

32-Bit CPU in Logisim

I. Processor Description:

The 32-bit CPU is implemented in Logisim as a Simple RISC Processor, assembled with the following components:

1. ALU unit with 32-bit Adder & Subtractor, AND & OR logic operations
2. Register Files (or “Register Bank”)
3. Control Unit at least generating control signals for the above ALU operations
4. Calculate Immediate (from 32-bit Instruction Format)
5. Memory Unit
6. Separate Instruction and Data Memory (using RAM)

Additional subsystems include:

1. Control Buffers
2. Instruction Register
3. Program Counter connected to Instruction Memory

The 32-bit CPU is a non-pipeline approach. The CPU has a load/store architecture with addressing between the instruction decoder gates and the data memory. System clocks are directly wired to the clock pins of memory units. The 32-bit ALU has 32 1-bit ALU units and performs addition, subtraction, AND, and OR operations. The outputs of the register bank are wired as inputs of the ALU. Bit extenders between tunnels and main pins are used for confirming bit compatibility across the CPU.

To start the CPU, download the CIRC file in Logisim and click on the main circuit named “CPU.” Simulate the program by clicking on the “Write” Pin or by simulating any of the 32-bit input pins named “A” or “B” by toggling between 0 and 1 and vice versa.

II. Starting Guide:

Process for Programming Instruction Memory with Hex Numbers:

- “*Value*” corresponds to the literal numbers
- Writing a “*Value*” to the register sends the 8-bit literal numbers from the Control Unit to the Register Bank
- **Load/Store operation:** “*Addr*” in Data Memory must be specified: address bits go to the Data Memory from the Control Unit
- The Instruction Decoder decodes the instruction type per 3-bit gate (turns on control bits and control buffers and sets read/write bits)
- **Store operation:** storing data into Data Memory from Register Bank through the buffer
 - HIGH “*Value*” signals bits in Data Memory to be written in “*Addr*” extracted from Control Unit

- **Data operation:** HIGH “Data” signals A and B to ALU from Register Bank and returns to control buffer back to the Register Bank
 - To Write: Set “Data” bit HIGH to OR Gate configured to Register Bank
 - Set Destination Register
- **Load Operation:** “Load” bit in the Instruction Decoder of the Control Unit is HIGH, loaded into Data Memory, triggered “Addr” goes through control bugger; Register Write is enabled into Register Bank
 - The “Load” literal is HIGH and stored as the 8-bit literal in the “Value” of the Control Unit
- The instruction is “fetched” from the memory and the entered values loaded to the registers are “fetched” from the registers along with the immediate value from the ALU calculations that follow the operations in the instruction

Instruction Format: 3 types of instructions: 1. LDR, 2. STR, and 3. Op

LDR Rd, =N

Load 8-bit literal number N into register

LDR Rd, [A]

Load 8-bit value from RAM (10-bit address A) into register

STR Rs, [A]

Store value in register in RAM (10-bit address 10)

op Rd, Ra, Rb

Rd = Ra operation Rb

Example:

