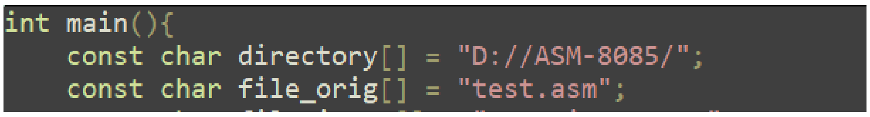
8085 Assembler User Guide

1) How to use the assembler?

Step-1: Write the assembly code in any text editor and store the file in known path.

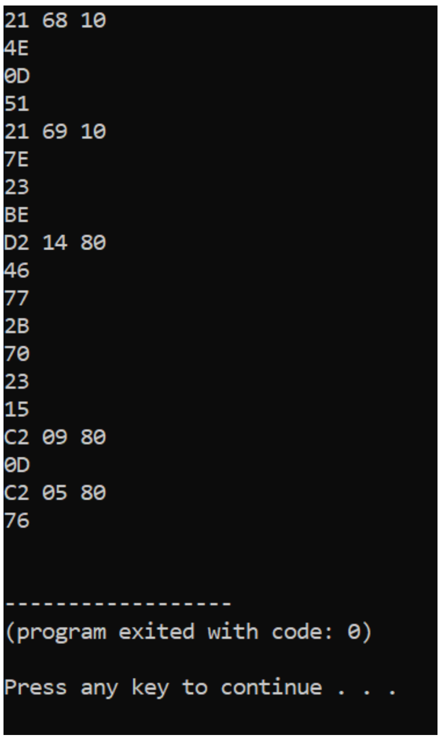
Step-2: Copy the path and file name of the assembly code and paste in the 'int main ()' section of the source code as values for the constant strings 'directory' and 'file_orig'. For ex: if the path is 'D://ASM-8085/' and the file name is 'test.asm' then the resulting code section is given below:



```
int main(){
    const char directory[] = "D://ASM-8085/";
    const char file_orig[] = "test.asm";
```

The 2-pass assembler generates two files in the same directory during the assembly process. They are of the format '<name>**_inter.**<file_type>' and '<name>**_final.** <file_type>'. For ex: if the file name is 'test.asm' then the two files generated by the assembler are 'test_inter.asm' and 'test_final.asm'. The latter file contains the final machine code for the assembly language code.

Step-3: Compile the source code and execute it. The machine code for the assembly language program will be printed on the screen and will be stored in the '<name>_final. <file_type>' file. If there happens to be any errors during the assembly process, two possible situations can arise. If the error is detected in pass-1 of the assembler, only the error message is displayed. If the error is detected in pass-2 of the assembler, the machine code up to the line of first error will be printed, after which the error message is displayed. Below are 3 examples illustrating the different cases.

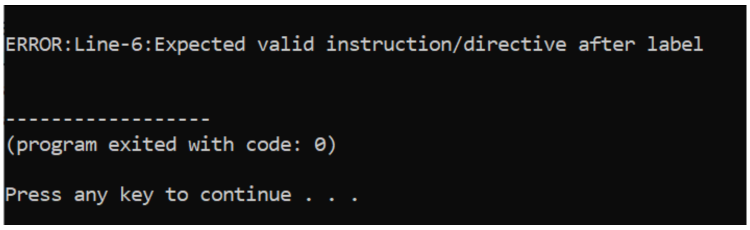


```
21 68 10
4E
0D
51
21 69 10
7E
23
BE
D2 14 80
46
77
2B
70
23
15
C2 09 80
0D
C2 05 80
76


------------------
(program exited with code: 0)

Press any key to continue . . .
```

No – Error

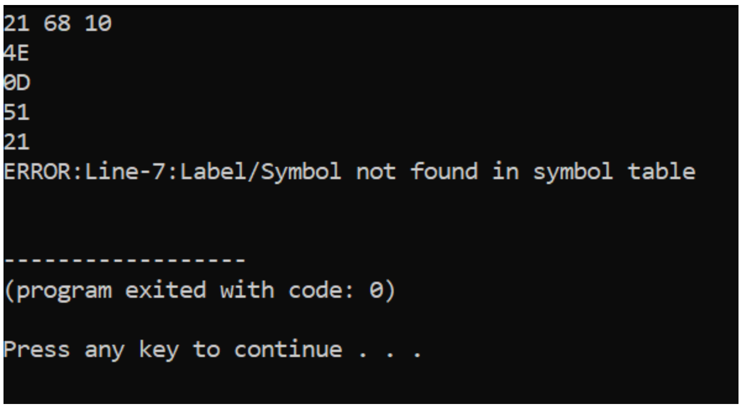


```
ERROR:Line-6:Expected valid instruction/directive after label


------------------
(program exited with code: 0)

Press any key to continue . . .
```

