CO Assembler Documentation

Hamzah Akhtar (2018051) | Shashwat Aggarwal (2018097)

Description

The program assembler.py takes an input file containing the assembly code to be assembled. The assembly code is converted into machine code and saves it in MachineCode.txt. The intermediate tables such as Opcode table, Symbol table and Literal table are also created and saved in their respective files.

How to run

To run the assembler you have to save the input file in the same directory as the python script. For example, if the input file is “input.txt” then write the following command:

```
$ python3 assembler.py input.txt
```

How it Works

The input file is read line by line during the first pass. We then check the number of words in the line and then cross-check the opcodes with the opcodes that we have in our list. If it is a valid opcode we add it to our opcode table. If the opcode is not valid then we add the corresponding errors to the error stack (the opcodes for our program are listed at the end of this document).

Then we check the operands. The operands can be either a memory location, literal or symbol.
If the operand is a literal it must of the format ‘=<numeric>’ else it will give an error that is saved in the error stack.

Otherwise, if it is a symbol then we save it in the symbol table but keep its offset as “NONE” until we find its definition. Once we find the definition of that symbol we set the offset for that symbol as the current location counter. The location counter is incremented at each valid line.

If the error stack is empty we print the opcode table, literal table and symbol table and then call the second pass function to proceed with the second pass. Else the error stack is printed in a tabular form.

Then in the second pass, we open the input file again and read it again and convert the line into machine code. We have set each instruction location to 8 bits and each operand to 8 bits as well. Once the machine code is generated it is saved in the file “MachineCode.txt”.

Errors Handled

- The opcode specified is not a valid opcode
- ‘START’ statement is missing
- ‘START’ is not the first line
- Multiple ‘START’ statements
- ‘END’ statement is missing
- ‘END’ written before ‘START’
- Special characters in operands
- Invalid memory location in ‘START’
- Opcode used as an operand
- The operand is not present
- A symbol has been declared/used but not defined
- Symbol has been defined more than once
- An opcode has been supplied with a lot of operands

- Insufficient number of operands in the statements
- Literals are not numeric and not written in the proper format
- Invalid input file
- Code written after ‘END’ is assembled but a warning is shown

Symbol Table

We use an array of strings to store all the symbols and their offsets which are then used to convert assembly code to its equivalent assembly code by using its offsets and converting
it to binary

Opcode Table

In the opcode table, the first column has the opcodes, the second column has the corresponding instruction in a statement, and the third column has the offset of the instructions.

Machine Code Table

In the machine code output, we have 3 columns out of which the first column is of instruction’s location in the program counters memory, the second column is for the opcode and the third column has the offsets of literals or symbols in the memory location converted to binary.

Assembly code	Opcode	Meaning
CLA	0000	Clear Accumulator
LAC	0001	Load into accumulator from address
SAC	0010	Store accumulator contents into the address
ADD	0011	Add address contents to the accumulator contents
SUB	0100	Subtract address contents from accumulator contents
BRZ	0101	Branch to address if accumulator contains zero
BRN	0110	Branch to address if accumulator contains negative value
BRP	0111	Branch to address if accumulator contains positive value
INP	1000	Read from the terminal and put in the address
DSP	1001	Display value in an address on terminal
MUL	1010	Multiply accumulator and address contents
DIV	1011	Divide accumulator contents by address content. The quotient in R1 and remainder in R2
STP	1100	Stop execution
DW	1101	Variable Declaration