Pass1

Requirements Specification :

Assembler that excutes pass1 on entering pass1<src file name> .

The parser of the assembler should handle :

- 2 byte instructions as TIXR A , ADDR S,A .
- 3 , 4 byte instructions with symbolic or non-symbolic operands including immediate , indirect , and indexed addressing .
- All storage directives (byte,word,resb,resw) in addition to start , end.
- Instructions and directives in upper or lower cases.
- The fixed format of the program which is :
    - bytes 1–8 label
    - 9 blank
    - 10–15 operation code
    - 16–17 blank
    - 18–35 operand
    - 36–66 comment
- If the line contains a '.' in the first byte , the entire line is a comment.

The output of pass 1 consists of the symbol table , and output file containing source program in a required format along with the addresses and the errors if any .

'Line no. address statement line as read from file'

******error in the statement –if any-**

Design :

The pass1 assembler is divided into consecutive stations :

1- Parsing .

2- Loading op-table.

3- Validation.

4- Addressing and printing.

All these stations share a public vector containing the source code in a more organized form , statements vector .

The parsing station is responsible for reading the source code from the file and parsing it into a statement structure and pushing it to the public vector in order to be used by successive stations .

The second station is considered is responsible for loading the op-table containing the operations along with their opcodes , formats , and other characteristics which may be needed in further stations .

The third stations Is responsible for validating each statement stored into the statements vector , if an error is found in any of them the error attribute of the statement is set with a value according to the error type .

The fourth station deals with the statements vector after it has been validated , it assign addresses to each statement consecutively according to addressing rules , taking errors in concerns , producing the final statements vector that will be printed to the output file with addresses , source line , and the error of the statement if any .

Main data structures should go here

Vector
it has been used through the program to hold data in many regions

Set
it has been used to represent the SYMTAB to include only one copy of each symbol

Structs
used to form a bundle of data types required to fulfill some tasks like :
- statement : is a struct contains variables represent the contents of the statement in the program to be assembled such as : string label , string mnemonic , vector<string> operand, string comment , string line , int error , int address , bool n, i, x, b, p, e , bool is_comment.
- op_line : is a struct contains some variables representing the mnemonic such as : string mnemonic , opcode , int format, no_args , char input_type, category , bool directive_flag, good.

Algorithms description

Parsing and validating format:

Design and algorithm used:

It was required in this part to parse the lines read from the file line by line and construct a vector of statements where each statement contains its: label, mnemonic, operands, comment and flag bits for this statement.

First the line is converted to small letters to ease the process of validation then parsed into these parts according to their places which are:

1. Bytes 1–8 label

2. 9 blank

3. 10–15 operation code

4. 16–17 blank

5. 18–35 operand

6. 36–66 comment

Then they are put as substrings in their right places and if the format of the line isn't like the legal format an illegal format error for this statement is assigned.

And for each substring (label, mnemonic, operands) we check if it in the right format and that it is in the right place and containing no spaces in it

For flags a check for '+' sign is done to set the 'e' flag also a check for the existence for '#' or '@' signs to set 'n' and 'I' flags. Also a check is done to search for ',x' to know if there is indexed addressing or not.

Assumptions:

- If there is a 'tab' in a line the statement error is set to '0' which is illegal format error
- If a line is empty or containing only spaces, it will be skipped
- If the operands or label doesn't exist it doesn't mean illegal format, this part only checks for the existence of mnemonics
- If there is a space in the operands substring it gives an error unless this space is found between " C' ' "
- A comment is added to the statement if a string is found beyond index 35

<u>Loading op-table :</u>

This part or station was reading the opcode table from a file called "op_file.txt":

Simply it reads from the file each record according to the type of it.

In the file itself the data was divided into rows and columns, each row stands for an operation command.

The columns where divided into 7 columns: mnemonic, input type, format, opcode, number of arguments, directive flag and category.

Mnemonics: Holds the name of the mnemonic itself for example: ADD, JSUB, CLEAR, …. etc.

Input type: This was a column that stands for the type of input to be read, which is memory or address type, and register type meaning that this operation takes either a memory input or register input. Some assumptions that should be mentioned, some operations don't need an input they have there on input type meaning no input, and other operations are directives they also have another type of input.

Format: The data in this columns shows whether the data is of type 1, 2, 3, 4, in the table in the pdf there was only 2 types, format 2 and format 3 / 4 which was changed to be format 5 as a shortcut.

Opcode: Simply holds the opcode of the operation.

