0000707f
[INFO]: RISCV LC is halted.
```

Command: rdump

```
RISCV LC SIM > rdump
```

```
a3      [x13]:  0xfffffffe
```

Command: mdump 0x84 0xAA

```
memory content [0x00000084..0x000000aa]:
-------------------------------------
  0x00000084 (132) : 0xfffffff7
  0x00000088 (136) : 0x00000000
  0x0000008c (140) : 0x00000017
  0x00000090 (144) : 0x00000000
  0x00000094 (148) : 0xfffffffe
```

The final value of a3 = 0xfffffffe (-18) and MEMORY[0x84+16] = 0xfffffffe

count10.bin

```
$ ./sim benchmarks/count10.bin
```

Result:

Command: go

```
RISCV LC SIM > go
```

...

```
[WARN]: Lab2-2 assignment: BNE
[INFO]: cur_inst = 0x0000707f
[INFO]: RISCV LC is halted.
```

Command: rdump

```
RISCV LC SIM > rdump
```

```
t2      [x7]:   0x00000037
```

The final value of t2 is 0x00000037 = 55

swap.bin

```
$ ./sim benchmarks/swap.bin
```

Result:

Command: run 6

Command: mdump 0x30 0x50 (Before)

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

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

Command: go

Command: mdump 0x34 0x38 (After)

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

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

The value of 0x00000034 and 0x00000038 is swapped compared to initial value and final value

add4.bin

```
$ ./sim benchmarks/add4.bin
```

Result:

Command: run 4

Command: mdump 0x38 0x3C

```
RISCV LC SIM > mdump 0x38 0x3C
```

```
memory content [0x00000038..0x0000003c]:
-------------------------------------
  0x00000038 (56) : 0xfffffffb
  0x0000003c (60) : 0x00000000
```

Command: go

Command: mdump 0x38 0x3C

```
RISCV LC SIM > mdump 0x38 0x3C
```

```
memory content [0x00000038..0x0000003c]:
-------------------------------------
  0x00000038 (56) : 0xffffffff
  0x0000003c (60) : 0x00000000
```

The value of BL (address = 0x00000038) changes from 0xfffffffb (-5) to 0xffffffff (-1)