CENG 3420 lab 2 report

Name: Xu Haoran

SID: 1155124383

Date: 2023/3/27

The implementation of lab 2-2:

```
if (is_opcode(opcode) == ADDI) {
        binary = (0x04 << 2) + 0x03;
        binary += (reg_to_num(arg1, line_no) << 7);
        binary += (reg_to_num(arg2, line_no) << 15);
        binary += (MASK11_0(validate_imm(arg3, 12, line_no)) << 20);
    } else if (is_opcode(opcode) == SLLI) {
        binary = (0x01 << 12) + (0x04 << 2) + 0x03;
        binary += (reg_to_num(arg1, line_no) << 7);
        binary += (reg_to_num(arg2, line_no) << 15);
        binary += ((validate_imm(arg3, 12, line_no) & 0x1F)  << 20);
    } else if (is_opcode(opcode) == XORI) {
        binary = (0x04 << 12) + (0x04 << 2) + 0x03;
        binary += (reg_to_num(arg1, line_no) << 7);
        binary += (reg_to_num(arg2, line_no) << 15);
        binary += (MASK11_0(validate_imm(arg3, 12, line_no)) << 20);
    } else if (is_opcode(opcode) == SRLI) {
        binary = (0x05 << 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);
    } else if (is_opcode(opcode) == SRAI) {
        binary = (0x10 << 26) + (0x05 << 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);
    } else if (is_opcode(opcode) == ORI) {
        binary = (0x06 << 12) + (0x04 << 2) + 0x03;
        binary += (reg_to_num(arg1, line_no) << 7);
        binary += (reg_to_num(arg2, line_no) << 15);
        binary += (MASK11_0(validate_imm(arg3, 12, line_no)) << 20);
    } else if (is_opcode(opcode) == ANDI) {
        binary = (0x07 << 12) + (0x04 << 2) + 0x03;
        binary += (reg_to_num(arg1, line_no) << 7);
```

```
        binary += (reg_to_num(arg2, line_no) << 15);
        binary += (MASK11_0(validate_imm(arg3, 12, line_no)) << 20);
    } else if (is_opcode(opcode) == LUI) {
        binary = (0x0D << 2) + 0x03;
        binary += (reg_to_num(arg1, line_no) << 7);
        binary += handle_label_or_imm(
                line_no,
                arg2,
                label_table,
                number_of_labels
            ) & 0xFFFFF000;
    }

    // Integer Register-Register Operations
    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);
    } else if (is_opcode(opcode) == SUB) {
        binary = (0x20 << 25) + (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);
    } else if (is_opcode(opcode) == SLL) {
        binary = (0x01 << 12) + (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);
    } else if (is_opcode(opcode) == XOR) {
        binary = (0x04 << 12) + (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);
    } else if (is_opcode(opcode) == SRL) {
        binary = (0x05 << 12) + (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);
    } else if (is_opcode(opcode) == SRA) {
        binary = (0x20 << 25) + (0x05 << 12) + (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);
    } else if (is_opcode(opcode) == OR) {
        binary = (0x06 << 12) + (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);
    } else if (is_opcode(opcode) == AND) {
        binary = (0x07 << 12) + (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);
    }


    // Unconditional Jumps
    else if (is_opcode(opcode) == JALR) {
        binary = (0x19 << 2) + 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 += (ret->imm << 20);
    } else if (is_opcode(opcode) == JAL) {
        binary = (0x1B << 2) + 0x03;
        binary += (reg_to_num(arg1, line_no) << 7);
        int offset;
        offset = handle_label_or_imm(arg2, label_table, cmd_no,
line_no);
        // imm[19:12]
