CSF 2019 Programming Assignment #4

Overview

For this assignment, you will be writing a program in C or C++ that assembles a SCRAM program from an assembler level description. 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 + 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 `asm`. 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 assembler file 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 SCRAM code to. This only applies to the case where the first argument is the SCRAM assembler file. Otherwise, the SCRAM code should be output to `stdout`.
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!**

Input format

The input consists of lines with the following format:

```
label:  opcode  argument  ; comment
```

The `label` is any string starting with a letter (a-z, A-Z) and composed of letters, numbers, and underscores. The label is optional, but if present, must be followed by a colon.

The `opcode` is any of the ones listed on the previous page, or an assembler directive. It is optional.

The `argument` is only present on those opcodes that need it. It can be either a number (decimal, octal or hexadecimal – see `strtol()`) or a label.

The `comment` is also optional, but if present, must be preceded by a semicolon.

Note that empty lines, as well as lines with only a label or only a comment, or other such combinations are allowed. Also, there can be any number of spaces or tab characters between the label, opcode, argument and comment.

Assembler directives

In addition to the opcodes listed on the first page, the assembler must recognize two directives.

ORG: This directive requires an argument which is the address of where the subsequent opcodes should be assembled to.

DAT: This directive requires an argument which is the data to be placed at the current address.

Output format

The output is a binary file, which does not have a particular format. You can verify that the output is correct by running your disassembler on it and comparing the result to the original SCRAM assembler file supplied to the assembler.

If the input program is too long (requiring more than 256 bytes of SCRAM memory), exit with a non-zero status (see exit status below). If the input program is too short, pad the output to 256 bytes with `HLT` instructions.

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: The program is too large.

return 5: An invalid op-code was encountered (this includes `EXT 0`).

return 6: An invalid argument was encountered (for instance it's too large or it's not an integer).

return 7: An invalid label was encountered (bad label format or duplicate).

return 8: There was an unresolved reference (missing label).

return 9: Memory allocation error.

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) in assembler format.
- `ex1_out`: A file containing the assembled code.