LDR R2, =10 //Loads R2 = 10: 0x601C

LDR R3, =25 //Loads R3 = 25: 0x6016

ADD R1,R2,R3 //Performs Addition of R1 = 10+25 = 35: 0x2066

SUB R4,R3,5 ./Performs Subtraction of R4 = 25-5 = 20: 0x8039

from

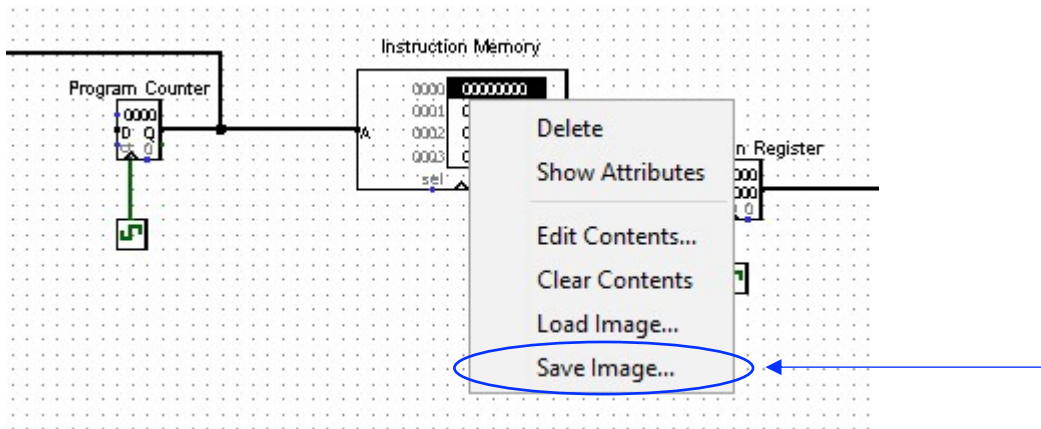
0000	0000	0000	0000	0110	0000	0001	1100	0x601C
0000	0000	0000	0000	0110	0000	0001	0110	0x6016
0000	0000	0000	0000	0010	0000	0110	0110	0x2066
0000	0000	0000	0000	1000	0000	0011	1001	0x8039

Using “Save Image” Protocol to Program Data in Logisim through Text Files:

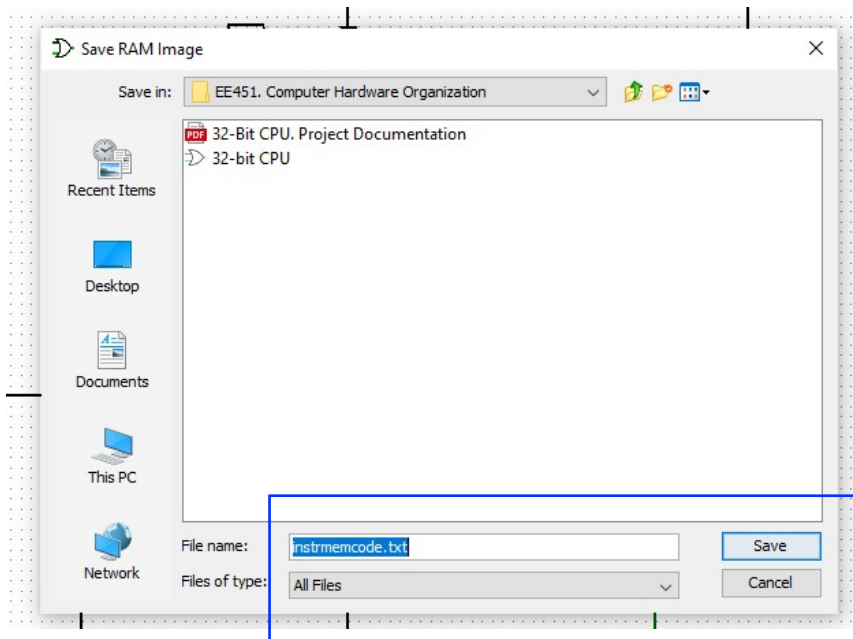
The following protocol can be implemented in place of writing code in C++ to compile (using a compiler, to program into the CPU) or in place of writing code in Assembly Language (through an Assembler) to program the machine code into the CPU.

