CPU and Assembler

User Manual: by Lili Fortes and Sean Payba

I pledge my honor that I have abided by the Stevens Honor System.

Job description:

Sean: Created the CPU in Logisim, wrote CPU architecture description and Instruction set portion.

Lili: Wrote the code for "makefile.py" (AKA the assembler) and wrote demo code/data segments in "instructions.txt". Wrote assembler instructions.

How to use the assembler:

Our assembly language has 6 instructions: "LDR Rt, [Rn, Rm]", "LDR Rt, [Rn, imm6]", "ADD Rd, Rn, Rm", "ADD Rd, Rn, imm6", "SUB Rd, Rn, Rm", and "SUB Rd, Rn, imm6". Each instruction's usage is detailed in the CPU architecture section.

To create our assembly program, you must have "makefile.py" in the same folder as an "instructions.txt" file so that the makefile can read instructions and data from the instruction file. The general structure of the instruction file must be a .text segment at the top containing all of your instructions, and a .data section under the .text section containing addresses and the data you plan to store in them.

In the .text section, you write the instructions using our mnemonics as described. Each instruction must be on its own line. You can make comments in your code both in line and on their own lines using "//" preceding your comment. Any text between "//" and the end of the line is ignored by the assembler.

Sample .text section:

```
.text
     LDR X1, [X0, 7] // Load value at address 7 to X1 (X1 = 32)
     LDR X2, [X0, 3] // Load value at address 3 to X2 (X2 = 3)
     ADD X0, X1, X2 // Add X1 + X2 (X0 = 35)
     ADD X3, X3, 3 // Add X3 + 3 (X3 = 3)
     SUB X2, X1, X3 // Subtract X1 - X3 (X2 = 29)
     // this is a sample comment
     SUB X0, X1, 2 // Subtract X1 - 2 (X0 = 30)
```

We can use the .data section to add values directly to memory. You must mark this section with .data, if you decide to use it. Each data declaration is formatted as such:
`Address: data at address, data at address+1, data at address+2,…`
When declaring data at different addresses, separate different locations on different lines. Different addresses must be in chronological order.

Sample .data section:

```
.data
     0: 0, 1, 2, 3, 4, 5
     7: 32, 4, 6, 52, 3 // you may make comments on the data!
```

After you have finished writing your assembly program, you can assemble it by running the makefile in the same folder as your instruction file. When this happens, an imagefile will be produced with the machine code of your instructions, as well as a memoryimage containing the information from your .data segment. You must load the imagefile to instruction memory and the memoryimage to data memory to run your program in Logisim.

CPU Architecture:
In the CPU, we used 4 general purpose registers, which can be referred to in an assembly program with X registers from 0 to 3. These can be encoded in 2 bits, so all the register files take 2 bits. Each register can hold up to 8 bits of data.

The CPU is capable of adding, subtracting, and loading data with registers or immediate numbers for the offset. Note that the immediate values use 6 bits, so the maximum value you may use as an immediate is 63, and the minimum is 0.

Instruction Set:
In general, we need 1 bit for the first field to determine whether we are adding or subtracting (0 for adding, 1 for subtracting), 1 bit for the second field to determine if we are loading or not, 1 bit for the third field to determine if we are using immediate numbers, 2 bits for the fourth, fifth, and sixth fields to determine the destination, first, and second registers respectively, 6 bits for the seventh field for the size of the immediate number (if it is being used), and 1 bit for the eighth field to determine whether we will continue the program or not.

Instruction 1: LDR Rt, [Rn, Rm]

Binary Encoding: 0 1 0 Rt Rn Rm imm6 enable

Binary encoding	0	1	0	Rt	Rn	Rm	imm6	enable
Num bits	1	1	1	2	2	2	6	1

For LDR with a second register offset, the first bit is 0 to demonstrate that we are adding, the second bit is 1 to demonstrate that we are loading, and the third bit is 0 since we are not using immediates. We will be using Rt, Rn, and Rm. Rt is the register we are loading to, Rn is the address of the data we are loading from memory, and Rm is the offset that is added to Rn. The 6 bits for immediates will not be used in this instruction, and the last bit is 1 since we are running this instruction.

Instruction 2: LDR Rt, [Rn, imm6]
Binary Encoding: 0 1 1 Rt Rn Rm imm6 enable

Binary encoding	0	1	1	Rt	Rn	Rm	imm6	enable
Num bits	1	1	1	2	2	2	6	1

For LDR with an immediate offset, the first bit is 0 to demonstrate that we are adding, the second bit is 1 to demonstrate that we are loading, and the third bit is 1 since we are using immediates. We will be using Rt, Rn, and imm. Rt is the register we are loading to, Rn is the address of the data we are loading from memory, and Rm will not be used in this instruction. Imm is the offset that is added to Rn, and the last bit is 1 since we are running this instruction.

Instruction 3: ADD Rd, Rn, Rm
Binary Encoding: 0 0 0 Rd Rn Rm imm6 enable

Binary encoding	0	0	0	Rd	Rn	Rm	imm6	enable
Num bits	1	1	1	2	2	2	6	1

For ADD between a register and an immediate, the first bit is 0 to demonstrate that we are adding, the second bit is 0 to demonstrate that we are not reading from memory, and the third bit is 0 since we are not using immediates. We will be using Rd, Rn, and Rm. Rd is the register that will hold the result of addition, Rn is our first register we are adding, and Rm the second. The 6 bits for immediates will not be used in this instruction, and the last bit is 1 since we are running this instruction.

Instruction 4: ADD Rd, Rn, imm6
Binary Encoding: 0 0 1 Rd Rn Rm imm6 enable

Binary encoding	0	0	1	Rt	Rn	Rm	imm6	enable
Num bits	1	1	1	2	2	2	6	1

For ADD between two registers, the first bit is 0 to demonstrate that we are adding, the second bit is 0 to demonstrate that we are not reading from memory, and the third bit is 1 since we are using immediates. We will be using Rd, Rn, and Rm. Rd is the register that will hold the result of addition, Rn is our first register we are adding, and Rm will not be used in this instruction. Imm will be used in this instruction to add to Rn, and the last bit is 1 since we are running this instruction.

Instruction 5: SUB Rd, Rn, Rm
Binary Encoding: 1 0 0 Rd Rn Rm imm6 enable

Binary encoding	1	0	0	Rt	Rn	Rm	imm6	enable
Num bits	1	1	1	2	2	2	6	1

For SUB between two registers, the first bit is 1 to demonstrate that we are subtracting, the second bit is 0 to demonstrate that we are not reading from memory, and the third bit is 0 since we are not using immediates. We will be using Rd, Rn, and Rm. Rd is the register that will hold the result of addition, Rn is our first register we are adding, and Rm is being subtracted from Rn. The 6 bits for immediates will not be used in this instruction, and the last bit is 1 since we are running this instruction.

Instruction 6: SUB Rd, Rn, imm6

Binary Encoding: 1 0 1 Rd Rn Rm imm6 enable

Binary encoding	1	0	1	Rt	Rn	Rm	imm6	enable
Num bits	1	1	1	2	2	2	6	1

For SUB between a register and an immediate, the first bit is 1 to demonstrate that we are subtracting, the second bit is 0 to demonstrate that we are not reading from memory, and the third bit is 1 since we are using immediates. We will be using Rd, Rn, and Rm. Rd is the register that will hold the result of addition, Rn is our first register we are subtracting, and Rm will not be used in this instruction. Imm will be used in this instruction to be subtracted from Rn, and the last bit is 1 since we are running this instruction.