University *of* Alexandria
Faculty of Engineering
Computer and Systems Engineering Department

Term Project - SIC/XE Assembler
Phase (1)

Ahmed Elsayed Mahmoud(5)

Ahmed Eid Abdelmon'em (9)

Shehab Kamal El-samany (32)

Mohamed Ahmed Abd-ElTwab (52)

Mohamed Samir Shaaban(56)

Requirements specification:

- The term project is to implement a (cross) assembler for (a subset of) SIC/XE assembler , written in C/C++ , producing code for the absolute loader used in the SIC/XE programming assignments.

- The pass1 is to execute by entering pass1 <source-file-name> .

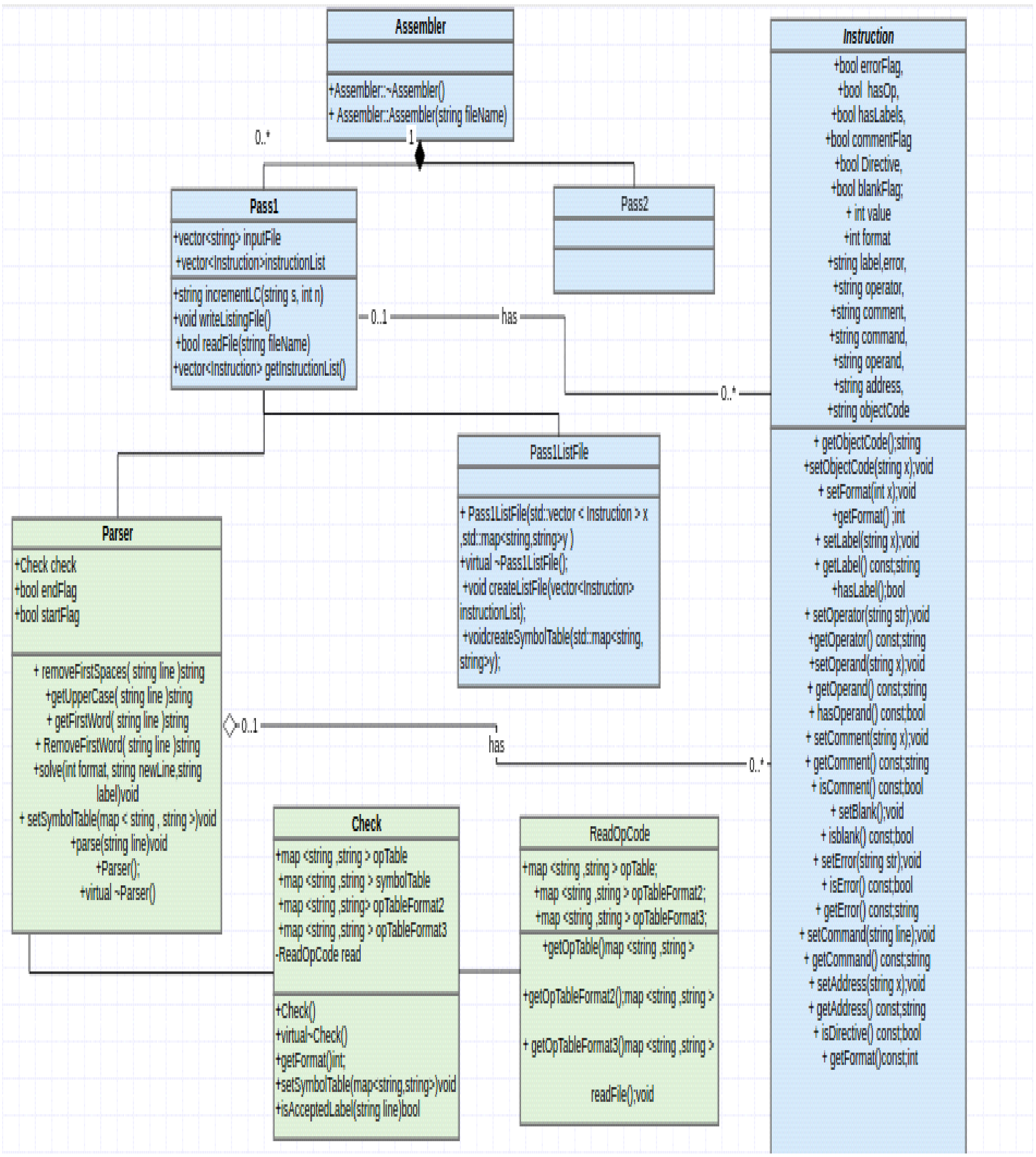
- The source file for the main program for this phase is to be named pass1.c .