Number of operands: Each operation holds a number of operands or needs a certain number of operands this column holds this kind of information, any important assumption to be mentioned is that the number of arguments need for an instruction like WORD is infinity or as much as the line can take there for a huge number was put for the this certain operation.

Directive flag: Is just a boolean added to show if the operation is a directive or not.

Category: Is the last column where, each operation is given a certain letter, this letter helps in validation.

The previous input is read into a map data structure, where the key is the mnemonic name or a string, and the value that it holds is a struct divided similarly to the data explained.

The map which holds the data, is put into a struct which has some functions to help use it, first the main function called create which opens the file and reads the data from it, second a function called get this function returns an opcode struct, it searches through the maps and returns the required data if found, and returns an empty struct if not found, lastly a function called print which prints the data which exists in a struct just like taken from the file, the purpose of this function is for testing only.

Validation :

Validation station has 3 steps , validate start directive , validate end directive and validate each statement in the program .

Each statement is validated as following :

-If there is a label in the symbol field , this label is checked to be a valid one , fist digit must be alphabetic character or dollar sign and the other digits can be alphabetic character , dollar sign or numbers (0-9) otherwise "Illegal label" is set to the error flag , if there is a duplication in label definition "duplicate label definition" is set to the error flag .

-The operation is checked to be in the operation table otherwise "Illegal mnemonic" is set to the error flag .

-The operands type (memory , register , hexadecimal digits , ascii characters, decimal Integers) and its number depend on the mnemonic , the mnemonics are divided into groups , each group has the same types and number of operands , if the types or the number of the operands are not suitable for the mnemonic of the statement " Illegal operand(s)" is set to the error flag .

- Addressing mode is checked to be a valid mode and if there is an illegal addressing mode , "Illegal Addressing Mode" is set to the flag error .

Last Station -addressing and printing - :

After the statements are parsed , validated , and stored in statements vector , iterations are performed on statements in order to assign addresses .

Starting address is 0 by default , unless the start statement is valid and has a different starting address.

A statement is given the current address , then the address is incremented by such rules :

- If the statement is **not a directive** , then the address is incremented with a value equals to the operation format (obtained from the op-table).

- If the statement is **errored** , it is assigned with the current address but the address is not incremented .
- If the statement is a **BYTE** directive , in case its operand is of type hexadecimal - **X'11ff'** – the length of the operand must be even as each 2 characters represent a byte , so the address is incremented by the length of the operand/2 .
- If the statement is a **BYTE** directive , In case the operand is of type character - **C'abc'** – each character represents a byte so the address is incremented by the length of the operand .
- If the statement is a **WORD** directive , it may take a list of operands separated by commas , each operand represents 3 bytes . so the address is incremented by the number of operands * 3.
- If the statement is **RESB** directive , the address is incremented by the value of the operand .
- If the statement is **RESW** directive , the address is incremented by the value of the operand *3 .

