CS 4380 Project 2: 4380 Assembler Mark 1

NOTE: READ THIS DOCUMENT IN ITS ENTIRETY BEFORE YOU BEGIN WORKING

Introduction

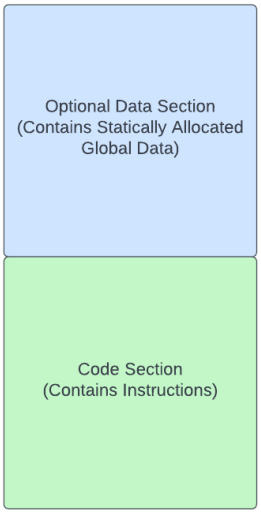
In Project 1 you created an emulator for the 4380 architecture. In that project you experimented with executing bytecode and experienced first-hand how hard it is to read, write, and understand programs represented in this way. An assembler allows programmers to represent programs in assembly language (a human readable representation of a computer's machine code) and then translate those programs into machine code for execution. In this project you will build an assembler that targets the 4380 Mark 1.

4380 Assembler Mark 1

Just as a C++ programmer is "talking" to a compiler as they write code, an assembly programmer is "talking" to the corresponding assembler and must speak its language and meet its requirements. The assembler must in turn be able to "talk" to the target (i.e. computer/emulator/VM)/etc.) – speaking ITS language and meeting its requirements. To BUILD an assembler a programmer must understand both the assembly language that must be accepted as input, as well as the target language (in our case bytecode) that must be generated as output. In Project 1 you learned 4380 bytecode; here we introduce the syntax elements and requirements of 4380 assembly programming. Where appropriate, we provide example code (in monotype text) to demonstrate syntactic elements and rules.

File Requirements

4380 assembly programs must be contained in text files with the '.asm' file extension (e.g. gradebook.asm). Each file is comprised of up to two sections - an optional Data Section (which contains the static and global variable definitions for the program), followed by a Code Section containing the program instructions. The Data Section (if present) must precede the Code Section. Both sections are described in greater detail below.



Line Format

Four valid line types may occur within a 4380 assembly file:

1. Blank lines: Containing only whitespace characters. These will be ignored by the assembler.
2. Comment lines: Containing zero or more whitespace characters followed by a valid comment. These will be ignored by the assembler.
3. Directive lines: Beginning with a label or whitespace character(s) followed by a valid directive.
4. Instruction lines: Beginning with a label or whitespace character(s) followed by a valid instruction.

Note that both directive and instruction lines begin with a label or whitespace, meaning that if a valid line begins with an alphanumeric character, that character is the beginning of a label.

Programmer Comments

4380 assembly files may contain programmer comments throughout the file. Comments begin with the ';' character and extend to the end of a line (much like C-style comments). All text following the ';' symbol shall be ignored by the 4380 Assembler.

```
;This example comment occupies an entire line.
A_LABEL     .BYT  #245 ;This comment occupies part of a line
```

Labels

Labels may be used to mark important locations within both the Data and Code sections of a 4380 assembly program (these sections are discussed below). Label definitions appear as the optional first elements of Directive Lines and Instruction Lines (they must not appear on a line by themselves). They must be left justified (no preceding whitespace characters). Label names must begin with an alphanumeric character and may contain alphanumeric characters, underscores, and the '$' character.

For example:

```
labelName9

NUM_APPLES

function$
```

...are all valid labels, while:

```
_bad
```

...is invalid.

Note: Use of the '$' character in manually written 4380 assembly labels is discouraged, as limiting this character to compiler generated labels can prevent naming collisions between programmer and compiler generated labels/identifiers.

Label names can be used as operands for instructions in the Code Section of a 4380 assembly.

Literal Data Representation

Both directives and instructions may make use of literal data within a 4380 assembly file. The supported numeric and character literals are as follows:

- Numeric: Signed 8 or 32 bit integer value represented as a base 10 number preceded by the '#' character. Examples: #45, #2, #-67
- Character: Apostrophe delineated ascii character (including escape sequences for special characters), or a character's equivalent ascii code expressed as a decimal value. Examples: 't', 'Y', '6', '#', '\n', #83 (83 is the ascii code for the character 'S'). The following escape sequences must be supported by the 4380 assembler:
    - \t (tab)
    - \\ (backslash)
    - \n (newline)
    - \' (single quote)
    - \" (double quote)
    - \r (carriage return)
    - \b (backspace)

Data Section and Directives

The Data Section of a 4380 assembly contains assembler directives that define static and global variables. The Data Section implicitly begins with the first line of an assembly file and continues until the first line containing an assembly instruction.