- build a parser that is capable of handling source lines that are instructions, storage declaration, comments, and assembler directives (a directive that is not implemented should be ignored possibly with a warning) .

-The output of this phase should contain (at least):

- The symbol table.
- The source program

Design:


Assembler
+Assembler::~Assembler()
+ Assembler::Assembler(string fileName)
1
0..*
Pass1
+vector<string> inputFile
+vector<Instruction>instructionList
+string incrementLC(string s, int n)
+void writeListingFile()
+bool readFile(string fileName)
+vector<Instruction> getInstructionList()
Pass2
0..1
has
0..*
Instruction
+bool errorFlag,
+bool hasOp,
+bool hasLabels,
+bool commentFlag
+bool Directive,
+bool blankFlag;
+ int value
+int format
+string label,error,
+string operator,
+string comment,
+string command,
+string operand,
+string address,
+string objectCode
+ getObjectCode():string
+setObjectCode(string x):void
+ setFormat(int x):void
+getFormat() :int
+ setLabel(string x):void
+ getLabel() const:string
+hasLabel():bool
+ setOperator(string str):void
+getOperator() const:string
+setOperand(string x):void
+ getOperand() const:string
+ hasOperand() const:bool
+ setComment(string x):void
+ getComment() const:string
+ isComment() const:bool
+ setBlank():void
+ isblank() const:bool
+ setError(string str):void
+ isError() const:bool
+ getError() const:string
+ setCommand(string line):void
+ getCommand() const:string
+ setAddress(string x):void
+ getAddress() const:string
+ isDirective() const:bool
+ getFormat()const:int
Pass1ListFile
+ Pass1ListFile(std::vector < Instruction > x ,std::map<string,string>y)
+virtual ~Pass1ListFile();
+void createListFile(vector<Instruction> instructionList);
+voidcreateSymbolTable(std::map<string, string>y);
Parser
+Check check
+bool endFlag
+bool startFlag
+ removeFirstSpaces(string line)string
+getUpperCase(string line)string
+ getFirstWord(string line)string
+ RemoveFirstWord(string line)string
+solve(int format, string newLine,string label)void
+ setSymbolTable(map < string , string >)void
+parse(string line)void
+Parser();
+virtual ~Parser()
0..1
has
0..*
Check
+map <string ,string > opTable
+map <string ,string > symbolTable
+map <string ,string> opTableFormat2
+map <string ,string > opTableFormat3
-ReadOpCode read
+Check()
+virtual~Check()
+getFormat()int;
+setSymbolTable(map<string,string>)void
+isAcceptedLabel(string line)bool
ReadOpCode
+map <string ,string > opTable;
+map <string ,string > opTableFormat2;
+map <string ,string > opTableFormat3;
+getOpTable()map <string ,string >
+getOpTableFormat2():map <string ,string >
+ getOpTableFormat3()map <string ,string >
readFile():void

Main data structures:

- **map:**

  - map is a sorted associative container that contains key-value pairs with unique keys. Keys are sorted by using the comparison function Compare. Search, removal, and insertion operations have logarithmic complexity. Maps are usually implemented as red-black tree.

  - we use it to store the op-Table and get the operator foramt and get operator op-code.

- **vector:**

  - vector is a sequence container that encapsulates dynamic size arrays.

  - why we use it ?

    - Random access - constant *O(1)*
    - Insertion or removal of elements at the end - amortized constant *O(1).*
    - Insertion or removal of elements - linear in distance to the end of the vector *O(n).*

  - we use it to store each instruction in it.

- **unordered-set:**

  - Unordered set is an associative container that contains a set of unique objects of type Key. Search, insertion, and removal have average constant-time complexity.