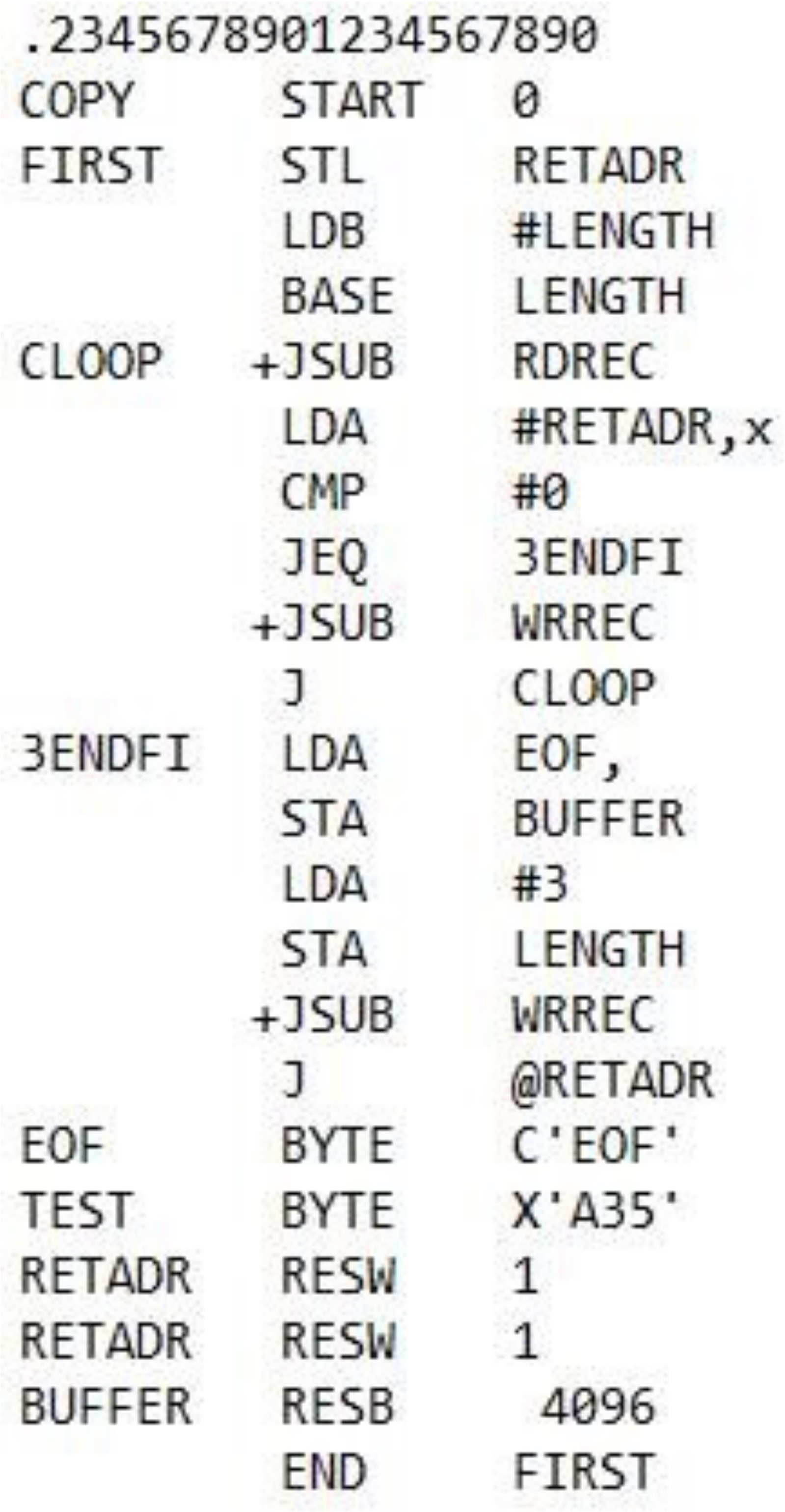
After assigning the addresses for statements based on previous rules and assumptions , the statements are printed to the output file organized in the required form :

'Line no.   address   statement line as read from file'