        binary += (offset & 0xFF000);
        // imm[11]
        binary += ((offset & 0x800) << 9);
        // imm[10:1]
        binary += ((offset & 0x7FE) << 20);
        // imm[20]
        binary += ((offset & 0x100000) << 11);
    }

    // Conditional Branches
    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);
    } else if (is_opcode(opcode) == BNE) {
        binary = (0x01 << 12) + (0x18 << 2) + 0x03;
        binary += (reg_to_num(arg1, line_no) << 15);
        binary += (reg_to_num(arg2, line_no) << 20);
        int offset;
        offset = label_to_num(arg3, 12, label_table, cmd_no, line_no);
        // 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);
    } else if (is_opcode(opcode) == BLT) {
        binary = (0x04 << 12) + (0x18 << 2) + 0x03;
        binary += (reg_to_num(arg1, line_no) << 15);
        binary += (reg_to_num(arg2, line_no) << 20);
        int offset;
        offset = label_to_num(arg3, 12, label_table, cmd_no, line_no);
        // 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);
    } else if (is_opcode(opcode) == BGE) {
        binary = (0x05 << 12) + (0x18 << 2) + 0x03;
        binary += (reg_to_num(arg1, line_no) << 15);
        binary += (reg_to_num(arg2, line_no) << 20);
        int offset;
        offset = label_to_num(arg3, 12, label_table, cmd_no, line_no);
        // 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);
    }

    // Load and Store Instructions
    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);
    } else if (is_opcode(opcode) == LH) {
        binary = (0x01 << 12) + 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 += (ret->imm << 20);
    } else if (is_opcode(opcode) == LW) {
        binary = (0x02 << 12) + 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 += (ret->imm << 20);
    } else if (is_opcode(opcode) == SB) {
        binary = (0x08 << 2) + 0x03;
        binary += (reg_to_num(arg1, line_no) << 20);
        struct_regs_indirect_addr* ret = parse_regs_indirect_addr(arg2,
line_no);
        binary += (reg_to_num(ret->reg, line_no) << 15);
        // imm[4:0]
        binary += ((ret->imm & 0x1F) << 7);
        // imm[11:5]
        binary += ((ret->imm & 0xFE0) << 20);
    } else if (is_opcode(opcode) == SH) {
        binary = (0x01 << 12) + (0x08 << 2) + 0x03;
        binary += (reg_to_num(arg1, line_no) << 20);
```

```
        struct_regs_indirect_addr* ret = parse_regs_indirect_addr(arg2,
line_no);
        binary += (reg_to_num(ret->reg, line_no) << 15);
        // imm[4:0]
        binary += ((ret->imm & 0x1F) << 7);
        // imm[11:5]
        binary += ((ret->imm & 0xFE0) << 20);
    } else if (is_opcode(opcode) == SW) {
        binary = (0x02 << 12) + (0x08 << 2) + 0x03;
        binary += (reg_to_num(arg1, line_no) << 20);
        struct_regs_indirect_addr* ret = parse_regs_indirect_addr(arg2,
line_no);
        binary += (reg_to_num(ret->reg, line_no) << 15);
        // imm[4:0]
        binary += ((ret->imm & 0x1F) << 7);
        // imm[11:5]
        binary += ((ret->imm & 0xFE0) << 20);
    }
    return binary;
}
```

The output for lab 2-1:

```
bash tools/validate.sh
tools/../benchmarks/isa.asm
Processing input file tools/../benchmarks/isa.asm
Writing result to output file isa.bin
tools/../benchmarks/swap.asm
Processing input file tools/../benchmarks/swap.asm
Writing result to output file swap.bin
tools/../benchmarks/count10.asm
Processing input file tools/../benchmarks/count10.asm
Writing result to output file count10.bin
[INFO]: You have passed the Lab.