- **Queue:**

  - are a type of container adaptor, specifically designed to operate in a FIFO context (first-in first-out), where elements are inserted into one end of the container and extracted from the other.

  - we use it to store operator and operant in fixed format.

Algorithms description:

- **getUpperCase(string line):**

  - change char from lower case to upper case.

- **readFile(string fileName):**

  - read source code form given file

  - store each line in vector.

- **Pass1(string fileName,int type):**

  - take two parameter :

    - file Name : the name of file where the source code had been written.

    - type: to Distinguish between fixed format and free format.

  - call read file function to read file.

  - call parse function to parse each line and get the instruction which contain some flags and some string.

- make some check :

  - ➢ if this line's operator "END" set the end flag equal true because it is illegal to write any instruction after end in source code.
  - ➢ if instruction don't have any errors make other check :
    - ✓ if the operator is "START" set the address by current location counter and set the label in symbol table if any.
    - ✓ if the operator is "END" set the address by current location counter.
    - ✓ otherwise increment the location counter by the operator's format and set the label to symbol table if any .
  - ➢ if the code don't have "END" operator set end error.

- **getFormat(string line):**

  - make some check :

  - ➢ if the operator has '+' then check if it in the op Table format 3 return 3 otherwise return 0.
  - ➢ if the operator in the directive list return 1 .
    - ✓ 1 to know it is directive when i solve in pass 1.

        - ✓ 0 to know it is not valid operator and set error message.
    - ➢ if the operator in the op table format 3 return 3.
    - ➢ if the operator in the op table format 4 return4.
    - ➢ otherwise return 0.

- **parse(string line, int format):**

  - take two parameter:

    - ➢ line : each line we read form the file.
    - ➢ type: to Distinguish between fixed format and free format.

  - if fixed format call the validate Fixed function.

  - make some check:

    - ➢ if line is empty or have new line set in instruction zero format and blank line.
    - ➢ if the line have '.' in the first set in instruction comment line.
    - ➢ if the instruction had been written after end operator set " unrecognised operation error".
    - ➢ if the label in the symbol table set" redifined Label"
    - ➢ otherwise call solve function .

- **solve(int format, string line,string label, string operators):**
    - ➢ check if the operator different from operant .
    - ➢ if format equal 0 set " unrecognised operation"
    - ➢ if format greater than one we have three cases:
        - ✓ if format equal 2 set instruction format = 2.
        - ✓ if format equal 3 set instruction format = 3.
        - ✓ if format equal 4 set instruction format = 4.
    - ➢ if format equal one so it is directive operator then make some check to know if it "BASE" or "NONASE" or "ORG" or "EQU" or "LTORG" or "START" or "END" or "BYTE" or "RESB" or "RESW" or "WORD" and in each case set the instruction format by own format.
- **validateFixed(string input):**

  - parse the line by fixed format :

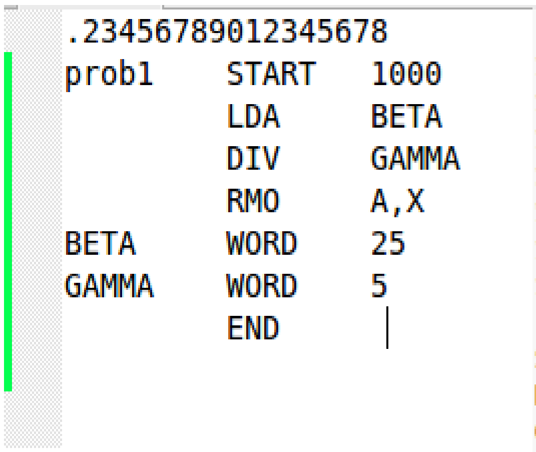
    - ➢ bytes 1–8 label.
    - ➢ 9 blank.
    - ➢ 10–15 operation code.
    - ➢ 16–17 blank.
    - ➢ 18–35 operand.
    - ➢ 36–66 comment.