******error in the statement –if any-**

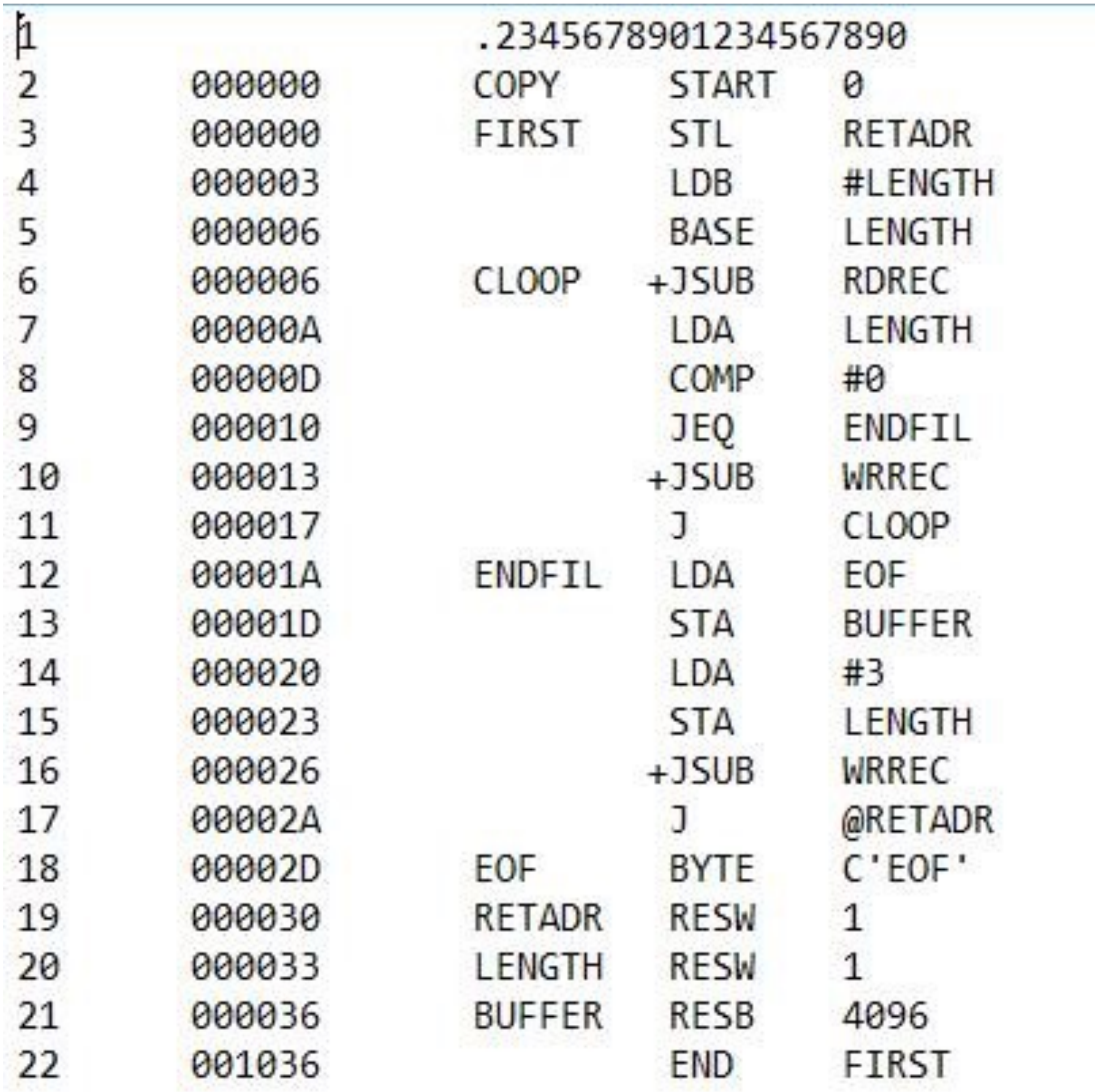
<u>sample runs</u>

program.txt



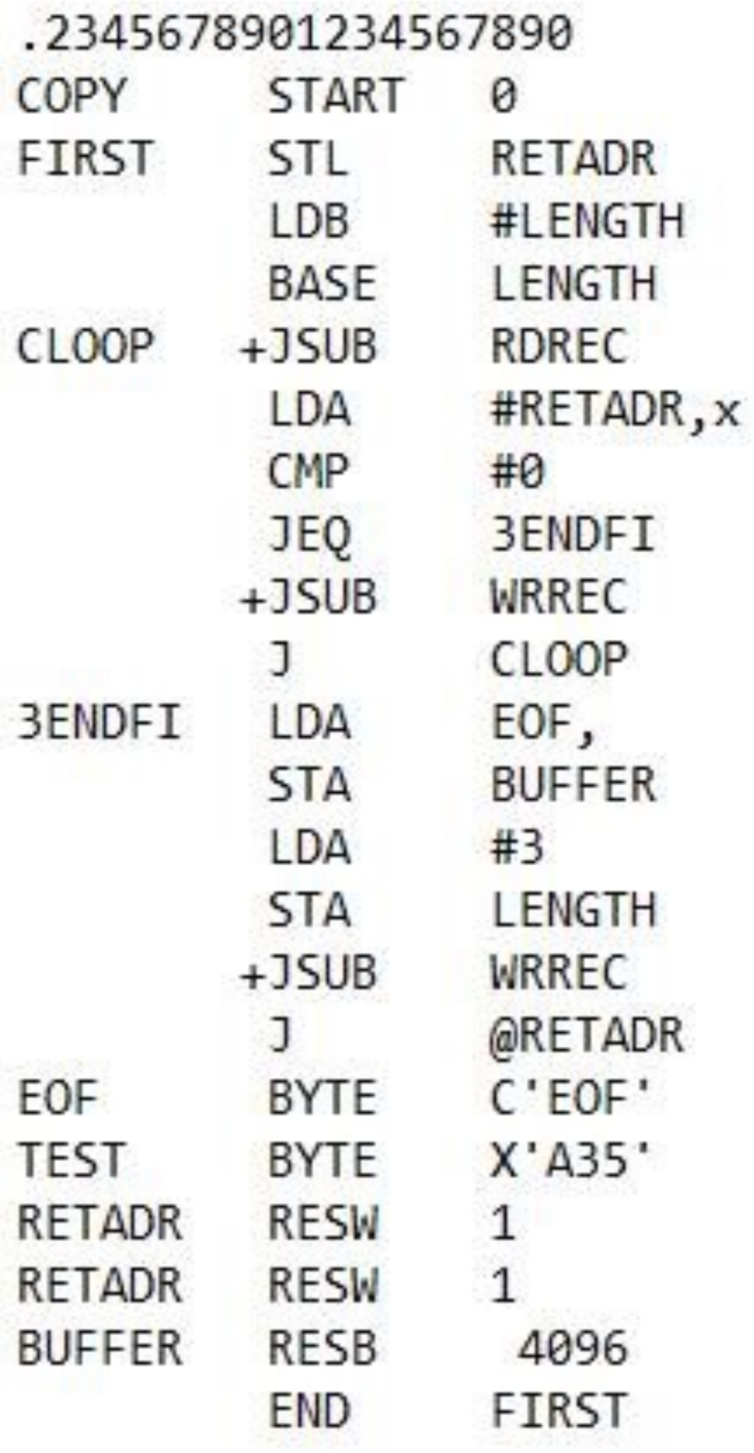
```
.2345678901234567890
COPY     START   0
FIRST    STL     RETADR
         LDB     #LENGTH
         BASE    LENGTH
CLOOP   +JSUB    RDREC
         LDA     #RETADR,x
         CMP     #0
         JEQ     3ENDFI
        +JSUB    WRREC
         J       CLOOP
3ENDFI   LDA     EOF,
         STA     BUFFER
         LDA     #3
         STA     LENGTH
        +JSUB    WRREC
         J       @RETADR
EOF      BYTE    C'EOF'
TEST     BYTE    X'A35'
RETADR   RESW    1
RETADR   RESW    1
BUFFER   RESB     4096
         END     FIRST
```