Error detected in Pass-1


```
21 68 10
4E
0D
51
21
ERROR:Line-7:Label/Symbol not found in symbol table


-----------------
(program exited with code: 0)

Press any key to continue . . .
```

Error detected in Pass-2

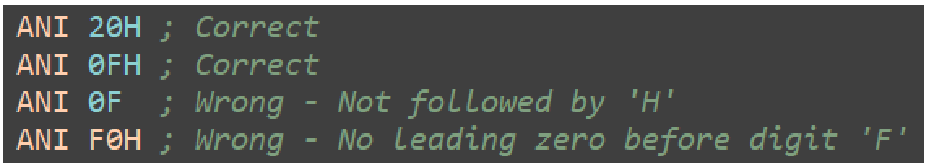
In summary, the steps are:

Write assembly program => Paste directory and name of file in 'int main ()' => save, compile and execute the source code

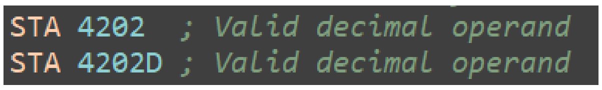
2) <u>Features of the assembler</u>

2.1) Supported number formats

The assembler supports only hexadecimal and decimal number formats. Any numeric operand should to be entered after conversion to any of the two formats. Hexadecimal numbers need to be ended with the letter 'H'. A very important point to keep in mind when dealing with hexadecimal format is that if the number starts with an alphabet (A, B, C, D, E, F), at least one leading zero must be added before the first digit of the number. This is to ensure that the number is not misinterpreted as a symbol/label. The below image illustrates this point.

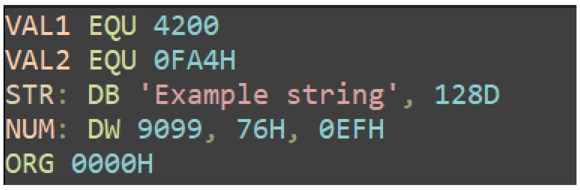

```
ANI 20H ; Correct
ANI 0FH ; Correct
ANI 0F  ; Wrong - Not followed by 'H'
ANI F0H ; Wrong - No leading zero before digit 'F'
```

Decimal numbers can be entered in two ways. The number can be typed without any ending character or the number can be followed by the character 'D'.


```
STA 4202  ; Valid decimal operand
STA 4202D ; Valid decimal operand
```

2.2) Supported assembler directives

The assembler recognizes a total of 6 directives only. They are DB, DS, DW, END, EQU, ORG. A detailed discussion on the functionality and usage of these directives can be found in chapter 4 of the 8080/8085 assembly language programming manual released by Intel. The assembler doesn't support the evaluation of mathematical and logical expressions. So, expressions of such sort cannot be used as operands to directives or to instructions. Only absolute numbers and strings enclosed in single quotes are allowed as arguments to directives. The following image shows few examples of the valid operands for assembler directives.



```
VAL1 EQU 4200
VAL2 EQU 0FA4H
STR: DB 'Example string', 128D
NUM: DW 9099, 76H, 0EFH
ORG 0000H
```

2.3) Supported instructions

The assembler can generate the op-code for all the 247 instructions in the 8085-instruction set. In order to understand the functionality and usage of each instruction refer chapter 3 of the 8080/8085 assembly language programming manual released by Intel. In order to find the op-code for each instruction, refer the last 3 pages of the Intel 8085AH datasheet. The source code contains two global constant string arrays by the name 'INSTRUCTION' and 'OPCODE' that contain all the 247 assembly instructions and their respective op-codes.

2.4) Error messages

The error messages of the assembler are helpful to spot errors and debug the program. If there happens to be an error, the assembler can point the line of error and the type of error on the screen. The error messages are not very specific to the extent of pointing the token/character of error. So, find below a discussion of all the error messages and what they can imply.

1) ***'Invalid Label'***: A label is a variable that usually contains a memory address. It is mainly used by the jump and call instructions to execute loops and subroutines. The syntax for a label is:

a) Start with an alphabet

b) Followed by alphanumeric characters only

c) Any label must be ended with a colon ':'.