Assumptions :

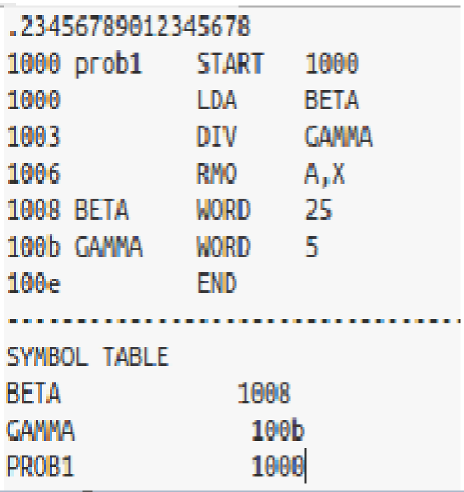
- default start address equal zero in case not mentioned in the code.
- free format is handled.
- no comment is supported on the same line of code.
- code is case insensitive.
- line which has an error has won't be saved in memory.

Sample Runs :

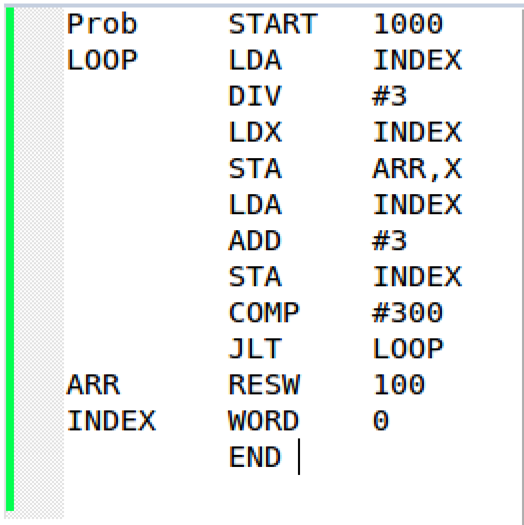
- **fixed format tests:**



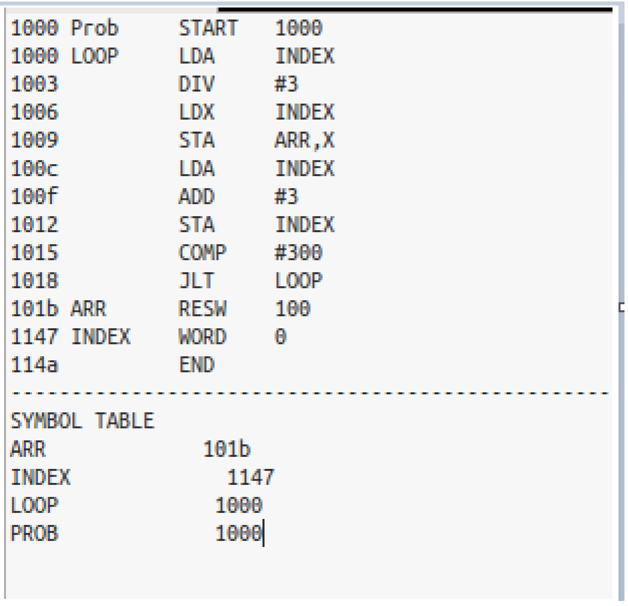
```
.23456789012345678
prob1     START   1000
          LDA     BETA
          DIV     GAMMA
          RMO     A,X
BETA      WORD    25
GAMMA     WORD    5
          END
```



```
.23456789012345678
1000 prob1     START   1000
1000           LDA     BETA
1003           DIV     GAMMA
1006           RMO     A,X
1008 BETA      WORD    25
100b GAMMA     WORD    5
100e           END
---------------------------------------
SYMBOL TABLE
BETA                1008
GAMMA               100b
PROB1               1000
```