output.txt



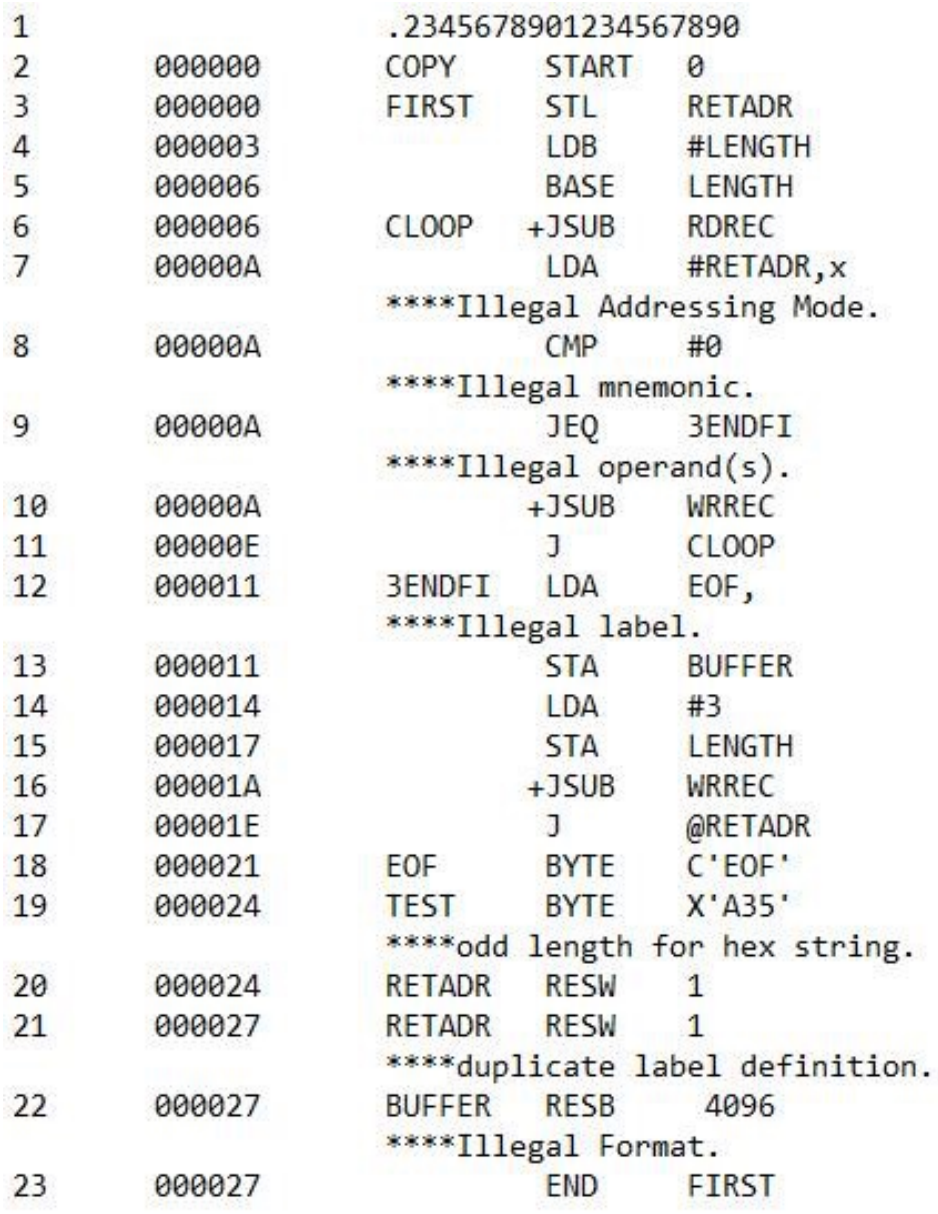
```
1                   .2345678901234567890
2       000000      COPY    START   0
3       000000      FIRST   STL     RETADR
4       000003              LDB     #LENGTH
5       000006              BASE    LENGTH
6       000006      CLOOP  +JSUB    RDREC
7       00000A              LDA     LENGTH
8       00000D              COMP    #0
9       000010              JEQ     ENDFIL
10      000013             +JSUB    WRREC
11      000017              J       CLOOP
12      00001A      ENDFIL  LDA     EOF
13      00001D              STA     BUFFER
14      000020              LDA     #3
15      000023              STA     LENGTH
16      000026             +JSUB    WRREC
17      00002A              J       @RETADR
18      00002D      EOF     BYTE    C'EOF'
19      000030      RETADR  RESW    1
20      000033      LENGTH  RESW    1
21      000036      BUFFER  RESB    4096
22      001036              END     FIRST
```

