Arin Ravindran && Jonathan Pruchansky
I pledge my Honor that I have abided by the Stevens Honor System.
SHIFTYCPU - Instruction Manual
All implementations designed for potential EC are marked with yellow

Job Description

Register File Implementation - Jonathan Pruchansky
1.) Used the Lab 08 Register File - Tweaked the Fourth Register to be a Zero Register
2.) Sent out an output of a Control Signal StoreRegData signal and two Data Signals

ALU Implementation - Arin Ravindran
1.) Used a negator to turn the second input into a negative number, incase of subtraction
2.) Used an Multiplexer to determine addition or subtraction using the ALUOP control signal
3.) Sent both outputs through the adder for the final ALU output

RAM and PC Implementation - Jonathan Pruchansky
1.) Created two rams in the circuit, one for the Instructions (2 Bytes) and the other for Data
2.) Wired the Program counter with an automatic Adder for our single cycle circuit

Immediate Value Implementation - Arin Ravindran
1.) Wired the Immediate Values (Last 7 bits) along with RegOutput 2 to a Multiplexer
2.) Used a control signal (Bit between last Register and Immediate/Throwaway Bits) to determine which value to pass to the ALU, either the Register value or the Immediate
3.) Able to use MOV commands aswell as Addition or Subtraction

Control Unit Implementation - Arin Ravindran
1.) Used Logic Gates to generate 5 Control Signals based on instruction and register design

Assembler Implementation - Jonathan Pruchanksy
1.) Coded the Assembler in Python

Splitting and Opcode Design - Jonathan Pruchansky
1.) Splits the instruction data into 16 bits

Assembler

1.) Github Repository Link: https://github.com/tigritik/Shifty-CPU

Program

1.) Our program calculates the Triangular Number of a preset data value, in our case 5. The Triangular number of 5 is 15, which at the end of the program will be stored in the 5th place in ram.

```
MOV b 0 0          // Initializes the b register to 0, this is the summation
MOV c 0 0          // Initializes the c register to 0, this is counter
LDR a c 0          // Loads in tthe value of byte (a)
Add c c 1          // Increments C by 1, c is now 1
Add b b c          // B + C, sum is now 1
Add c c 1          // Increments C by 1, c is now 2
Add b b c          //  B + C, sum is now 3
Add c c 1          // Increments C by 1, c is now 3
Add b b c          // B + C, sum is now 6
Add c c 1          // Increments C by 1, c is now 4
Add b b c          // B + C, sum is now 10
Add c c 1          // Increments C by 1, c is now 5
Add b b c          // B + C, sum is now 15
Str b d c          // Stores the sum in B into the c (5th) place in RAM
data
byte 5
```

Architecture

CPU Breakdown

1.) 2 Rams | One for the Instructions and one for the Data
    a.) Instruction RAM
        i.) Made to handle 2 Bytes of Instruction
    b.) Data RAM
        i.) Made to handle 1 Byte of Data
2.) Register File
    a.) 4 General Purpose Registers
        i.) Registers A B C and a Zero Register
3.) One ALU and one Control Unit
    a.) ALU
        i.) ALU can handle Addition and Subtraction

(1) Takes a ALUOP Control signal and 2 Data Inputs
(2) Uses a Multiplexer, an Adder, and a negator
(3) Sends Input #1 to the adder
(4) Branches the second input into a negative and positive
(5) Uses the ALUOP control signal to determine whether to use positive (Addition) or negative (Subtraction)
(6) Sends both numbers to adder and returns

b.) Control Unit
i.) Takes the opcode as input and puts it through the gates
ii.) The NOT gates are designed to flip certain bits in order to reach the desired added output
iii.) Generates Control Signals
(1) RegW - Indication if we are writing to a Register
(2) MemW - Indication if we are writing to Memory
(3) MemR - Indication if we are reading from Memory
(4) ALUOp - Indication of the ALU operation, 0 for addition, 1 for subtraction
(5) WriteCntrl - If we are writing from RAM or from ALU output

4.) Splitter System
a.) Splits 16 bits into 6 segments
i.) Bit 15-14: Opcode - routed to the Control Unit
ii.) Bit 13-12: The Store Register - routed to WriteReg and StoreReg
iii.) Bit 11 -10: The first Read Register - routed to ReadReg 1
iv.) Bit 9-8: Second Read Register - routed to ReadReg2
v.) Bit 7 - Immediate Control Bit, routed to the Immediate Multiplexer
vi.) Bit 6 - 0 - Immediate Data (Filled with numbers incase of no immediate
b.) Sign Extension
i.) Uses a sign extension block to convert the 7 bit immediate to 8 bits as input to the multiplexer
c.) Multiplexer
i.) Check whether to assign value as an immediate or use the register data in RegData 2 output for the

Functions
1.) Addition (ADD) - between two registers or an register and an immediate
2.) Subtraction (SUB) - between two registers or an register and an immediatre
3.) Moving (MOV) - moving an immediate into a register, using the zero register
4.) Loading (LDR) - loading from ram into a register
5.) Storing (STR) - Storing from a register into a ram

OPCODE Breakdown

Opcode	Bits A | B	RegW	MemW	MemR	AluOP	WriteCntrl
LDR						
STR						
ADD						
SUB						

MOV A 5

Instruction Format:

1.) Opcode - 2 Bits
2.) Register 1 - 2 Bits
3.) Register 2 - 2 Bits
4.) Register 3 - 2 Bits
5.) Immediate Control - 1 Bit (0 is for Immediate use, one is for register use)
6.) Immediate - Signed 7 Bits - Extended to 8 bits using Signed extension for ALU

Immediate is wired to 0, the register is wired to one for the Multiplexer