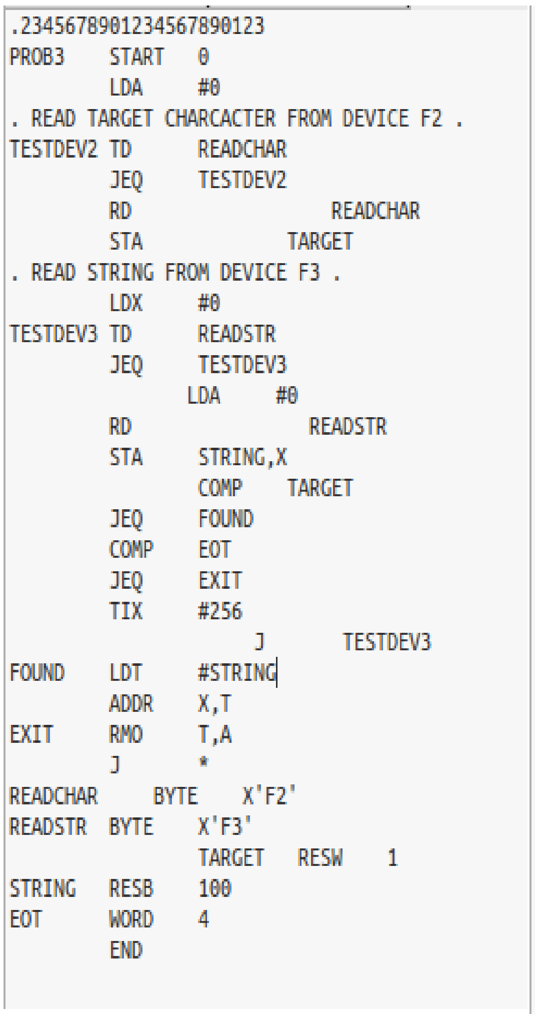


```
Prob      START   1000
LOOP      LDA     INDEX
          DIV     #3
          LDX     INDEX
          STA     ARR,X
          LDA     INDEX
          ADD     #3
          STA     INDEX
          COMP    #300
          JLT     LOOP
ARR       RESW    100
INDEX     WORD    0
          END
```

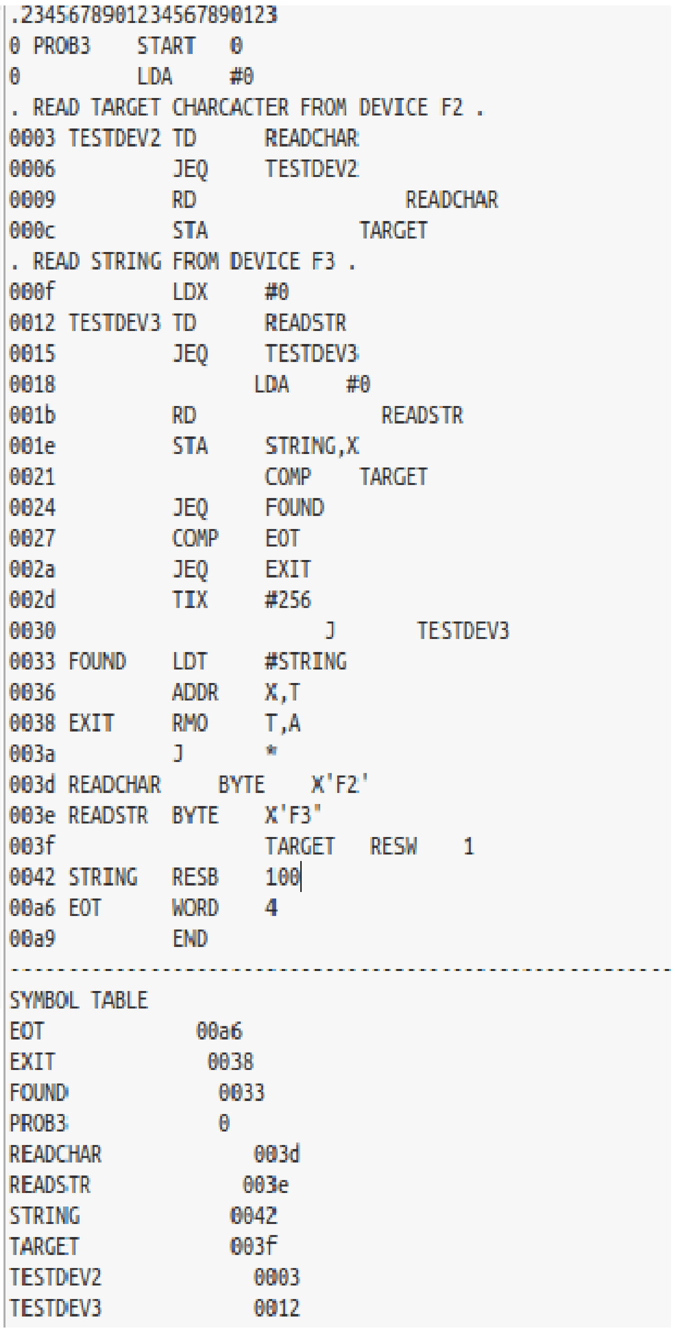


```
1000 Prob      START   1000
1000 LOOP      LDA     INDEX
1003           DIV     #3
1006           LDX     INDEX
1009           STA     ARR,X
100c           LDA     INDEX
100f           ADD     #3
1012           STA     INDEX
1015           COMP    #300
1018           JLT     LOOP
101b ARR       RESW    100
1147 INDEX     WORD    0
114a           END
-----------------------------------------------
SYMBOL TABLE
ARR              101b
INDEX              1147
LOOP             1000
PROB             1000
```

