Labels:

1. The Label name can be followed by any number of spaces but must always be succeeded by a ‘:’.
2. Eg:

Hello :

Comments:

1. The comments are followed by ‘;’.
2. An Entire statement can entirely be a comment

Eg: ;Hello this is a comment

1. A comment can also be placed after a normal instruction. In this case the instruction part will be the only one to get evaluated.

Eg: ADD A ;This comment is following a normal instruction

1. The comment will not get translated during the assembly process.

Opcodes:

1. Opcodes are created completely according to the list offered in the project specification.
2. Each opcode will only require 1 operand or 0 operand.
3. If there are no errors and declaration problems then the program will terminate after ‘STP’ opcode.
4. There is no opcode for start used in the assembler and the virtual addressing will always start from 0.

Operands:

1. Operands will be of 4 types:
   1. Label name:

* When the Label name is specified they the value of location counter at that time will fill the associated offset address column.
* Size and value column will be empty.
* Type column will be filled with “Label”.
* Range is 0 to 110
  1. Variables associated to a memory location(VariableDW):
* The memory they are associated with should be specified in the end.
* They are referred to by the type ‘DW’.
* In this case the associated memory will be used to fill the offset address column in the Symbol Table.
* Type will be “VariableDW”.
* Value column will be empty.
* Size column will be “7 bits”.
* Associated address will be given in the decimal form just as a number:  
  Eg: A DW 5
* Range is 111 to 127
  1. Variables associated with a constant:
* The value they are associated with should be specified in the end.
* They are referred to by the type ‘DC’.
* In this case the associated value will be used to fill the value column in the Symbol Table.
* Type will be “VariableDC”.
* Range is 0 to 127.
* Offset Address column will be empty.
* Size column will be “7 bits”.
* Associated vakue will be given in the decimal form just as a number:  
  Eg: A DC 5
* When used with branching type opcodes, their values represent an address and for others it represents a constant value.
  1. Literal constants:
* They will be the constants that needs to be supplied with an opcode.
* Suppose ‘x’ is a constant value that needs to be used then it should be supplied in the following format:

‘=x’

x in this case will be stored in the literal table with the size value as “7 bits”

* They should always be in the range 0 to 127. Otherwise a fatal RangeError will be given resulting in termination of the program.
* The constant will always be considered a memory address. To use a constant value use them as in the (c) category.

1. The code is made to handle even multiple operands which can be separated by a space or a comma(Even though there is no need in Accumulator architecture).For unnecessary trailing commas in such cases the code will throw an error. Usually the code will be operated with a single or zero operands as just translation is taking place you can provide even more operands.
2. Symbols:(a),(b) and (c) in the above categorization.
3. Operand names should not have any space or comma or ‘=’ in them, anything other than that can be used in creating the operands.
4. There can be any number of spaces between any elements in the code.

General Instruction:

(THINGS IN THE SQUARE BRACKETS CAN BE OMITTED)

1. Format:

* Instruction:

[LabelName :] Opcode Operands:based on the opcode(can be present or omitted) [;comment]

* Definition:

Symbol Name Symbol type:DC or DW DW:Associated address ,DC:Associated value

1. The individual parts in the above formats can be separated by any number of spaces.
2. Eg: Instruction:
3. ADD A ;HELLO
4. STP
5. BRP Loop (LABEL NAME)
6. SUB ‘=3’

Definition:

1. A DW 5
2. B DC 6

4. Any number entered should be considered in the decimal format.

5. Final file will only be give the binary code

6. There can be any number of spaces between any 2 lines.

7. Only spaces should be used for separation and not any tabs.

8. Label can be omitted by not supplying ‘:’ and comments by not supplying ‘;’.

9. There can be any order for definitions and their usage, but required warnings will be issued.

Location counter and Bit format used: The location counter will indicate the word number and the used format is:

Addressing\_mode(0=Direct and 1= Indirect(always 0),(1 bit)) Opcode(4 bits) Operand(7 bits)

Tables and Files created:

1. All the files created and required must be within the same folder.
2. All the tables will be shown at the during the end of the program at least once. For tables containing entries, multiple windows of tables indicating their filling one by one will be displayed at the end.
3. Input file name: INPUT.txt
4. Temporary file name: TEMP.txt
5. Final file name: FINAL.txt
6. Tables:

* Symbol Table: Symbol,Type,Offset Address,Value,Size
* Opcode Table:Opcode,Type,NumOperands,Corresponding Code, length
* Literal Table: Value, Size
* Error Table: Error Statement,Type Of Error, Offset Address

Error Reporting:

WARNINGS AND FATAL ERRORS WILL BE GIVEN ON THE SPOT WHEN FOUND BUT THE OTHER THINGS WILL BE GIVEN BEFORE THE END OF 1ST PASS.

