Project 1 Documentation

What is the Project about?

This is a 2-pass Assembler the capacity to handle OpCodes mentioned below and variables. Just like any standard 2-pass Assembler, it scans through the code twice, 1st time to find out how long the machine code will be and storing the OpCodes in the Intermediate Table and hence the address of labels and storing them in the Symbol Table. Then the actual Machine Code is generated using the Intermediate Table and Symbol Table.

The assembler handles the following Assembly OpCodes:

CLA : Clears the accumulator

LAC <Memory Address> : Load into accumulator from address

SAC <Memory Address> : Store accumulator contents into address

ADD <Memory Address> : Add address contents to accumulator contents

SUB <Memory Address> : Subtract address contents from accumulator contents

BRZ <Memory Address> : Branch to address if accumulator contains zero

BRN <Memory Address> : Branch to address if accumulator contains negative value

BRP <Memory Address> : Branch to address if accumulator contains positive value

INP <Memory Address> : Read from terminal and put in address

DSP <Memory Address> : Display value in address on terminal

MUL <Memory Address> : Multiply accumulator and address contents

DIV <Memory Address> : Divide accumulator contents by address content. Quotient in register R1 corresponding to address 00000000 and remainder in register R2 corresponding to address 00000001.

STP : Stop execution

Syntax for the above mentioned OpCodes:

1) Start your code with “START <Address(optional)>”.
2) A line should have only 1 OpCode.
3) OpCodes “CLA” and “STP” shouldn’t have any address provided to them as operand while all the other operands must have a valid address.
4) To define a label follow this syntax ➔ <LabelName> : <OpCode> <Address(if required)>.
5) To add a comment type ";". Everything ahead of it is treated as a comment.

The output provides us with the corresponding machine code like so:

In the 1st pass we make the Intermediate Table and Symbol Table by checking the following:

1) If the inputted OpCode is valid. Any invalid instruction return "Invalid Opcode!".
2) If the instruction requires address, and hence the length of the instruction. If the instruction is too long, it returns "Too long instructions".
3) If the instruction contains Label. We hence add the Label to the Symbol Table with the current location counter value. We also add the label with the OpCode and the operand(if required) to Intermediate Table.
4) We ignore the comments which are identified using ";".
5) As the pass ends, we assign the incremental location counter values to the addresses of variables.

In the 2nd pass we traverse the Intermediate Table and construct the corresponding Machine Code.

Errors handled

1) Use of Invalid OpCodes.
2) Missing "STP" OpCode.
3) Missing Operand for OpCodes that require one.
4) Extra Operand provided for OpCodes that don't require it.
5) Use of undefined labels.
6) Defining a label more than once.
7) Invalid syntax for label definition.
8) START must be present.
9) START can’t have more than 1 operand.
10) START must have a valid address/ operand.
11) Handles empty/blank lines.