CS3339 Project 2 rev3

Description: In this project, you will extend your ARM disassembler with a simulator.

Part 2: Simulator

Your program will create an instruction-by-instruction simulation of the ARM program. This simulation will execute instructions sequentially (non-pipelined) and output the contents of all registers and memory (the state of the processor and memory) after each instruction. You will not have to implement exception/interrupt handling.

Instructions: (all arg refs to machine not assembly instruction - based on my code - your milage may vary) I will add more as I completely test them in my code.

B: Extends and shifts (multipies by 4) the 26 bit argument (words) in arg1 and adds the value to the PC.

CBZ, CBNZ: Does comparison against zero of arg2 and if condition met adds extended and shifted offset to the PC. Offset can be positive or negative. Offset in words.

ADDI: Adds extended immediate value to the value in arg1 register and puts value in arg3 register. Immediate can be positive or negative.

SUBI: Subtracts extended immediate value from the value in arg1 register and puts value in arg3 register. Immediate can be positive or negative

ALL R FORMAT: arg2 register <operation>arg1 register result into arg3 register.

MOVZ: 16 bit pattern in arg3 register shifted left by either 0,16,32,48 positions determined by 2 bit arg1 value and written into zeroed arg2 register

MOVK: 16 bit pattern in arg3 register shifted left by either 0,16,32,48 positions determined by 2 bit arg1 value and written into arg2 register leaving all other bits intact.

ASR: This is ">>" will divide by 2.

LSR: This is a pattern shift. Whatever is in register is shifted right with zero fill.

LSL: Shift left arg2 positions adding zeros on the right.

The simulation file will have the following format:

- 20 equal signs and a newline
- cycle: [cycle number] [tab] [instruction address] [tab] [instruction string (same as step 3 above)]
- [blank line]
- registers:
- r00: [tab] [integer value of R00][tab] [integer value of R01][tab] …[integer value of R07]
- r08: [tab] [integer value of R08][tab] [integer value of R09][tab] …[integer value of R15]
- r16: [tab] [integer value of R16][tab] [integer value of R17][tab] …[integer value of R23]
- r24: [tab] [integer value of R24][tab] [integer value of R25][tab] …[integer value of R31]
- [blank line]
- data:
- [starting data address]: [tab] [show in blocks of max 8 data words, with tabs in between]
- …[continue until last data word]

Instructions and arguments should be in capital letters. All integer values should be in decimal. Immediate values should be preceded by a # sign. Be careful and consider which instructions take signed values and which take unsigned values. Be sure to use the correct format depending on the context.

***Displaying Data:**

Data is displayed in sets of 8 data words. You should be able to display all data in memory. Obviously whatever data you start with needs to be displayed but more importantly if the executing program writes to memory locations beyond the already used memory, your program and output will need to reflect that by creating more 8 word display lines with appropriate starting data address until you can display this new data. Assume that they are created initialized to zero. (Normally I would initialize to 0xDEADBEEF but that makes a messy output)

NEW OUTPUT REQUIREMENTS!

Your program will produce 2 output files named <command line output argument>_dis.txt, which contains the disassembled program code for the input ARM machine code, and <command line output argument>__sim.txt which id the simulation output,. So use the command line provided output file name like you did for dis and make your sim file be the same way. Append _sim to the command line output file input.