Directives can only be used within the Data Section. The presence of one of these directives in the Code Section shall result in an assembler error. All directives begin with a period, making them easy to identify when parsing a 4380 assembly file. The supported directives for the Mark 1 assembler are as follows:

Directive	Operand	Examples	Behavior
.INT	Optional signed decimal value in the range -2147483648 to 2147483647 inclusive.	.INT #45 .INT #-12 .INT #2147483647	Allocates memory in place for a 4 byte integer and initializes it with the optional operand value. If no operand is provided the value is initialized to 0.
.BYT	Optional unsigned decimal value in the range 0 to 255 inclusive, OR apostrophe delineated ascii character (including escape sequences for special characters)	.BYT #45 .BYT 'a' .BYT '\n'	Allocates 1 byte of memory in place and initializes it with the optional decimal value or ascii code. If no operand is provided the value is initialized to 0.

Note that the assembler must treat directives as case INsensitive. For example ".int" and ".INT" must be treated the same by the assembler. The Data Section of an assembly may contain as many directives as desired (within the limitations of the emulator's memory) but shall be limited to one directive per line. Each directive may optionally be preceded by a label, which in this context serves as a variable or array name.

```
NUM_APPLES        .INT  #42; An integer variable initialized to 42
CURRENT_MONTH     .BYT ; An unsigned int variable initialized to 0
A_CHAR            .BYT  'A'; A char variable initialized to the
                          ; character 'A'
A_CHAR_ARRAY      .BYT  'H'; The first character of an array of
                          ; characters
                  .BYT  'e'
                  .BYT  'l'
                  .BYT  'l'
                  .BYT  'o'
```

Directive Format

Considering the preceding discussion the format of a 4380 directive can be summarized as follows:

```
[optional_label] <.directive> [optional_value] [optional_comment]
```

Note the presence of required whitespace between the individual components of a directive (including between the optional comment and optional value). This whitespace must be comprised of space and/or tab characters.

Code Section

The Code Section of a 4380 assembly file implicitly begins with the first line containing an instruction.

As with the Data Section and directives, the Code Section may contain as many instructions as are desired (within the limitations of the emulator's memory), but shall be limited to one instruction per line.

NOTE: Assembly directives and instructions cannot be interleaved. Directives are limited to the Data Section, and instructions must reside in the Code Section of a file.

Instruction Format

All 4380 Mark 1 instructions shall be supported by the 4380 Mark 1 assembler. Instructions must be formatted as follows:

```
[optional_label] <operator> <operand_list> [optional_comment]
```

As with the format for directives, note the required whitespace (which can be comprised of spaces and/or tabs) separating the individual components of an instruction.

An instruction's `operator` is limited to one of the three or four-character representations of valid 4380 operators as listed in the instruction tables utilized in this course. For convenience we provide these tables again below. The assembler treats operators as case INsensitive. For example "aDd" and "ADD" should both be recognized/accepted by the assembler.

The `operand_list` is a comma separated list of the necessary operands for the operator being invoked. Valid operands include:

- Registers - indicated via the two or three-character register names listed in the register tables utilized in this course (again provided for convenience below). Note, the assembler shall treat register names as case INsensitive. R13 is considered the same as r13.
- Numeric constants - represented as literal values (e.g. #45).
- Labels - as defined in the assembly file and later resolved to addresses by the assembler. Labels are to be treated as case SENSITIVE by the assembler.

Instructions and Register Tables

Note: Several operands are marked as DC, meaning "Don't Care". This means that while the operand is required - its specific value doesn't matter to execution of the instruction.

Jump Instructions:

Value	Operator	Operand 1	Operand 2	Operand 3	Immediate Value	Description
1	JMP	DC*	DC*	DC*	Address**	Jump to Address

*This operand shall be omitted in 4380 assembly instructions but is required in 4380 bytecode.
**When represented in an assembly file the Address shall be represented as a label. This label must be resolved to a 32 bit unsigned integer value in bytecode.

Move Instructions:

Value	Operator	Operand 1	Operand 2	Operand 3	Immediate Value	Description
7	MOV	RD	RS	DC*	DC*	Move contents of RS to RD
8	MOVI	RD	DC*	DC*	Imm***	Move Imm value into RD
9	LDA	RD	DC*	DC*	Address**	Load address into RD
10	STR	RS	DC*	DC*	Address**	Store integer in RS at Address
11	LDR	RD	DC*	DC*	Address**	Load integer at Address to RD
12	STB	RS	DC*	DC*	Address**	Store least significant byte in RS at Address
13	LDB	RD	DC*	DC*	Address**	Load byte at Address to RD

*This operand shall be omitted in 4380 assembly instructions but is required in 4380 bytecode.
**When represented in an assembly file the Address shall be represented as a label. This label must be resolved to a 32 bit unsigned integer value in bytecode.