Steps Using the Assembly Code Example (from above):

1. Access the CPU in Logisim once opening the submitted “32-bit CPU.circ” file in Logisim
2. Right-click the Instruction Memory RAM unit and select “Save Image”:



3. Name the text file “instrmemcode.txt” and save it on the desktop:

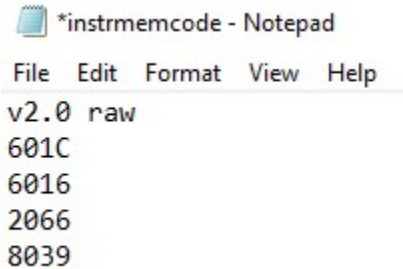


4. Minimize Logisim and access the .txt file from the desktop:

<input type="checkbox"/> Name	Date modified	Type	Size
32-Bit CPU. Project Documentation	12/4/2022 10:11 AM	Microsoft Edge PDF Doc...	55 KB
32-bit CPU	12/5/2022 9:47 PM	CIRC File	169 KB
instrmemcode	12/5/2022 9:51 PM	Text Document	1 KB

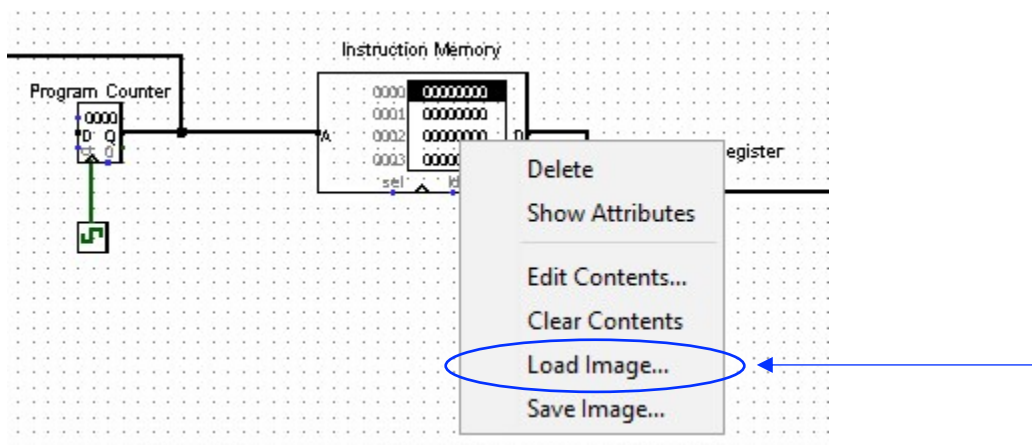
Note: The .txt file will be automatically saved to the same desktop folder that the Logisim file is already saved in. The file can be named anything else. This example uses “instrmemcode.txt” as the example name.

- Keep the default header and start writing the hex numbers beginning from the next line below the default header
- Write the hex numbers from the assembly code example above (each hex number per line in the .txt file), without writing the “0x” per hex number:

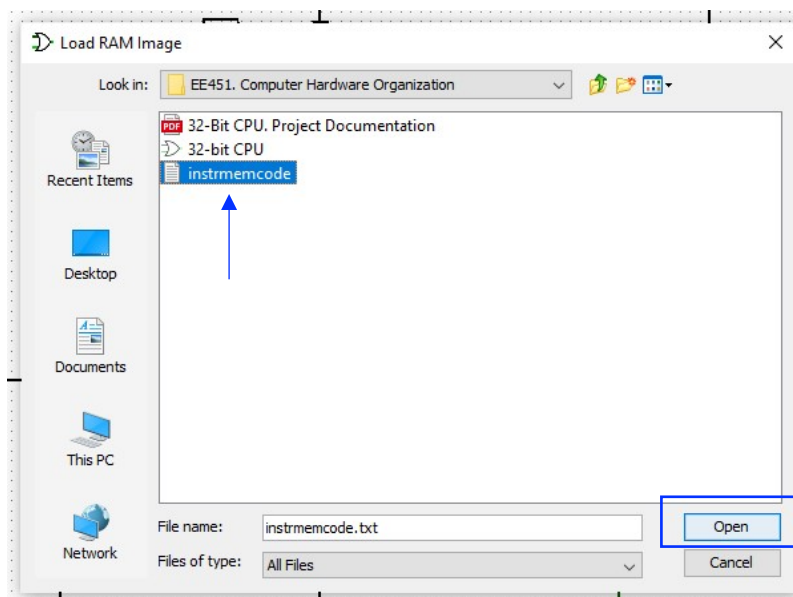


```
*instrmemcode - Notepad
File Edit Format View Help
v2.0 raw
601C
6016
2066
8039
```

