6.004 Tutorial Problems
L09 – Programmable Machines

**Computational Instructions**

R-type: Register-register instructions: opcode = OP = 0110011

<table>
<thead>
<tr>
<th>Arithmetic</th>
<th>Comparisons</th>
<th>Logical</th>
<th>Shifts</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD, SUB</td>
<td>SLT, SLTU</td>
<td>AND, OR, XOR</td>
<td>SLL, SRL, SRA</td>
</tr>
</tbody>
</table>

Assembly instr: oper rd, rs1, rs2
Behavior: reg[rd] <= reg[rs1] oper reg[rs2]

SLT – Set less than
SLTU – Set less than unsigned
SLL – Shift left logical
SRL – Shift right logical
SRA – Shift right arithmetic

I-type: Register-immediate instructions: with opcode = OP-IMM = 0010011

<table>
<thead>
<tr>
<th>Arithmetic</th>
<th>Comparisons</th>
<th>Logical</th>
<th>Shifts</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADDI</td>
<td>SLTI, SLTIU</td>
<td>ANDI, ORI, XOR</td>
<td>SLLI, SRLI, SRAI</td>
</tr>
</tbody>
</table>

Assembly instr: oper rd, rs1, immI
Behavior: imm = signExtend(immI)
reg[rd] <= reg[rs1] oper imm

Same functions as R-type except SUBI is not needed.
Function is encoded in funct3 bits plus instr[30]. Instr[30] = 1 for SRAI. So SRLI and SRAI use same funct3 encoding.
immI is a 12 bit constant.

U-type: opcode = LUI or AUIPC = (01|00)10111

LUI – load upper immediate
AUIPC – add upper immediate to PC
**Assembly instr:**

\[ \text{lui } rd, \text{immU} \]

**Behavior:**

\[ \text{imm} = \{\text{immU}, 12'\text{b}0\} \quad \text{// No extension needed: 32bit} \]
\[ \text{Reg}[rd] \Leftarrow \text{imm} \]

For example \text{lui } x2, 2 would load register x2 with 0x2000. \text{immU} is a 20 bit constant.

**Load Store Instructions**

**I-type: Load: with opcode = LOAD = 0000011**

**LW** – load word

**Assembly instr:**

\[ \text{lw } rd, \text{immI}(rs1) \]

**Behavior:**

\[ \text{imm} = \text{signExtend}(\text{immI}) \]
\[ \text{Reg}[rd] \Leftarrow \text{Mem}[\text{R}[rs1] + \text{imm}] \]

**S-type: Store: opcode = STORE = 0100011**

**SW** – store word

**Assembly instr:**

\[ \text{sw } rs2, \text{immS}(rs1) \]

**Behavior:**

\[ \text{imm} = \text{signExtend}(\text{immS}) \]
\[ \text{Mem}[\text{R}[rs1] + \text{imm}] \Leftarrow \text{R}[rs2] \]

\text{immS} is a 12 bit constant.

**Control Instructions**

**SB-type: Conditional Branches: opcode = 1100011**

**Assembly instr:**

\[ \text{oper } rs1, rs2, \text{label} \]

**Behavior:**

\[ \text{imm} = \text{distance to label in bytes} \]
\[ = \text{signExtend}\{\text{immB}[12:1], 0\} \]
\[ \text{pc} \Leftarrow (\text{R}[rs1] \text{ comp R}[rs2]) \ ? \text{pc} + \text{imm} : \text{pc} + 4 \]

Compares register rs1 to rs2. If comparison is true then pc is updated with pc + imm, otherwise pc becomes pc + 4. Comparison type is defined by operation.

BEQ – branch if equal (==)
BNE – branch if not equal (!=)
BLT – branch if less than (<)
BGE – branch if greater than or equal (>=)
BLTU – branch if less than using unsigned numbers (< unsigned)
BGEU – branch if greater than or equal using unsigned numbers (>= unsigned)
UJ-type: Unconditional Jump: opcode = JAL = 110111

Assembly instr: JAL rd, label

Behavior:
imm = distance to label in bytes
\[ \text{signExtend}(\text{imm}[20:1],0) \]
R[rd] <= pc + 4; pc <= pc + imm

I-type: Unconditional Jump: opcode = JALR = 1100111

Assembly instr: JALR rd, rs1, immI

Behavior:
imm = signExtend(immI)
R[rd] <= pc + 4; pc <= (R[rs1]+imm) & ~0x01
(zero out the bottom bit of pc)

JAL – jump and link
JALR – jump and link register

Common pseudoinstructions:

j label = jal x0, label (ignore return address)

li x1, 0x1000 = lui x1, 1
li x1, 0x1100 = lui x1, 1; addi x1, x1, 0x100
li x4, 3 = addi x4, x0, 3

mv x3, x2 = addi x3, x2, 0

beqz x1, target = beq x1, x0, target
bneqz x1, target = bneq x1, x0, target
Figure 2.2: RISC-V base instruction formats.