```

The implementation of lab 2-2:

```
void handle_slli(unsigned int cur_inst) {
    unsigned int rd = MASK11_7(cur_inst), rs1 = MASK19_15(cur_inst);
    int shamt = sext(MASK24_20(cur_inst), 5);
    NEXT_LATCHES.REGS[rd] = CURRENT_LATCHES.REGS[rs1] << shamt;
    warn("Lab2-2 assignment: SLLI\n");
}



void handle_xori(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: XORI\n");
}

void handle_srli(unsigned int cur_inst) {
    unsigned int rd = MASK11_7(cur_inst), rs1 = MASK19_15(cur_inst);
    int shamt = sext(MASK24_20(cur_inst), 5);
    unsigned int temp = CURRENT_LATCHES.REGS[rs1];
    NEXT_LATCHES.REGS[rd] = temp >> shamt;
    warn("Lab2-2 assignment: SRLI\n");
}

void handle_srai(unsigned int cur_inst) {
    unsigned int rd = MASK11_7(cur_inst), rs1 = MASK19_15(cur_inst);
    int shamt = sext(MASK24_20(cur_inst), 5);
    int temp = CURRENT_LATCHES.REGS[rs1];
    NEXT_LATCHES.REGS[rd] = temp >> shamt;
    warn("Lab2-2 assignment: SRAI\n");
}

void handle_ori(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: ORI\n");
}

void handle_andi(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: ANDI\n");
}

void handle_lui(unsigned int cur_inst) {
    unsigned int rd = MASK11_7(cur_inst);
    int imm20 = MASK31_12(cur_inst);
    NEXT_LATCHES.REGS[rd] = (imm20 << 12);
```

```
    warn("Lab2-2 assignment: LUI\n");
}


void handle_add(unsigned int cur_inst) {
    unsigned int rd = MASK11_7(cur_inst),
        rs1 = MASK19_15(cur_inst),
        rs2 = MASK24_20(cur_inst);
    NEXT_LATCHES.REGS[rd] = CURRENT_LATCHES.REGS[rs1] +
CURRENT_LATCHES.REGS[rs2];
}


void handle_sub(unsigned int cur_inst) {
    unsigned int rd = MASK11_7(cur_inst),
    rs1 = MASK19_15(cur_inst),
    rs2 = MASK24_20(cur_inst);
    NEXT_LATCHES.REGS[rd] = CURRENT_LATCHES.REGS[rs1] -
CURRENT_LATCHES.REGS[rs2];
    warn("Lab2-2 assignment: SUB\n");
}


void handle_sll(unsigned int cur_inst) {
    unsigned int rd = MASK11_7(cur_inst),
    rs1 = MASK19_15(cur_inst),
    rs2 = MASK24_20(cur_inst);
    NEXT_LATCHES.REGS[rd] = CURRENT_LATCHES.REGS[rs1] <<
CURRENT_LATCHES.REGS[rs2];
    warn("Lab2-2 assignment: SLL\n");
}


void handle_xor(unsigned int cur_inst) {
    unsigned int rd = MASK11_7(cur_inst),
    rs1 = MASK19_15(cur_inst),
    rs2 = MASK24_20(cur_inst);
    NEXT_LATCHES.REGS[rd] = CURRENT_LATCHES.REGS[rs1] ^
CURRENT_LATCHES.REGS[rs2];
    warn("Lab2-2 assignment: XOR\n");
}


void handle_srl(unsigned int cur_inst) {
    unsigned int rd = MASK11_7(cur_inst),
```

```
    rs1 = MASK19_15(cur_inst),
    rs2 = MASK24_20(cur_inst);
    unsigned int temp = CURRENT_LATCHES.REGS[rs1];
    NEXT_LATCHES.REGS[rd] = temp >> CURRENT_LATCHES.REGS[rs2];
    warn("Lab2-2 assignment: SRL\n");
}

void handle_sra(unsigned int cur_inst) {
    unsigned int rd = MASK11_7(cur_inst),
    rs1 = MASK19_15(cur_inst),
    rs2 = MASK24_20(cur_inst);