1. LabelNotSuppliedError: Label’s distinguishing symbol ‘: ‘ is being used but still it nothing is given before that.
2. WrongFormatError:

* If the literal is given without any trailing ‘ character.Eg: =3 instead of ‘=3’.
* The program will give a parsing (NumberFormatException) and terminate by itself if there is no integral value in the required format.Eg: ‘=three’.
* If no ‘=’ is used in the literal then also the WrongFormatError will be given.Eg: ’3’.
* Corresponding error along with the required details will be provided at the end of 1st pass.

1. RedefinitionError: If the same Label is being defined at different places.
2. FatalError:

* If the passed literal is not lying with the allowable range of 0 to 127.This will be followed by program termination.
* If the program length(0-110), VariableDC value(0 to 127) and VariableDW(111-127) lies outside their corresponding ranges. There is no explicit test for label Range.
* In the 2nd pass, If the taken operand cannot be represented within 7 bits.
* In the 2nd pass,If the opcode requires a Label but is supplied with a VariableDW or vice versa.

1. If during the 2nd pass, there will be some error during reading the file, the program will show the stack trace and continue with the execution but the final file may not be the expected one. All the tables in their final states will also be given.
2. In the 2nd pass if there is some symbol found which is not in the symbol table then a Fatal Error will be given and the program will terminate.
3. If the number of operands required and supplied for a particular opcode differs.
4. If at the end of 1st pass, There are some:

Lables or VariableDW with no offset address

VariableDC with no value

Then the program will output corresponding errors and locations and then terminate the 1st pass.

1. If there is some symbol defined at the end which is not being used in the program then a corresponding warning will be issued but the program will function as normal with the corresponding symbol added to the symbol table.
2. Errors will be given in the following way:

Error->show them->terminate. No Error->Nodefinition errors->show them->terminate.

No Error->No Nodefinition errors->First pass complete.

**UNLESS SPECIFIED ALL THINGS ARE HAPPENING IN THE 1ST PASS.**

ASSUMPTIONS:

1. If more than one operands are used then the program’s behavior will be unknown for which developer will not take responsibility.
2. If invalid opcodes or incorrect opcode names are used then the program’s behavior will be unknown for which developer will not take responsibility.
3. Size of each statement(instructions or definitions) is assumed to be 1 word.
4. location\_counter will represent the word number(which will start from 0).
5. Definitions can be anywhere with respect to their use in the code(Even before their use).
6. All the instructions and definitions should rigorously follow the provided format.
7. Label definition must be followed by an instruction.Eg:

Loop:

ADD A

Is not correct.

But should be written as:

Loop: ADD A

1. If any DC or DW is defined again and again then an overwriting will take place and the final value will be considered.

Eg: A DW 1 A,Offset Address=1

A DW 2 A,Offset Address=2

Final Offset Address for A=2

B DC 1 B,Value=1

B DC 2 B,Value=2

Final Value for B=2

1. Program will can have an entire length(instructions +definitions(not including comments))=111

And the remaining 17 words will be reserved for ‘VariableDW’ type of variables.

As we have used an address association approach the final ranges for addresses are:

Program: 0 to 110

VariableDW: 111 to 127

Label: 0 to 110

VariableDC: 0 to 127

Literals: 0 to 127

(THESE NUMBERS INDICATE WORD INDEX).

1. All the restrictions to prevent memory clashes has been put but for the literals programmer discretion is advised to ensure the correct and free addresses according to their requirements.
2. There are some instances where the program will just handle the given Exception by just printing the stack trace for that exception.
3. Do not define different variables for the same address.
4. In case of an error the final files and the tables produced may not be the correct ones as the program has terminated in between.
5. In case of opcodes with no operands, the final machine code will contain “0000000” as the operands.
6. The input file will be named “INPUT.txt”,temporary file for input to 2nd pass “TEMP.txt” and final machine code file “FINAL.txt”.

**RUN THE ASSEMBLER FILE FROM THE COMMAND PROMPT BY REACHING THE DIRECTORY**