Anthony Eryan
“I pledge my honor that I have abided by the Stevens Honor System” -A. Eryan

OffBrandARM CPU

10th December 2024

OVERVIEW

OffBrandArm CPU is an eight-bit CPU that has a 2^2 binary representation of 4 total instructions, which can do arithmetic and memory-related operations, based off the Arm® architecture we learned in CS 382!

GOALS

1. Execution of addition and subtraction operations given 8-bit instructions via registers.
2. Execution of LDR and STR with both immediates and registers via recognizing the .data and .text segment in the assembler.

SPECIFICATIONS

The two most significant bits are recognized as operations, where there are four general purpose registers that are considered to be Rn, Rm, Rt. Please consult the table if you wish to get technical:

Instruction	Opcode	Rt (in binary) WriteReg	Rn (in binary) ReadReg2	Rm (in binary) ReadReg1
ADD	00	--	--	--
SUB	01	--	--	--
LDR	10	--	--	--
STR	11	--	--	--

Thus, the next two bits after the opcode are Rt (bit 5-4), Rn (bit 3-2), and lastly Rm (bit 1-0), where the target register is Rt (the register that gets written to), Rn is the 2nd register

read and the Rm is the 1st register read in order to perform the operation. The registers are recognized as 00 (X0), 01 (X1), 10 (X2) and 11 (X3). The aforementioned instructions can internally be made aware by binary representation of their flag:

Instruction\Flags	WriteReg	ReadMem	WriteMem	ALUOp
ADD	1	0	0	0
SUB	1	0	0	1
LDR	1	0	0	0
STR	0	0	1	0

INSTRUCTIONS

ADD (`ADD Rt = Rn,Rm`)

Performs arithmetic operation addition upon ReadReg2 + ReadReg1. Here's an exact example:

```
ADD X0 = X1,X2
```

Where X1 adds from X2 and then the result is then written to the X0 register.

SUB (`SUB Rt = Rn,Rm`)

Performs arithmetic operation subtraction upon ReadReg2 - ReadReg1. Here's an example:

```
SUB X3 = X1,X0
```

Where X1 subtracts from X0 and then the result is then written to the X3 register.

LDR (`LDR Rt = [Rn:Rm]`,`LDR Rt = [Rn,immediate]`)

Loads the value of a register with an offset of a register (or immediate) into a destinated register. Here's two examples with one being an immediate and one being a register:

```
LDR X1 = [X2:4]
```

```
LDR X1 = [X2:X3]
```

STR (`STR Rt = [Rn:Rm]`,`STR Rt = [Rn,immediate]`)

Stores the target register with the value of ReadReg2 with an offset of an immediate or register value. Here's two examples with one being an immediate and one being a register:

```
STR X1 = [X2:4]
```

```
STR X1 = [X2:X3]
```

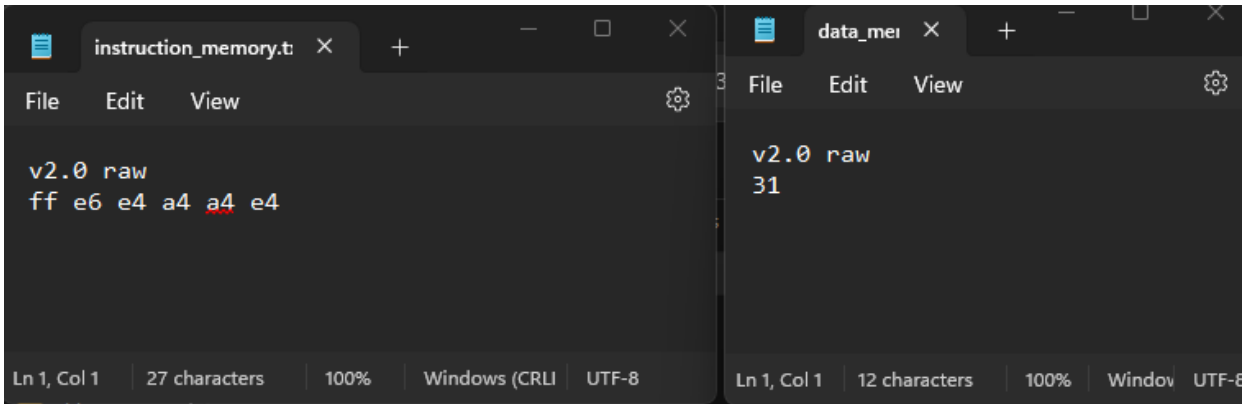
Note: Given OffBrandARM is an 8-bit CPU, immediates are expected to be constrained to the two hexadecimal numbers, so the range is 0-255 (FF in hexadecimal).

ASSEMBLER

Running assembler.py is as conventional as it gets when running scripts on your own computer. Running it on the command line is as easy as any other program you would type `assembler.py` (may slightly vary on your Operating System) into your command line or execute the file in your preferred IDE.

```
on.exe "c:/Users/mrepi/OneDrive - stevens.edu/shared/CS382/Project2/assembler.py"assembler.py
Welcome to the OffBrandARM assembler!
Creating files in: C:\Users\mrepi\OneDrive - stevens.edu\shared\CS382\Project2

Instruction Format Examples:
  ADD X0 = X1,X2     (Add X1 and X2, store in X0)
  SUB X3 = X1,X0     (Subtract X0 from X1, store in X3)
  LDR X1 = [X2:4]    (Load into X1 from address X2+4)
  LDR X1 = [X2:X3]   (Load into X1 from address X2+X3)
  STR X1 = [X2:4]    (Store X1 to address X2+4)
  STR X1 = [X2:X3]   (Store X1 to address X2+X3)


Enter instruction (or 'q'/'quit' to exit):
ADD X0 = X1,X2
Binary: 00000110
Hex: 06

Enter instruction (or 'q'/'quit' to exit):
STR X2 = [X1:49]
Binary: 11100100
Hex: e4

Enter instruction (or 'q'/'quit' to exit):
STR X3 = [X3:X3]
Binary: 11111111
Hex: ff

Enter instruction (or 'q'/'quit' to exit):
SUB X1 = X0,X3
Binary: 01010011
Hex: 53

Enter instruction (or 'q'/'quit' to exit):
q

Instructions written to C:\Users\mrepi\OneDrive - stevens.edu\shared\CS382\Project2\instruction_memory.txt
Immediates written to C:\Users\mrepi\OneDrive - stevens.edu\shared\CS382\Project2\data_memory.txt

Assembler successfuly terminated.
PS C:\Users\mrepi\OneDrive - stevens.edu\shared\CS382\Project2>
```

Given the screenshot attached, the assembler is as simple as following the syntax in the instructions section. Per correct instruction, it will continually ask the user to input their selective instruction, and once the user quits, it will write to instruction_memory.txt and depending on the memory-related instruction, it will also output data_memory.txt which will generate an additional image file that will be used for DataMemory RAM:



This is where you must find both files from the directory you stored the assembler, and

input them to their correct places in memory in the `.circ` file. Load the `instruction_memory.txt` into the InstructionMemory RAM and load the data_memory.txt into the DataMemory RAM in the attached `.circ` file (if you used immediates). Now, you are ready to tick the clock of the CPU per instruction in OffBrandARM CPU!

Note: Given the tight constraints of 8-bit CPU instruction, you may have noticed that when using an immediate in memory-related instruction as LDR or STR, the last two bits are 00. This means that the Rm register is X0. Therefore, it is to be noted that X0 serves a dual purpose: being a general-purpose register for instructions aforementioned and a flag for having an immediate and then appending the appropriate immediate value into `data_memory.txt`.