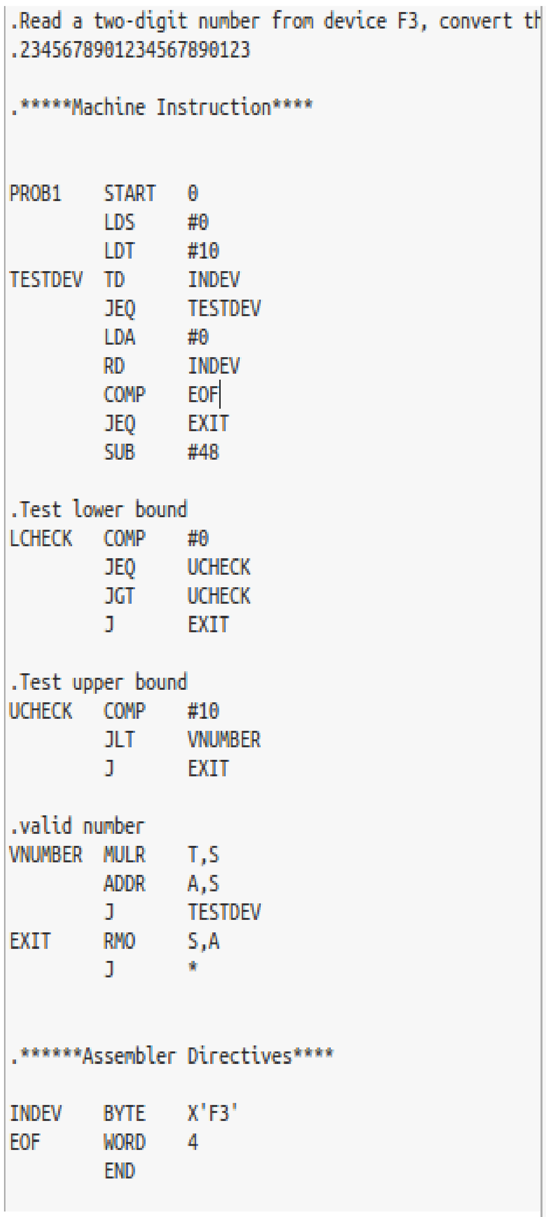
- **free format tests:**



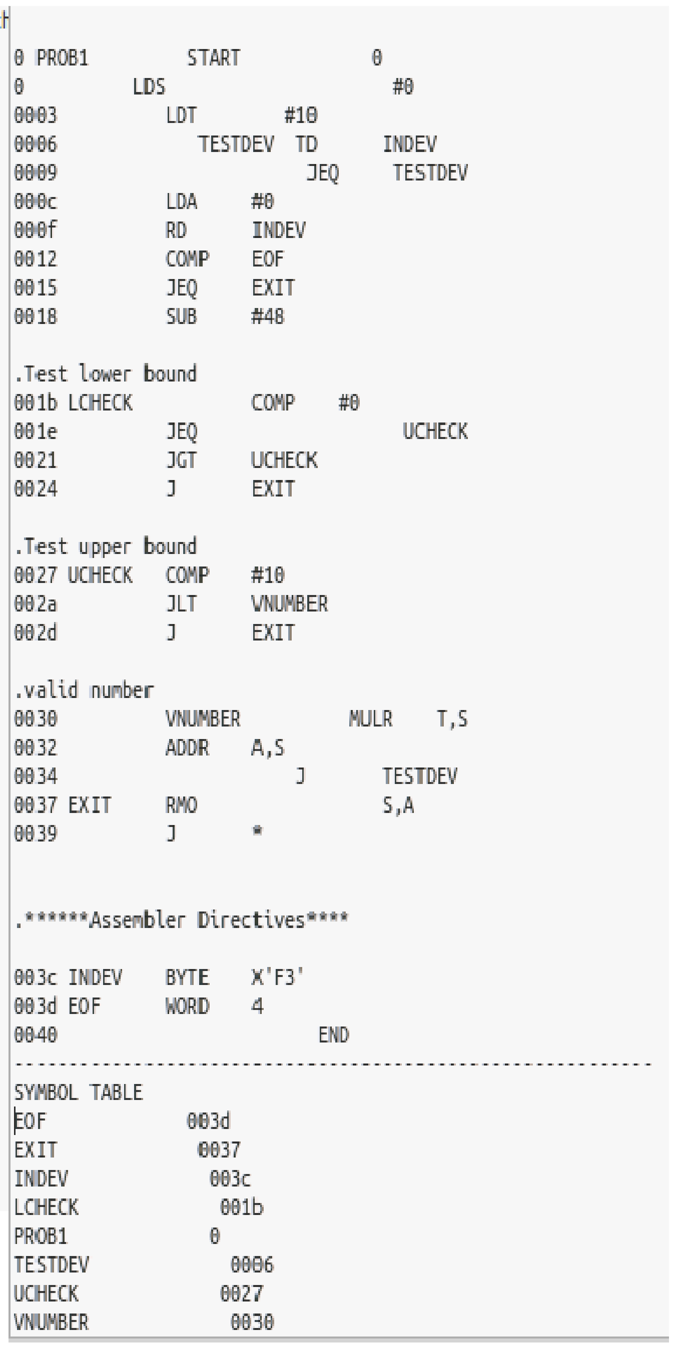
```
.234567890123456789012З
PROB3    START   0
         LDA     #0
. READ TARGET CHARCACTER FROM DEVICE F2 .
TESTDEV2 TD      READCHAR
         JEQ     TESTDEV2
         RD                  READCHAR
         STA             TARGET
. READ STRING FROM DEVICE F3 .
         LDX     #0
TESTDEV3 TD      READSTR
         JEQ     TESTDEV3
                LDA     #0
         RD                READSTR
         STA     STRING,X
                 COMP    TARGET
         JEQ     FOUND
         COMP    EOT
         JEQ     EXIT
         TIX     #256
                      J       TESTDEV3
FOUND    LDT     #STRING
         ADDR    X,T
EXIT     RMO     T,A
         J       *
READCHAR     BYTE    X'F2'
READSTR  BYTE    X'F3'
                 TARGET   RESW    1
STRING   RESB    100
EOT      WORD    4
         END
```



```
.234567890123456789012З
0 PROB3    START   0
0          LDA     #0
. READ TARGET CHARCACTER FROM DEVICE F2 .
0003 TESTDEV2 TD      READCHAR
0006          JEQ     TESTDEV2
0009          RD                  READCHAR
000c          STA             TARGET
. READ STRING FROM DEVICE F3 .
000f          LDX     #0
0012 TESTDEV3 TD      READSTR
0015          JEQ     TESTDEV3
0018                 LDA     #0
001b          RD                READSTR
001e          STA     STRING,X
0021                  COMP    TARGET
0024          JEQ     FOUND
0027          COMP    EOT
002a          JEQ     EXIT
002d          TIX     #256
0030                       J       TESTDEV3
0033 FOUND    LDT     #STRING
0036          ADDR    X,T
0038 EXIT     RMO     T,A
003a          J       *
003d READCHAR     BYTE    X'F2'
003e READSTR  BYTE    X'F3'
003f                  TARGET   RESW    1
0042 STRING   RESB    100
00a6 EOT      WORD    4
00a9          END
---------------------------------------------------------
SYMBOL TABLE
EOT             00a6
EXIT             0038
FOUND             0033
PROB3             0
READCHAR             003d
READSTR             003e
STRING            0042
TARGET            003f
TESTDEV2             0003
TESTDEV3             0012
```

