CSF 2019 Programming Assignment #3

Overview

For this assignment, you will be writing a program in C or C++ that simulates the contents of a file containing a 256 byte SCRAM program. The extended SCRAM instructions that you must decode are:

Mnemonic	Encoding								C updated?	Comments
`HLT`	0	0	0	0	0	0	0	0	no	
`EXT`	0	0	0	0	b_7	b_6	b_5	b_4	no	$b_i \neq 0$
`LDA`	0	0	0	1	b_3	b_2	b_1	b_0	no	A ← M[b]
`LDI`	0	0	1	0	b_3	b_2	b_1	b_0	no	A ← M[M[b]]
`STA`	0	0	1	1	b_3	b_2	b_1	b_0	no	M[b] ← A
`STI`	0	1	0	0	b_3	b_2	b_1	b_0	no	M[M[b]] ← A
`ADD`	0	1	0	1	b_3	b_2	b_1	b_0	yes	A ← A + $\overline{\text{M[b]}}$
`SUB`	0	1	1	0	b_3	b_2	b_1	b_0	yes	A ← A + $\overline{\text{M[b]}}$ + 1
`JMP`	0	1	1	1	b_3	b_2	b_1	b_0	no	PC ← b
`JMZ`	1	0	0	0	b_3	b_2	b_1	b_0	no	PC ← b if A=0
`AND`	1	0	0	1	b_3	b_2	b_1	b_0	no	A ← A ∧ M[b]
`IOR`	1	0	1	0	b_3	b_2	b_1	b_0	no	A ← A ∨ M[b]
`XOR`	1	0	1	1	b_3	b_2	b_1	b_0	no	A ← A ⊕ M[b]
`ADL`	1	1	0	0	b_3	b_2	b_1	b_0	yes	A ← A + b (sign extended)
`ADC`	1	1	0	1	b_3	b_2	b_1	b_0	yes	A ← A + M[b] + C
`SBB`	1	1	1	0	b_3	b_2	b_1	b_0	yes	A ← A + $\overline{\text{M[b]}}$ + C
`NEG`	1	1	1	1	0	0	0	0	no	A ← $\overline{\text{A}}$ + 1
`COM`	1	1	1	1	0	0	0	1	no	A ← $\overline{\text{A}}$
`CLR`	1	1	1	1	0	0	1	0	no	A ← 0
`SET`	1	1	1	1	0	0	1	1	no	A ← 0xFF
`RTL`	1	1	1	1	0	1	0	0	no	A ← ((A<<1)∨(A>>7))∧0xFF
`RTR`	1	1	1	1	0	1	0	1	no	A ← ((A<<7)∨(A>>1))∧0xFF
`LSL`	1	1	1	1	0	1	1	0	no	A ← (A<<1)∧0xFE
`LSR`	1	1	1	1	0	1	1	1	no	A ← (A>>1)∧0x7F
`ASR`	1	1	1	1	1	0	0	0	no	A ← ((A>>1)∧0x7F)∨(A∧0x80))
`TST`	1	1	1	1	1	0	0	1	no	A ← 0x01 if A ≠ 0
`CLC`	1	1	1	1	1	0	1	0	yes	C ← 0
`SEC`	1	1	1	1	1	0	1	1	yes	C ← 1
`TCA`	1	1	1	1	1	1	0	0	no	A ← C
`TVA`	1	1	1	1	1	1	0	1	no	A ← C ⊕ N
`JAL`	1	1	1	1	1	1	1	0	no	PC ← A and A ← PC + 1
`NOP`	1	1	1	1	1	1	1	1	no	

Requirements

1. Your code must compile with the command `make` and produce the executable called `sim`. Among other implications, this means that your source code must include a makefile. (Note that `make` must return 0, so do not include a compiled version of your code.)

2. Your code may not generate errors or warnings when compiled.

3. Your code must be able to accept the SCRAM memory image to simulate as either a file argument or directly on `stdin` (if no argument is supplied).

4. Your code must be able to accept a second argument which is the name of a file to output the final memory contents to. (This only applies to the case where the first argument is the SCRAM memory image to simulate.)

5. Your code may not crash. If you encounter an error, your code must exit cleanly with a non-zero status and an explanatory message. If the SCRAM code is simulated successfully, your code must exit with a zero status.

6. If you want partial credit or CA assistance, your code must be readable. **CA are allowed to refuse to review messy and/or unreadable code!**

Output format

For each instruction, output the address as `0x` followed by a two digit hexadecimal number representing the address, then two spaces, then the opcode (LDA, LDI, *etc*). If the instruction has no argument, add five spaces. Otherwise add a single space followed by `0x` and a two digit hexadecimal number representing the argument. Finally, the line should end with four spaces followed by `ACC=0x` and the accumulator contents (after execution of the instruction) as a two digit hexadecimal number.

Note that if the executed instruction is immediately preceeded by an `EXT` instruction, the argument of the current instruction should reflect this. *e.g.* LDA 0 that is immediately preceeded by EXT 1 should display as `LDA 0x10`. Many instructions are not affected by `EXT`, and those that are, are only affected if the `EXT` instruction is the immediately previously executed instruction. That is, the effect of EXT expires after one instruction.

Exit with a zero status when a `HLT` is encountered. If the input data is more than 256 bytes, ignore and discard anything beyond the 256th byte. If the input data is less than 256 bytes, simulate as normal, but generate an error if the program accesses data or instructions beyond the length of the input data.

Exit status

Your code should exit with a zero status unless one of the following errors occurs:

return 1: The wrong number of arguments were given.

return 2: The input file cannot be opened or cannot be read.

return 3: The output file cannot be opened or cannot be written.

return 4: An attempt was made to execute code beyond the end of the input file.

return 5: An attempt was made to read data beyond the end of the input file.

return 6: An attempt was made to write data beyond the end of the input file.

Examples

The supplied example input files (and corresponding output files) are:

- `ex1_in`: A file containing the solution to problem 1, homework 5 (shift left 4 bits), to be used as the first argument or redirected to `stdin`.
- `ex1_sim`: A file containing the simulator output (from `stdout`).
- `ex1_out`: A file containing the memory contents after completion of the simulation (optional second argument).