Figure 2.3: RISC-V base instruction formats showing immediate variants.

Figure 2.4: Types of immediate produced by RISC-V instructions. The fields are labeled with the instruction bits used to construct their value. Sign extension always uses $\text{inst}[31]$.

Table 9.1: RISC-V base opcode map, $\text{inst}[1:0]=11$
### RV32I Base Instruction Set (MIT 6.S084 subset)

<table>
<thead>
<tr>
<th>imm[31:12]</th>
<th>rd</th>
<th>0110111</th>
<th>LUI</th>
</tr>
</thead>
<tbody>
<tr>
<td>imm[12:10:5]</td>
<td>rs1</td>
<td>000</td>
<td>rd 1100111</td>
</tr>
<tr>
<td>imm[11:0]</td>
<td>rs1</td>
<td>010</td>
<td>rd 0000011</td>
</tr>
<tr>
<td>imm[11:0]</td>
<td>rs1</td>
<td>010</td>
<td>rd 0010011</td>
</tr>
<tr>
<td>imm[11:0]</td>
<td>rs1</td>
<td>011</td>
<td>rd 0010011</td>
</tr>
<tr>
<td>imm[11:0]</td>
<td>rs1</td>
<td>110</td>
<td>rd 0010011</td>
</tr>
<tr>
<td>imm[11:0]</td>
<td>rs1</td>
<td>111</td>
<td>rd 0010011</td>
</tr>
<tr>
<td>imm[11:0]</td>
<td>rs1</td>
<td>111</td>
<td>rd 0010011</td>
</tr>
<tr>
<td>imm[11:0]</td>
<td>rs1</td>
<td>111</td>
<td>rd 0010011</td>
</tr>
</tbody>
</table>

| shammt | rs2 | rd 010011 | ADD           |
| shammt | rs2 | rd 010011 | SUB           |
| shammt | rs2 | rd 010011 | SLL           |
| shammt | rs2 | rd 010011 | STL           |
| shammt | rs2 | rd 010011 | SLTU          |
| shammt | rs2 | rd 010011 | XOR           |
| shammt | rs2 | rd 010011 | SRL           |
| shammt | rs2 | rd 010011 | SRA           |
| shammt | rs2 | rd 010011 | OR            |
| shammt | rs2 | rd 010011 | AND           |
Problem 1.

Compile the following expressions to RISCV assembly. Assume \( a \) is stored at address 0x1000, \( b \) is stored at 0x1004, and \( c \) is stored at 0x1008.

1. \( a = b + 3c; \)

2. \( \text{if} \ (a > b) \ c = 17; \)

3. \( \text{sum} = 0; \)
   \( \text{for} \ (i = 0; \ i < 10; \ i = i+1) \ \text{sum} += i; \)
Problem 2.

Compile the following expression assuming that \( a \) is stored at address 0x1100, and \( b \) is stored at 0x1200, and \( c \) is stored at 0x2000. Assume \( a \), \( b \), and \( c \) are arrays whose elements are stored in consecutive memory locations.

\[
\text{for (i = 0; i < 10; i = i+1) c[i] = a[i] + b[i];}
\]
Problem 3.

A) Assume that the registers are initialized to: \( x1=8 \), \( x2=10 \), \( x3=12 \), \( x4=0x1234 \), \( x5=24 \) before execution of each of the following assembly instructions. For each instruction, provide the value of the specified register or memory location. **If your answers are in hexadecimal, make sure to prepend them with the prefix 0x.**

1. \( \text{SLL x6, x4, x5} \)
   Value of x6: ____________
2. \( \text{ADD x7, x3, x2} \)
   Value of x7: ____________
3. \( \text{ADDI x8, x1, 2} \)
   Value of x8: ____________
4. \( \text{SW x2, 4(x4)} \)
   Value stored: ________ at address: ________

B) Assume X is at address 0x1CE8

```
li x1, 0x1CE8
lw x4, 0(x1)
blt x4, x0, L1
addi x2, x0, 17
beq x0, x0, L2
L1: srai x2, x4, 4
L2:
```

Value left in x4? ____________

```
X: .word 0x87654321
```

Value left in x2? ____________
Problem 4.

Compile the following Fibonacci implementation to RISC-V assembly.

```python
# Reference Fibonacci implementation in Python
def fibonacci_iterative(n):
    if n == 0:
        return 0
    n -= 1
    x, y = 0, 1
    while n > 0:
        # Parallel assignment of x and y
        # The new values for x and y are computed at the same time, and then
        # the values of x and y are updated afterwards
        x, y = y, x + y
        n -= 1
    return y
```