*** When represented in an assembly file the Imm value shall be represented as a numeric literal.

Arithmetic Operations:

Value	Operator	Operand 1	Operand 2	Operand 3	Immediate Value	Description
18	ADD	RD	RS1	RS2	DC*	Add RS1 to RS2, store result in RD
19	ADDI	RD	RS1	DC*	Imm**	Add Imm to RS1, store result in RD
20	SUB	RD	RS1	RS2	DC*	Subtract RS2 from RS1, store result in RD
21	SUBI	RD	RS1	DC*	Imm**	Subtract Imm* from RS1, store result in RD
22	MUL	RD	RS1	RS2	DC*	Multiply RS1 by RS2, store result in RD
23	MULI	RD	RS1	DC*	Imm**	Multiply RS1 by Imm, store the result in RD
24	DIV	RD	RS1	RS2	DC*	Perform unsigned integer division RS1/ RS2. Store quotient in RD. Division by zero shall result in an emulator error
25	SDIV	RD	RS1	RS2	DC*	Store result of signed division RS1 / RS2 in RD. Division by zero shall result in an emulator error
26	DIVI	RD	RS1	DC*	Imm**	Divide RS1 by Imm (signed), result in RD. Division by zero shall result in an emulator error

*This operand shall be omitted in 4380 assembly instructions but is required in 4380 bytecode.

** When represented in an assembly file the Imm value shall be represented as a numeric literal.

Traps/Interrupts:

Value	Operator	Operand 1**	Operand 2**	Operand 3**	Immediate Value	Description
31	TRP	DC**	DC**	DC**	#0	Executes the STOP/Exit routine
31	TRP	DC**	DC**	DC**	#1	Write int in R3 to stdout (console)
31	TRP	DC**	DC**	DC**	#2	Read an integer into R3 from stdin
31	TRP	DC**	DC**	DC**	#3	Write char in R3 to stdout
31	TRP	DC**	DC**	DC**	#4	Read a char into R3 from stdin
31	TRP	DC**	DC**	DC**	#98	Print all register contents to stdout

**This operand shall be omitted in 4380 assembly instructions but is required in 4380 bytecode.

Instruction Size: Fixed at 8 Bytes (64 bits)

1 Byte Operation	1 Byte Reg Operand 1	1 Byte Reg Operand 2	1 Byte Reg Operand 3	4 Byte Immediate Value

Registers (22):

Name	Encoding	Description (all registers are 32 bits in size)
R0 – R15	0 - 15	General purpose registers, **initialized to 0 at startup**
PC	16	Program Counter, **initialized to the value of first instruction in memory**
SL	17	Stack Lower limit (lowest legal address available to the stack)
SB	18	Stack Bottom (highest address)
SP	19	Stack Pointer (latest allocated byte on the stack, grows downward)
FP	20	Frame Pointer (points to the first word beneath the return address)
HP	21	Heap Pointer (initially set to SL, grows upward))

Requirements

As mentioned above, for this project you must build an assembler that accepts as input a valid 4380 assembly file and outputs valid 4380 bytecode. The project requirements are as follows:

Req1) The assembler must reside in a PRIVATE GitHub repository called cs4380-emulator-assembler. This shall be the same repository that was utilized for Project 1.

Req2) Project 2 work must be completed in a separate branch called project-2. When all requirements are met, the branch should be merged into main (BUT NOT DELETED).

Req3) The assembler must be written in the **Python** programming language.

Req4) The main module/file for the assembler program shall be called **asm4380.py**

Req5) The asm4380.py module, and any supporting modules shall reside in a folder called "assembler" which shall reside in the root/base folder of the repository/project. **Note:** This means the assembler .py files do NOT reside in the src folder.

Req6) The assembler must accept a single command line argument which shall be the name of a text file containing 4380 assembly code as described up to this point in the course. The input filename must end with the '.asm' suffix/file extension. Upon beginning execution the assembler shall verify that a filename is present and valid before continuing. If either of these conditions is not met the program must print the message: "USAGE: python3 asm4380.py inputFile.asm" and exit with a return value of 1. (i.e. exit(1))

Req7) The assembler must output a single file containing the bytecode representation of the program contained in the input file specified in Req5. The name of this file shall match the name of the input file with the ".asm" suffix replaced with ".bin". As an example, if the user invoked the assembler with the command: "`python3 asm4380.py gradebook.asm`" the resulting output file would be called "gradebook.bin". When an assembly file is successfully

assembled and the resulting .bin file has been generated; the assembler shall exit with a return value of 0.

Req8) The assembler must enforce the syntactic rules of 4380 assembly code as explained in the preceding sections of this document. If a violation of a rule is encountered the assembler shall print the message "Assembler error encountered on line NUMBER!" (where NUMBER is the line number where the error was encountered in the input file), and exit with a return value of 2.

Req9) In the event that a missing label error occurs during assembly (which is not detectable until the end of the input file is reached), the line NUMBER referenced in Req8 shall be the last line of the input file.

What to Turn In:

When you have completed the project you will need to submit the URL for your repository on the assignment page in Canvas. Your repository should be private **but shared with your course grader and/or instructor.** Instructions for granting access to your repo for the course grader will be provided by your instructor.

Note: If you are invoking the Non-Late policy please insert a note at the top of your README file that states “INVOKING THE NON_LATE POLICY” followed immediately by your work log. (The rest of your README should still contain relevant project details including those discussed in previous projects).