LAB # 6
Load & Store Instruction Assembly Code

Objective:

- Get Familiar with Load & Store Instruction Assembly Code

System Module (Hardware/Software)

- Venus Simulator

• Theoretical Background:

Introduction:
RV32I is a load-store architecture, where only load and store instructions access memory and arithmetic instructions only operate on CPU registers. RV32I provides a 32-bit user address space that is byte-addressed and little-endian. The execution environment will define what portions of the address space are legal to access. Loads with a destination of x0 must still raise any exceptions and action any other side effects even though the load value is discarded.

Load Instructions:
Load instructions are used to move data from memory to registers (before operation). Loads are encoded in the I-type format. The effective byte address is obtained by adding register rs1 to the sign-extended 12-bit offset. Loads copy a value from memory to register rd. The assembly representation for load instructions are:

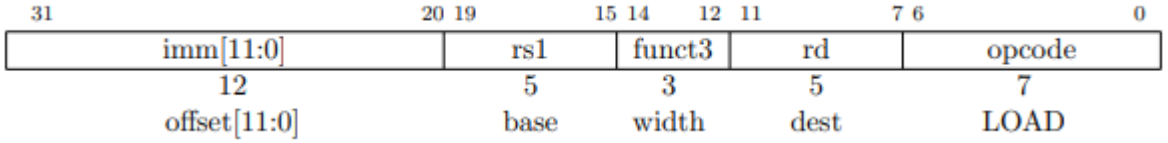
lw (destination_register), (offset)(source_register)
or
lw (rd), offset(rs1)

There are many types of load instructions. RISC-V support the following types

Types	Symbol
Load-Byte (signed)	LB
Load-Byte (unsigned)	LBU
Load-Halfword (signed)	LH
Load-Halfword (unsigned)	LHU
Load-Word	LW

The LW instruction loads a 32-bit value from memory into rd. LH loads a 16-bit value from memory, then sign-extends to 32-bits before storing in rd. LHU loads a 16-bit value from memory but then zero extends to 32-bits before storing in rd. LB and LBU are defined analogously for 8-bit values.

The instruction format of Load instructions is shown below:



Store Instructions:

Store instructions are used to move data from registers to memory (after operation). Stores are encoded in the S-type format. The effective byte address is obtained by adding register rs1 to the sign-extended 12-bit offset. Stores copy the value in register rs2 to memory.. The assembly representation for store instructions are:

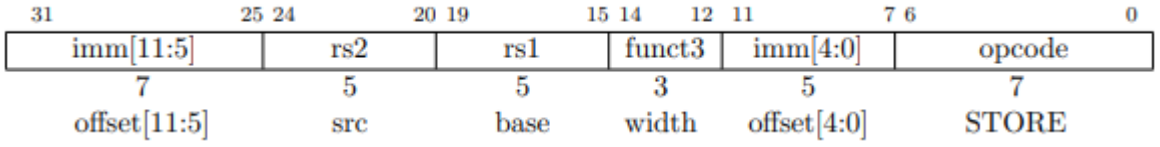
sw (source_register_2), (offset)(source_register_1)
or
sw (rs2), offset(rs1)

There are many types of store instructions. RISC-V support the following types

Types	Symbol
Store-Byte (signed)	SB
Store-Halfword (signed)	SH
Store-Word	SW

The SW instruction stores a 32-bit value from the low bits of register rs2 to memory. SH stores a 16-bit value from the low bits of register rs2 to memory. SB stores a 8-bit value from the low bits of register rs2 to memory.

The instruction format of Store instructions is shown below:



Lab Exercise 1:

Run the below assembly code on Venus Simulator

```
li s0, 0x12345678 # Data to be store
li s1, 0x00000020 # memory address
sb s0, 0x0(s1)
sh s0, 0x4(s1)
sw s0, 0x8(s1)
```

