CPU Name: Goated CPU
Creator: Kazi Tishan

How to use the assembler

Make the following 4 files and make sure they are in the same directory as the assembler:

1) instr.txt
   - This will hold all the instructions that you would like to execute
   - The instructions supported are ADD, SUB, LDR, and STR.
   - Make sure that lines separate instructions, for example:

     ```
     ADD X0, X0, 3
     SUB X0, X0, 1
     ```
2) instr-img.txt
   - This will be the image file for the Instruction Memory, generated by the assembler.
   - This image file is to be loaded into the Instruction Memory.
   - This file must be empty before running the assembler.
3) data.txt
   - This will hold all the data that you would like your Data Memory to hold.
   - The supported data includes all numbers from 0 to 255 inclusive
   - Make sure that commas separate data, for example:

     ```
     10, 2, 3
     ```
4) data-img.txt
   - This will be the image file for the Data Memory, generated by the assembler.
   - This image file is to be loaded into the Data Memory.
   - This file must be empty before running the assembler.

After these four files are in the same directory as the assembler, you can execute the assembler by running the following in the command line:

```
python3 assembler.py
```

OR you could open the directory in VSCode and open the assembler file. There should be a play button at the top right. If you press it, the assembler will execute.

After the assembler executes, you can now load instr-img.txt into the Instruction Memory in the CPU and data-img.txt into the Data Memory in the CPU. Enable auto-tick, start the simulation, and it should work as intended.

Architecture of the CPU

- The register file of this CPU contains four general-purpose registers and three read-ports. The three read ports are necessary here because there are some instructions that require three register reads.
- Registers can be referred to in the assembly program like this: X0, for the first register, X1, for the second register, X2, for the third register, and X3, for the fourth register.
- This CPU can add, subtract, load data from memory, and store data to memory
- There are LEDs to show which instruction is currently being passed through

- The Program Counter stops counting when it encounters two ‘ffff’ instructions in a row. ‘ffff’ is not an instruction that can ever occur with the assembler and binary encodings. Read more from the next section below to find out why.
- There is a pin that resets the values of the PC and all the registers. This allows you to rerun your instructions without having to reset the simulation. To use this, you must click on the pin to change the value of the pin from 0 to 1, let it rest in this position after a few clock cycles to ensure that there is no data in the wires. Then, click on the pin to change the pin from 1 to 0, to rerun your instructions without any mistakes.

Instructions and binary encodings

bit	15-14	6-13 for instructions with imm 7-13 for instructions with no imm	5 for instructions with imm 5-6 for instructions with no imm	3-4	2	0-1
Instruction	Opcode	Imm	Rm	Rd	Imm?	Rn
ADD Rd, Rn, Rm	00	0000000	-	-	0	-
ADD Rd, Rn, imm	00	-	0	-	1	-
SUB Rd, Rn, Rm	01	0000000	-	-	0	-
SUB Rd, Rn, imm	01	-	0	-	1	-
LDR Rd, [Rn, Rm]	10	0000000	-	-	0	-
LDR Rd, [Rn, imm]	10	-	0	-	1	-
STR Rd, [Rn, Rm]	11	0000000	-	-	0	-
STR Rd, [Rn, imm]	11	-	0	-	1	-

Opcode: 2 bits are required for the opcode because there are 4 different instructions that this CPU can run: ADD, SUB, LDR, and STR.

Imm: If an instruction uses an immediate number, 8 bits are needed for that number because register values can hold up to 8 bits, and the address width in data memory is 8 bits.

Rm: This is simply the last register referred to in the instruction from left to right. 2 bits are needed to represent this register because there are 4 possible registers to refer to.

Rd: This is simply the first register referred to in the instruction from left to right. 2 bits are needed to represent this register because there are 4 possible registers to refer to.

Imm?: This is a flag to determine if an instruction is using an immediate number. If an immediate number is being used, the value of this will be 1. However, if the value of this is 0, an immediate number is not being used.

Rn: This is simply the second register referred to in the instruction from left to right. 2 bits are needed to represent this register because there are 4 possible registers to refer to.

Why ‘ffff’ is not a possible instruction to be generated by the assembler
‘ffff’ means that all the bits are 1. This would mean that the second bit is 1, and that the fifth bit is 1. However, if we look at the binary encodings above, we notice that there are no possible instructions where bit 2 is 1 and where bit 5 is 1. This makes ‘ffff’ an easy way to determine the end of the instructions, which is what I did with my assembler.

For the demo:

- Use instr.txt and data.txt