    int temp = CURRENT_LATCHES.REGS[rs1];
    NEXT_LATCHES.REGS[rd] = temp >> CURRENT_LATCHES.REGS[rs2];
    warn("Lab2-2 assignment: SRA\n");
}

void handle_or(unsigned int cur_inst) {
    unsigned int rd = MASK11_7(cur_inst),
    rs1 = MASK19_15(cur_inst),
    rs2 = MASK24_20(cur_inst);
    NEXT_LATCHES.REGS[rd] = CURRENT_LATCHES.REGS[rs1] |
CURRENT_LATCHES.REGS[rs2];
    warn("Lab2-2 assignment: OR\n");
}

void handle_and(unsigned int cur_inst) {
    unsigned int rd = MASK11_7(cur_inst),
    rs1 = MASK19_15(cur_inst),
    rs2 = MASK24_20(cur_inst);
    NEXT_LATCHES.REGS[rd] = CURRENT_LATCHES.REGS[rs1] &
CURRENT_LATCHES.REGS[rs2];
    warn("Lab2-2 assignment: AND\n");
}

void handle_jalr(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.PC = CURRENT_LATCHES.REGS[rs1] + imm12;
    NEXT_LATCHES.REGS[rd] = CURRENT_LATCHES.PC + 4;
    warn("Lab2-2 assignment: JALR\n");
```

```
}


void handle_jal(unsigned int cur_inst) {
    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);
    NEXT_LATCHES.REGS[rd] = CURRENT_LATCHES.PC + 4;
    warn("Lab2-2 assignment: JAL\n");
}
void handle_bne(unsigned int cur_inst) {
    unsigned int rs1 = MASK19_15(cur_inst), rs2 = MASK24_20(cur_inst);
    int imm12 = (MASK31(cur_inst) << 12) + \
            (MASK7(cur_inst) << 11) + \
            (MASK30_25(cur_inst) << 5) + \
            (MASK11_8(cur_inst) << 1);
    if (CURRENT_LATCHES.REGS[rs1] != CURRENT_LATCHES.REGS[rs2])
        NEXT_LATCHES.PC = sext(imm12, 12);
    warn("Lab2-2 assignment: BNE\n");
}


void handle_blt(unsigned int cur_inst) {
    unsigned int rs1 = MASK19_15(cur_inst), rs2 = MASK24_20(cur_inst);
    int imm12 = (MASK31(cur_inst) << 12) + \
            (MASK7(cur_inst) << 11) + \
            (MASK30_25(cur_inst) << 5) + \
            (MASK11_8(cur_inst) << 1);
    if (CURRENT_LATCHES.REGS[rs1] < CURRENT_LATCHES.REGS[rs2])
        NEXT_LATCHES.PC = sext(imm12, 12);
    warn("Lab2-2 assignment: BLT\n");
}


void handle_bge(unsigned int cur_inst) {
    unsigned int rs1 = MASK19_15(cur_inst), rs2 = MASK24_20(cur_inst);
    int imm12 = (MASK31(cur_inst) << 12) + \
            (MASK7(cur_inst) << 11) + \
            (MASK30_25(cur_inst) << 5) + \
            (MASK11_8(cur_inst) << 1);
    if (CURRENT_LATCHES.REGS[rs1] >= CURRENT_LATCHES.REGS[rs2])
        NEXT_LATCHES.PC = sext(imm12, 12);
```

```
    warn("Lab2-2 assignment: BGE\n");
}


void handle_lb(unsigned int cur_inst) {
    unsigned int rd = MASK11_7(cur_inst), rs1 = MASK19_15(cur_inst);
    int imm12 = MASK31_20(cur_inst);
    NEXT_LATCHES.REGS[rd] = sext(MASK7_0(MEMORY[sext(imm12, 12) +
CURRENT_LATCHES.REGS[rs1]]), 8);
}


void handle_lh(unsigned int cur_inst) {
     unsigned int rd = MASK11_7(cur_inst), rs1 = MASK19_15(cur_inst);
    int imm12 = MASK31_20(cur_inst);
    NEXT_LATCHES.REGS[rd] = (MASK7_0(MEMORY[sext(imm12, 12) +