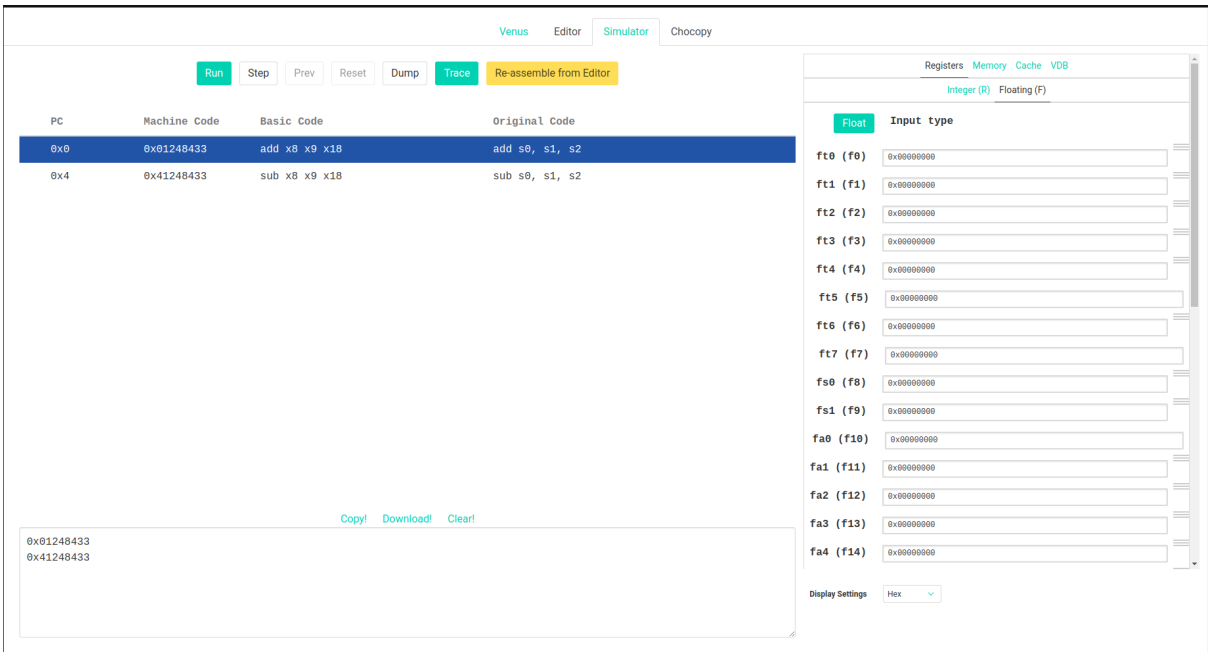
Lab Exercise 2:
Run the below assembly code on Venus Simulator

```
lb t0, 0x0(x0)
lbu t1, 0x4(x0)
lh t2, 0x8(x0)
lhu s0, 0xC(x0)
lw s1, 0x10(x0)
```

Procedure:

Venus User Interface:

1) Open the given link in Google Chrome. https://venus.cs61c.org/
2) Go to the Editor Tab to write the assembly code.
3) Go to the Simulator Tab to simulate your assembly code.
4) The Simulator screen is divided into three parts:
    a) **Register Display:** This shows the contents (bit patterns in hex) of all 32 general purpose registers, the floating point registers, and also the contents of memory.
    b) **Text Display:** This shows the assembly language program source, the machine instructions (bit patterns in hex) they correspond to, and the addresses of their memory locations.
    c) **Console:** This shows the machines code dump from the simulator



Running a program:

1) After writing your assembly code in Editor Tab go to Simulator Tab.
2) Click on **Assemble & Simulate from Editor** button.
3) To run the whole code at once click on **Run** button.
4) To run the code step by step click on the **Step** button.
5) To dump the machine code click on the **Dump** button.

6) To Reset the value to default click on the **Reset** button.
7) To go back to the previous step click on the **Prev** button.
8) To reassemble the code after changes click on the **Re-assemble from Editor** button.

In-Lab Tasks:

1. **Write down a simple assembly program to add, and subtract two integer numbers and store their result into different memory locations. Stimulate the code on Venus.**
2. **Write down a simple assembly program to load the contents from memory into registers and perform the logical operations on them. Stimulate the code on Venus.**

Conclusion:

What have you learnt from this lab?

_____

_____

_____

_____

Learning Outcomes:

Upon successful completion of the lab, students will be able to:

LO1: Run the basic assembly code on Venus Simulator.