program.txt



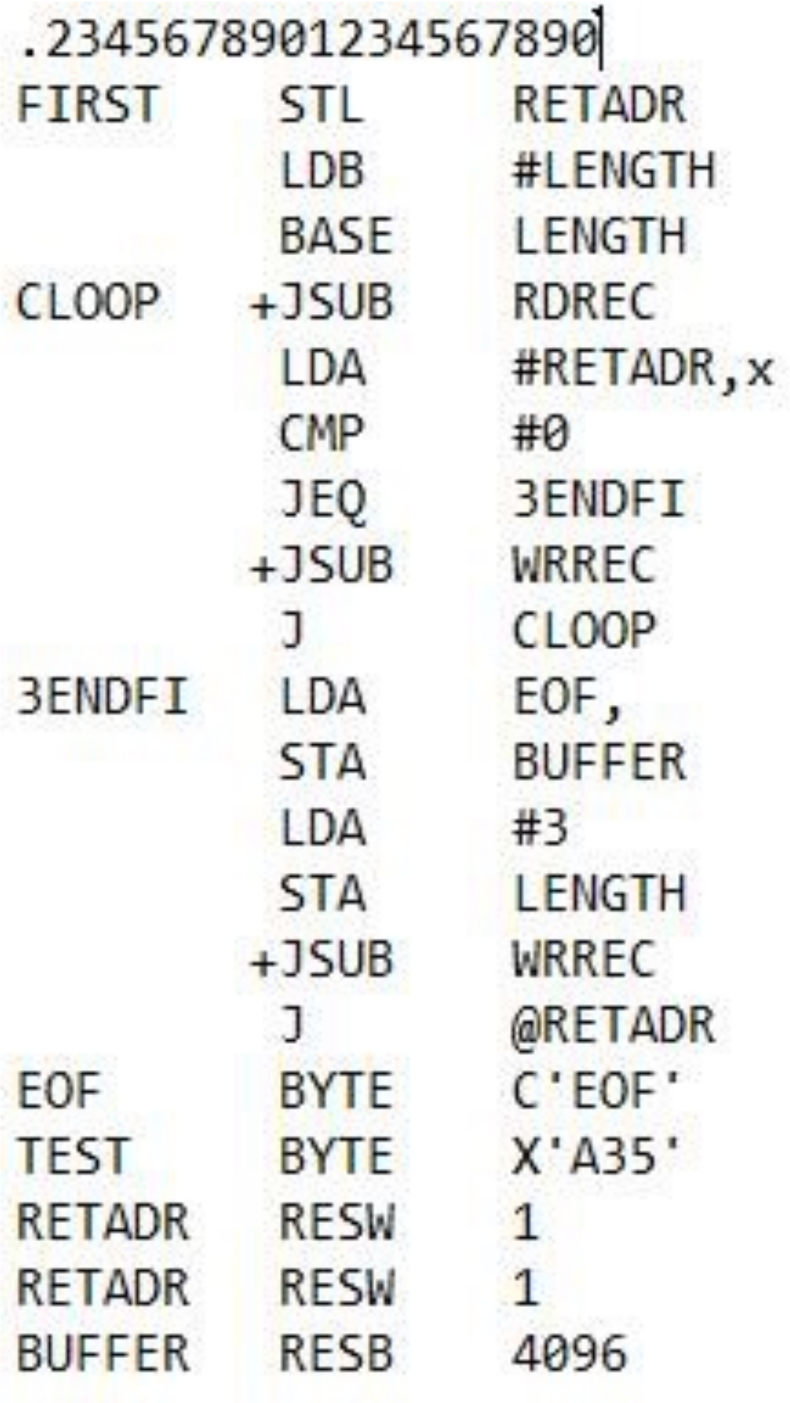
```
.2345678901234567890
COPY     START   0
FIRST    STL     RETADR
         LDB     #LENGTH
         BASE    LENGTH
CLOOP   +JSUB    RDREC
         LDA     #RETADR,x
         CMP     #0
         JEQ     3ENDFI
        +JSUB    WRREC
         J       CLOOP
3ENDFI   LDA     EOF,
         STA     BUFFER
         LDA     #3
         STA     LENGTH
        +JSUB    WRREC
         J       @RETADR
EOF      BYTE    C'EOF'
TEST     BYTE    X'A35'
RETADR   RESW    1
RETADR   RESW    1
BUFFER   RESB     4096
         END     FIRST
```

