CO Assembler Project

**General information for using two-pass assembler**

Files used:- 4 files are used (INPUT, SYMBOL TABLE, LABEL TABLE, OUTPUT) & 2 registers bases.

1.INPUT file:- the file where assembly code is saved

2.SYMBOL TABLE:- The file where Symbols/operands and their location will be stored

3.LABEL TABLE: The file where Label and their location will be stored

4.OUTPUT:-MACHINE CODE WILL BE STORED

**Registers bases**:

1.For storing locations of constants

2.storing operands,labels etc.

Output format

1.OpCodes will be used as given in the classroom.

2. All operands will be referred to using the names as used in the input code one can search for their addresses in the Symbol table file

3.however the labels & constants will be referred to using their location or memory address.

4. Comments and start statement will not be converted to machine code

Expected Input format:

1.START statement should be mentioned at the first line of comment

2.NO empty lines in the code are allowed

for defining labels, one needs to put ":" in the end for the name of Label

example "LABEL NAME":

3. For writing comments : one needs to use a new line with "com" as the first three letter without space and a space is required after . Example

"com " after which comment should start

 4. Valid operand codes are taken from the file provided on the classroom

5.code should have an STP statement in the last line of code

Error & warning reporting(first pass)

1.No START STATEMENT(code reports error and exit)

2.INVALID OPCODE (code reports error and exit)

3. incorrect number for operands for given opcode(code reports error and exit)

4. Operand not defined ( code reports error and exit)

5. Label not defined (code reports error and exit)

6. operand defined but not used(Code reports a warning and continuous)

7.label defined but not used/called(code reports warning and continuous)

Second pass

The respected code if not errors found in the first pass is processed into machine language code

Flow of code

First pass

1.WE check the first line of code and see whether the Start is given or not.

2.run a loop till the last line of code form the second line

3.check if the line is a comment or not if a comment then skips to the next line

4. Check whether the line has a label or not (if not continue else if Label used save in Label used array else save in Label declared array)

5. compare whether the operand code is valid or not using the protected 2d array consisting of the name of valid opcode and required operands.IF valid continue else report error and exit

6.Count the number of operands required for a specific opcode, report an error if not equal and else continue

7.Used & declared variables and labels are stores in diff arrays and are compared if not equal warning or error is displayed

8.check the last line if STP is not found report an error

**After we have completed the first pass without any errors we move to the second pass.**

**Second pass:**

We have used two files ; one for reading i.e Assembler file and the other for writing in which we create the object code. Before we start the reading, we create two 2D arrays; one with assembly opcodes and no. Of operands while the other with assembly opcodes and their corresponding machine opcodes. We also have a memoryadd counter for keeping track of the memory address.

File reading: We have used try-catch to check if if the we have operand as variable or as constant and write correspondingly.

We encounter different assembly opcodes during our 2nd pass:

1. CLA and STP: Both of them take none operands and are written directly into the file with their corresponding machine opcodes.
2. LAC,SAC,ADD,SUB,DSP,MUL: All of them take a single operand. If operand is variable we take it as register while constants are converted to binary.

1. DIV: It takes three operands. For this we have taken a for loop till the end of the assembly line(basically our string array).
2. BRZ,BRN,BRP: The are branch statements which take us to the label and perform the specified operation. We had already stored the the address of the labels and no. Of labels in the first pass.

If we do not encounter any of them then there must be a “label”.With the help of oparray1 and oparray2 defined in the starting of the second pass comes into play.

We have finally converted our assembly code to object code(machine code).