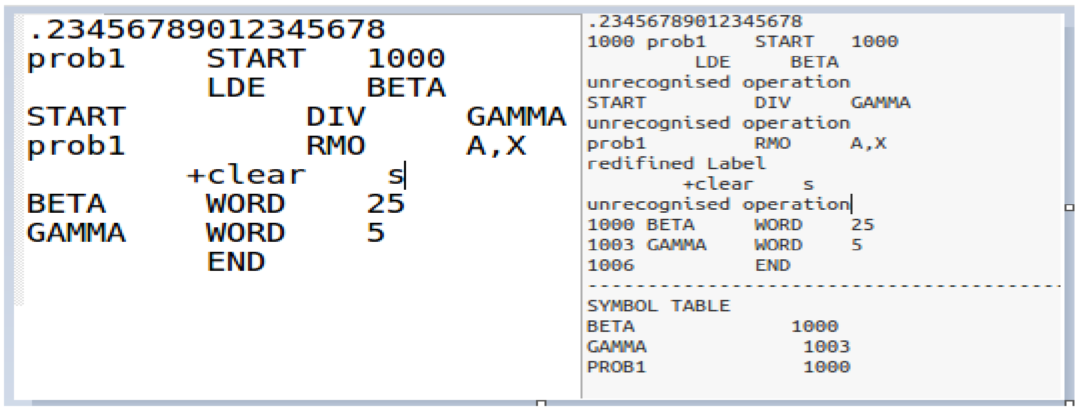


```
.Read a two-digit number from device F3, convert th
.2345678901234567890123

.*****Machine Instruction****


PROB1    START   0
         LDS     #0
         LDT     #10
TESTDEV  TD      INDEV
         JEQ     TESTDEV
         LDA     #0
         RD      INDEV
         COMP    EOF
         JEQ     EXIT
         SUB     #48

.Test lower bound
LCHECK   COMP    #0
         JEQ     UCHECK
         JGT     UCHECK
         J       EXIT

.Test upper bound
UCHECK   COMP    #10
         JLT     VNUMBER
         J       EXIT

.valid number
VNUMBER  MULR    T,S
         ADDR    A,S
         J       TESTDEV
EXIT     RMO     S,A
         J       *


.******Assembler Directives****

INDEV    BYTE    X'F3'
EOF      WORD    4
         END
```



```
0 PROB1        START          0
0         LDS                    #0
0003          LDT        #10
0006              TESTDEV  TD      INDEV
0009                          JEQ     TESTDEV
000c          LDA     #0
000f          RD      INDEV
0012          COMP    EOF
0015          JEQ     EXIT
0018          SUB     #48

.Test lower bound
001b LCHECK         COMP    #0
001e          JEQ                     UCHECK
0021          JGT     UCHECK
0024          J       EXIT

.Test upper bound
0027 UCHECK   COMP    #10
002a          JLT     VNUMBER
002d          J       EXIT

.valid number
0030          VNUMBER         MULR    T,S
0032          ADDR    A,S
0034                         J       TESTDEV
0037 EXIT     RMO                    S,A
0039          J       *


.******Assembler Directives****

003c INDEV    BYTE    X'F3'
003d EOF      WORD    4
0040                        END
-----------------------------------------------------------
SYMBOL TABLE
EOF             003d
EXIT             0037
INDEV             003c
LCHECK             001b
PROB1            0
TESTDEV             0006
UCHECK             0027
VNUMBER             0030
```

- **critical test:**



```
.2345678901234567
prob1       START     1000
            LDE       BETA
START               DIV       GAMMA
prob1               RMO       A,X
          +clear      s
BETA        WORD      25
GAMMA       WORD      5
            END
```

```
.2345678901234567 8
1000 prob1     START    1000
          LDE      BETA
unrecognised operation
START          DIV      GAMMA
unrecognised operation
prob1          RMO      A,X
redifined Label
         +clear     s
unrecognised operation
1000 BETA      WORD     25
1003 GAMMA     WORD     5
1006           END
------------------------------------------
SYMBOL TABLE
BETA                1000
GAMMA                1003
PROB1                1000
```