output.txt



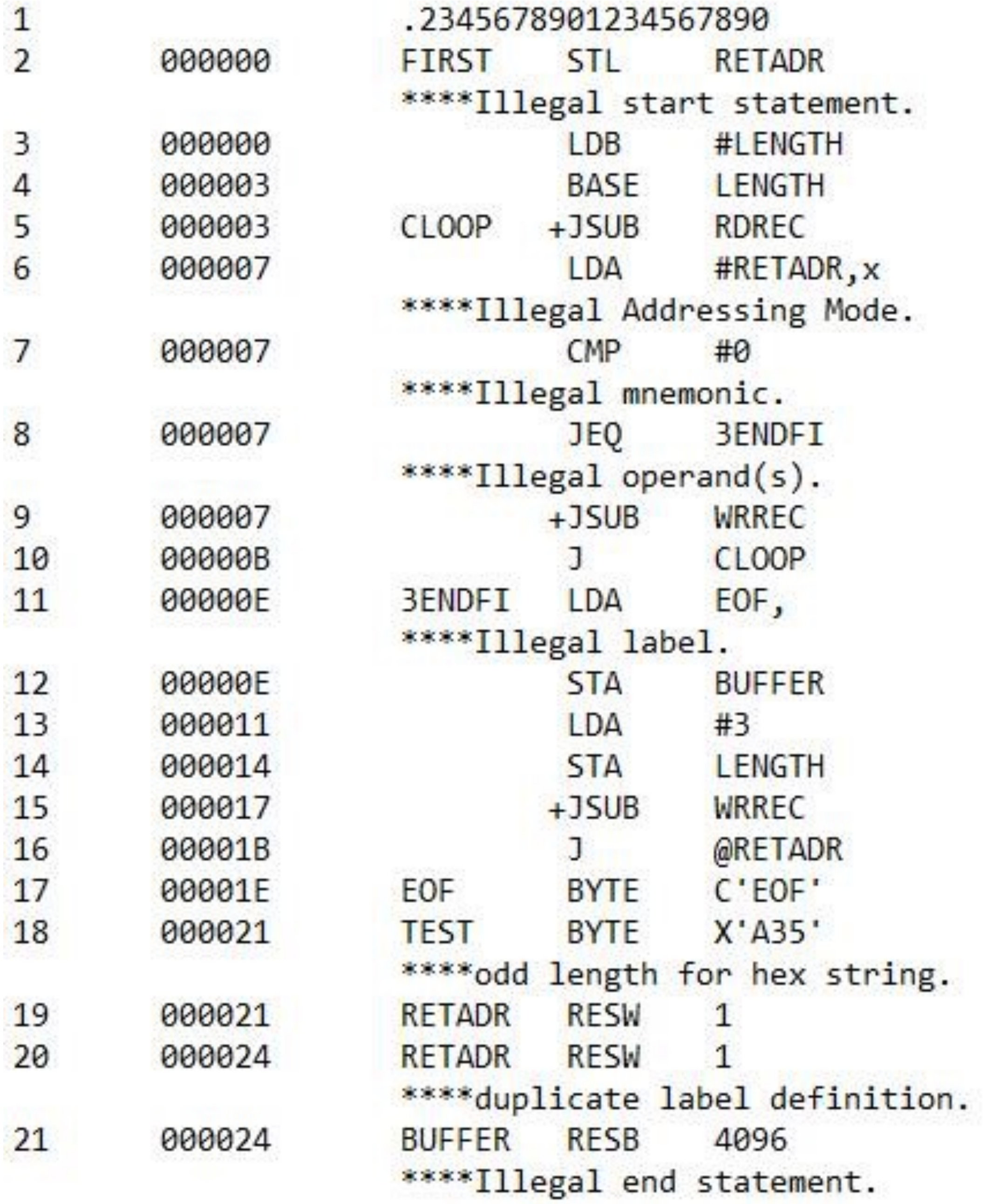
```
1                    .234567890123456789O
2       000000       COPY     START   0
3       000000       FIRST    STL     RETADR
4       000003                LDB     #LENGTH
5       000006                BASE    LENGTH
6       000006       CLOOP   +JSUB    RDREC
7       00000A                LDA     #RETADR,x
                     ****Illegal Addressing Mode.
8       00000A                CMP     #0
                     ****Illegal mnemonic.
9       00000A                JEQ     3ENDFI
                     ****Illegal operand(s).
10      00000A               +JSUB    WRREC
11      00000E                J       CLOOP
12      000011       3ENDFI   LDA     EOF,
                     ****Illegal label.
13      000011                STA     BUFFER
14      000014                LDA     #3
15      000017                STA     LENGTH
16      00001A               +JSUB    WRREC
17      00001E                J       @RETADR
18      000021       EOF      BYTE    C'EOF'
19      000024       TEST     BYTE    X'A35'
                     ****odd length for hex string.
20      000024       RETADR   RESW    1
21      000027       RETADR   RESW    1
                     ****duplicate label definition.
22      000027       BUFFER   RESB     4096
                     ****Illegal Format.
23      000027                END     FIRST
```

program.txt



```
.2345678901234567890
FIRST    STL     RETADR
         LDB     #LENGTH
         BASE    LENGTH
CLOOP   +JSUB    RDREC
         LDA     #RETADR,x
         CMP     #0
         JEQ     3ENDFI
        +JSUB    WRREC
         J       CLOOP
3ENDFI   LDA     EOF,
         STA     BUFFER
         LDA     #3
         STA     LENGTH
        +JSUB    WRREC
         J       @RETADR
EOF      BYTE    C'EOF'
TEST     BYTE    X'A35'
RETADR   RESW    1
RETADR   RESW    1
BUFFER   RESB    4096
```

output.txt



```
1                     .234567890123456789O
2       000000        FIRST    STL     RETADR
                      ****Illegal start statement.
3       000000                 LDB     #LENGTH
4       000003                 BASE    LENGTH
5       000003        CLOOP   +JSUB    RDREC
6       000007                 LDA     #RETADR,x
                      ****Illegal Addressing Mode.
7       000007                 CMP     #0
                      ****Illegal mnemonic.
8       000007                 JEQ     3ENDFI
                      ****Illegal operand(s).
9       000007                +JSUB    WRREC
10      00000B                 J       CLOOP
11      00000E        3ENDFI   LDA     EOF,
                      ****Illegal label.
12      00000E                 STA     BUFFER
13      000011                 LDA     #3
14      000014                 STA     LENGTH
15      000017                +JSUB    WRREC
16      00001B                 J       @RETADR
17      00001E        EOF      BYTE    C'EOF'
18      000021        TEST     BYTE    X'A35'
                      ****odd length for hex string.
19      000021        RETADR   RESW    1
20      000024        RETADR   RESW    1
                      ****duplicate label definition.
21      000024        BUFFER   RESB    4096
                      ****Illegal end statement.
```