If any of the above rules are not met, the 'Invalid Label' error is raised.

2) ***'Invalid Symbol'***: A symbol is usually used to store a value. In this assembler, a symbol can only be assigned by the 'EQU' directive. The syntax for a symbol is:

a) Start with an alphabet

b) Followed by alphanumeric characters only

If any of the above rules are not met, the 'Invalid Symbol' error is raised

3) '***Invalid instruction/directive***': If the instruction doesn't belong to the 8085-instruction set and the 6 directives mentioned in section 2.2, this error is raised. More specifically, the error points to the mnemonic section of the instruction.

4) '***Expected valid instruction/directive after label***': Every label must be followed by a valid instruction/directive. So, any instruction not belonging to the 8085-instruction set and any directive other than the 6 mention in section 2.2 will result in this error. This also means that the space after a label cannot be left empty. This rule is not applicable when the label is being used as an operand since the operand is the last token in any line.

5) ‘***Expected valid directive after symbol***’: Every symbol must be followed by a valid directive (In case of this assembler, the only possible directive after a symbol is ‘EQU’). This also means that the space after a symbol cannot be left empty. This rule is not applicable when the symbol is being used as an operand since the operand is the last token in any line.

6) ‘***Invalid operands after instruction***’: This error is purely dependent on what follows the mnemonic of an instruction. Every instruction expects certain type of operands. Some instructions require registers. Others require registers and numbers. Some instructions require no operands at all. A summary of the valid operand for every instruction and their respective op-codes can be found in the first 6 pages of the appendix of Microelectronics and Microprocessor book. If these operand types are not used for their appropriate instructions, this error is raised.

7) ‘***Invalid digit in numeric operand***’: If the numeric operand happens to be in hexadecimal format (ended by ‘H’), valid digits are 0-9 and A-F. If the numeric operand is in decimal format (ended by nothing or ‘D’), valid digits are 0-9. Any other characters used in numeric operands causes this error.

8) ‘***Numeric operand greater than instruction word size***’: Some instructions require one-byte operands (0-255) while others require two-byte operands (0-65535). If the entered number exceeds these ranges, this error is raised.

9) ‘***String operand not terminated***’: String operands can be used by the directives DB and DW. A string needs to be typed between two single quotes. If the string operand in not properly terminated by single quotes, this error is raised.

10) ‘***Numeric operand greater than directive's limit***’: The DB and DW directive can take numeric operands in the range 0-255 and 0-65535 respectively. If these ranges are exceeded, this error is raised.

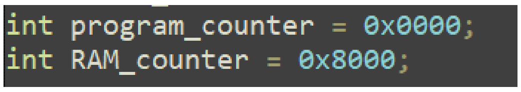
11) ‘***Invalid operands after directive***’: The 6 directives supported by the assembler require different types of operands. They can be referred in chapter 4 of the 8080/8085 assembly language programming manual. If these types are not used properly for the respective directives, this error is raised.

12) ‘***Invalid expression before directive***’: Some directives cannot be preceded by any symbol or label (In case of this assembler, they are END and ORG). If these directives are preceded by any label/symbol, this error is raised.

13) ‘***Label/Symbol not found in symbol table***’: This is an error that is raised during pass-2 of the assembler (after construction of symbol table). If a label/symbol is used as an operand for an instruction and it is not found in the symbol table, (which means that the label/symbol was not declared), this error is raised.

2.5) RAM and ROM space

The RAM and ROM space can be modified by changing their starting addresses. This can be done by assigning the desired values to the global integer variables ‘program_counter’ and ‘RAM_counter’. By default, the ROM space is from 0000h to 7FFFh and RAM space is from 8000h to FFFFh.



```
int program_counter = 0x0000;
int RAM_counter = 0x8000;
```

3) <u>Structure of line in assembly code</u>

The basic construction of any line of assembly code is:

label/symbol	**mnemonic/directive**	**operand**	**; comment**
(1)	(2)	(3)	(4)

The fields may be separated by any number of blanks, but must be separated by at least one blank. Each instruction and directive must be entered on a single line terminated by a carriage return (enter key). No continuation lines are possible, but lines consisting entirely of comments are allowed.

The assembler supports lower-case and upper-case characters. But it is case insensitive, so lower-case and upper-case characters are treated the same.

Note: Refer chapter 2 of the document 8080/8085 assembly language programming manual for further details.