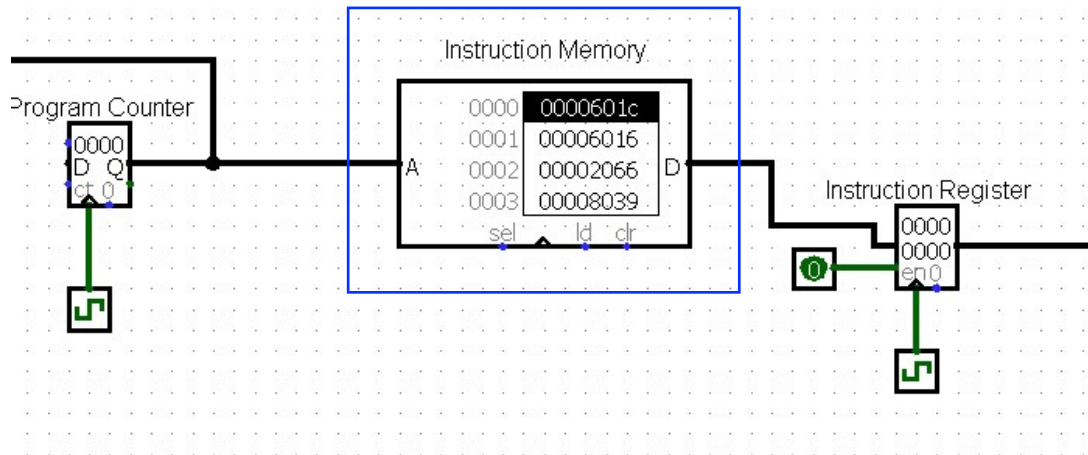
- Save the code image
- Return to the Logisim CPU file
- Right-click on the Instruction Memory and select “Load Image”:



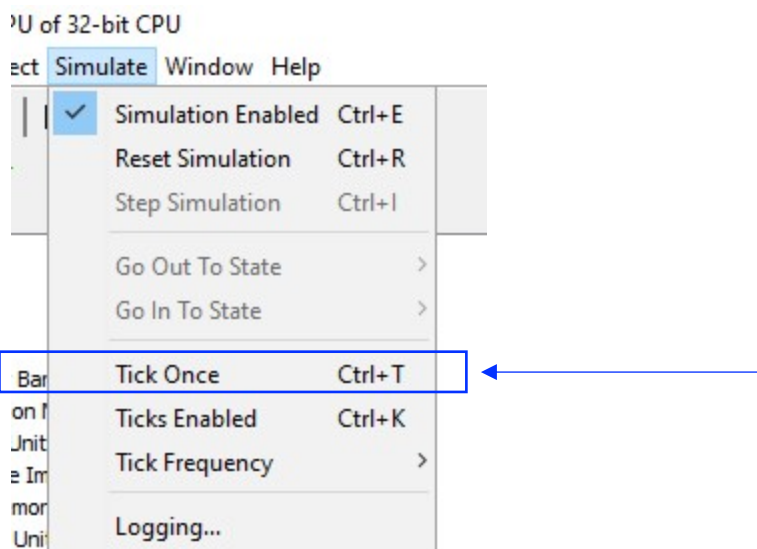
- Load the “instrmemcode.txt” file by selecting the file and clicking “Open”:



11. Confirm the loading of the code image by viewing the hex numbers in the Instruction Memory:



12. Click on “Simulate” at the top menu bar and click “Tick Once”:



13. Repeat step 12 and observe as the hex numbers are loaded and stored into the Register Bank using the CPU clocks as part of the Instruction Set
14. To clear instructions, click on “Simulate” at the top menu bar and click “Reset Simulation”
15. Similarly, data can also be loaded into the Data Memory