CURRENT_LATCHES.REGS[rs1]])) + \
                            (sext(MASK7_0(MEMORY[sext(imm12, 12) +
CURRENT_LATCHES.REGS[rs1] + 1]), 8) << 8);

    warn("Lab2-2 assignment: LH\n");
}

void handle_lw(unsigned int cur_inst) {
    unsigned int rd = MASK11_7(cur_inst), rs1 = MASK19_15(cur_inst);
    int imm12 = MASK31_20(cur_inst);
    NEXT_LATCHES.REGS[rd] = (MASK7_0(MEMORY[sext(imm12, 12) +
CURRENT_LATCHES.REGS[rs1]])) + \
                            (MASK7_0(MEMORY[sext(imm12, 12) +
CURRENT_LATCHES.REGS[rs1] + 1]) << 8) + \
                            (MASK7_0(MEMORY[sext(imm12, 12) +
CURRENT_LATCHES.REGS[rs1] + 2]) << 16) + \
                            (MASK7_0(MEMORY[sext(imm12, 12) +
CURRENT_LATCHES.REGS[rs1] + 3]) << 24);
    printf("Loaded 0x%08x from 0x%08x\n", NEXT_LATCHES.REGS[rd],
sext(imm12, 12) + CURRENT_LATCHES.REGS[rs1]);
    warn("Lab2-2 assignment: LW\n");
}


void handle_sb(unsigned int cur_inst) {
    unsigned int rs1 = MASK19_15(cur_inst), rs2 = MASK24_20(cur_inst);
    int imm12 = MASK31_25(cur_inst) << 5 + \
            MASK11_7(cur_inst);
```

```
    MEMORY[sext(imm12, 12) + CURRENT_LATCHES.REGS[rs1]] =
MASK7_0(CURRENT_LATCHES.REGS[rs2]);
    warn("Lab2-2 assignment: SB\n");
}


void handle_sh(unsigned int cur_inst) {
    unsigned int rs1 = MASK19_15(cur_inst), rs2 = MASK24_20(cur_inst);
    int imm12 = MASK31_25(cur_inst) << 5 + \
          MASK11_7(cur_inst);
    MEMORY[sext(imm12, 12) + CURRENT_LATCHES.REGS[rs1]] =
MASK7_0(CURRENT_LATCHES.REGS[rs2]);
    MEMORY[sext(imm12, 12) + CURRENT_LATCHES.REGS[rs1] + 1] =
MASK15_8(CURRENT_LATCHES.REGS[rs2]);
    warn("Lab2-2 assignment: SH\n");
}


void handle_sw(unsigned int cur_inst) {
    unsigned int rs1 = MASK19_15(cur_inst), rs2 = MASK24_20(cur_inst);
    int imm12 = MASK31_25(cur_inst) << 5 + \
           MASK11_7(cur_inst);
    MEMORY[sext(imm12, 12) + CURRENT_LATCHES.REGS[rs1]] =
MASK7_0(CURRENT_LATCHES.REGS[rs2]);
    MEMORY[sext(imm12, 12) + CURRENT_LATCHES.REGS[rs1] + 1] =
MASK15_8(CURRENT_LATCHES.REGS[rs2]);
    MEMORY[sext(imm12, 12) + CURRENT_LATCHES.REGS[rs1] + 2] =
MASK23_16(CURRENT_LATCHES.REGS[rs2]);
    MEMORY[sext(imm12, 12) + CURRENT_LATCHES.REGS[rs1] + 3] =
MASK31_24(CURRENT_LATCHES.REGS[rs2]);
    printf("Save 0x%08x to 0x%08x\n", CURRENT_LATCHES.REGS[rs2],
sext(imm12, 12) + CURRENT_LATCHES.REGS[rs1]);
    warn("Lab2-2 assignment: SW\n");
}
```

The output for lab 2-2:

```
bash tools/validate.sh
tools/../benchmarks/isa.asm
Processing input file tools/../benchmarks/isa.asm
Writing result to output file isa.bin
tools/../benchmarks/swap.asm
Processing input file tools/../benchmarks/swap.asm
Writing result to output file swap.bin
tools/../benchmarks/count10.asm
Processing input file tools/../benchmarks/count10.asm
Writing result to output file count10.bin
[INFO]: You have passed the Lab.
```