



The National Institute of Engineering

(Autonomous Institution)

Mysore – 570008

A project Report of

2 PASS ASSEMBLER

Submitted by

PRIYA DUTT M D

4NI15IS084

PUSHYA R

4NI15IS085

Under the guidance of

Mr. B N KIRAN

(Assistant Professor in the department of ISE)

Ms. MAYURA D TAPKIRE

(Assistant Professor in the department of ISE)

Ms.RAGHAVENDRA V

(Assistant Professor in the department of ISE)

Department of Information Science and Engineering

The National Institute of Engineering (Autonomous Institution)

MYSORE – 570008

THE NATIONAL INSTITUTE OF ENGINEERING

(AUTONOMOUS INSTITUTION)

Manandavadi Road, Mysore -570008



CERTIFICATE

This is to certify the project work entitled “2 PASS ASSEMBLER” is a bona fide work carried out by, **PRIYA DUTT M D** (4NI14IS084), **PUSHYA R** (4NI14IS085) for the 5th semester SS lab. It is certified that all corrections/suggestions indicated for internal assignment have been incorporated in this report.

Signature of the internal Guide

(B N KIRAN)

Signature of the internal Guide

(MAYURA TAPKIRE)

ACKNOWLEDGEMENT

The success and the final outcome of this project required a lot of guidance and assistance from many people and we are extremely fortunate to have got this all along the completion of our project work.

Our deepest thanks to the lecturers **Mr. B N Kiran, Mr. Raghavendra V, Mrs.Mayura D Tapkire** for guiding us and correcting our documents with attention and care. They have taken the keen interest to go through the project and help us make necessary correction as and when required.

Besides, we would like to thank the authorities of the information science Department for providing us a good environment and facilities to complete this project. Also we would like to take this opportunity to thank our college “The National Institute of Engineering”, Mysore for offering this subject, SS project.

This being our first design experience has proved to be extremely advantageous and will surely be of great help for the overall development of our future.

ABSTRACT

Computer Science Engineering has influenced today's technology in a great manner. We just cannot imagine our life without computers. They are an integral part of everyone's life.

As software engineers we make sure that the above paragraph is a reality. One of the major milestones in computer science is the evolution of programming languages. Using these languages we communicate with the computer and command it to do what we want.

When programming evolved it was an absolute nightmare. It used to consist of numbers which absolutely made no sense. Then came the era of the assembly language. Fairly understandable and robust programs were written. In our system software course we learn about a hypothetical computer called the Simplified Instruction Computer (SIC). We communicate with this computer using assembly language. For assembly language to work we need an assembler. In this project we have designed a 2 pass assembler that effectively communicates with SIC.

CONTENTS

1. INTRODUCTION

2. 2 PASS ASSEMBLER

- 2.1. Two Pass Assembler
- 2.2. Pass 1 of 2 Pass Assembler
- 2.3. Pass 2 of 2 Pass Assembler
- 2.4. Data Structures

3. ALGORITHMS

- 3.1. Pass One Algorithm
- 3.2. Pass Two Algorithm

4. PROGRAMS

- 4.1. Pass One Program
- 4.2. Pass Two Program

5. INPUT AND OUTPUTS

- 5.1. Input File
- 5.2. Operational Table
- 5.3. Output of pass 1
- 5.4. Symbol Table
- 5.5. Input Of Pass 2
- 5.6. Length Of Program
- 5.7. Output Of Pass2

6. CONCLUSION

7. LITERATURE SURVEY

Chapter 1

INTRODUCTION

System Software consists of a variety of programs that support the operations of a computer. It makes possible for the user to focus on an application or other problem to be solved, without needing to know the details of how the machine works internally. We may have used high level language like C,C++ ,using text editor to create a program and assembler to translate these programs into machine language. The resulting machine language program was loaded into memory and prepared for execution by loader and linker, and may have been tested using the debugger.

An *assembler* is a program that takes basic computer instructions and converts them into a pattern of bits that the computer processor can use to perform its basic operations. The fundamental functions like translating mnemonic operation codes to their machine language equivalents and assigning machine addresses to symbolic labels used by the programmer and finally producing the target machine code is done by the assembler. There are two types of assembler based on the number of passes it takes to process the source code and produce machine code.

One-pass assembler , that scans the program only once and creates the equivalent binary program. The assembler substitutes all of the symbolic instructions with machine code in one pass. Two-pass assembler , that scans the program twice to generate the object program.

Two-pass assembler : Here we discuss the design and implementation of two-pass assembler. In single pass assembler , generally generates the object code directly in memory for immediate execution. It parses the source code only once. If the assembler while on its way encounters the undefined label, it puts into a symbol table along with the address where the undefined symbol's value has to be placed when the symbol is found in future. Hence the one-pass assembler cannot resolve the forward references of data.

Two-pass assembler solves this dilemma by devoting one pass to exclusively resolve all (data/label) forward references and then generate object code with no hassels in the next pass.

Chapter 2

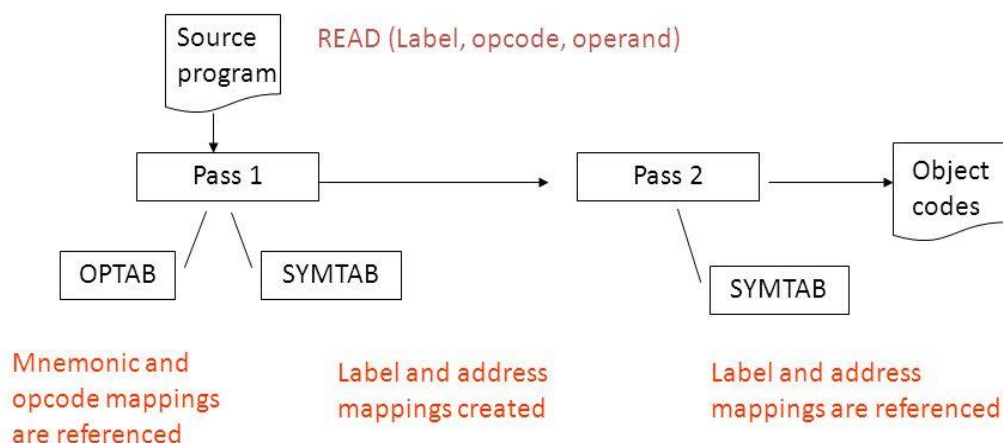
TWO-PASS ASSEMBLER

2.1. Two Pass Assembler

An assembler is a software that takes instructions as an input and translates it into machine level language which is understood by the computer. High level language is not understood by the computer. Hence assemblers translate this high level language into machine code so that programming can be done effectively. Assemblers are mainly classified into based on their working principle. Our main concern is in 2 pass assemblers.

A pass is said to be a onetime scan of the entire code. As the name suggests 2 pass assemblers scan the code twice. In the first pass the assembler checks for any syntax errors and updates the symbol table and in the second pass the object program is generated.

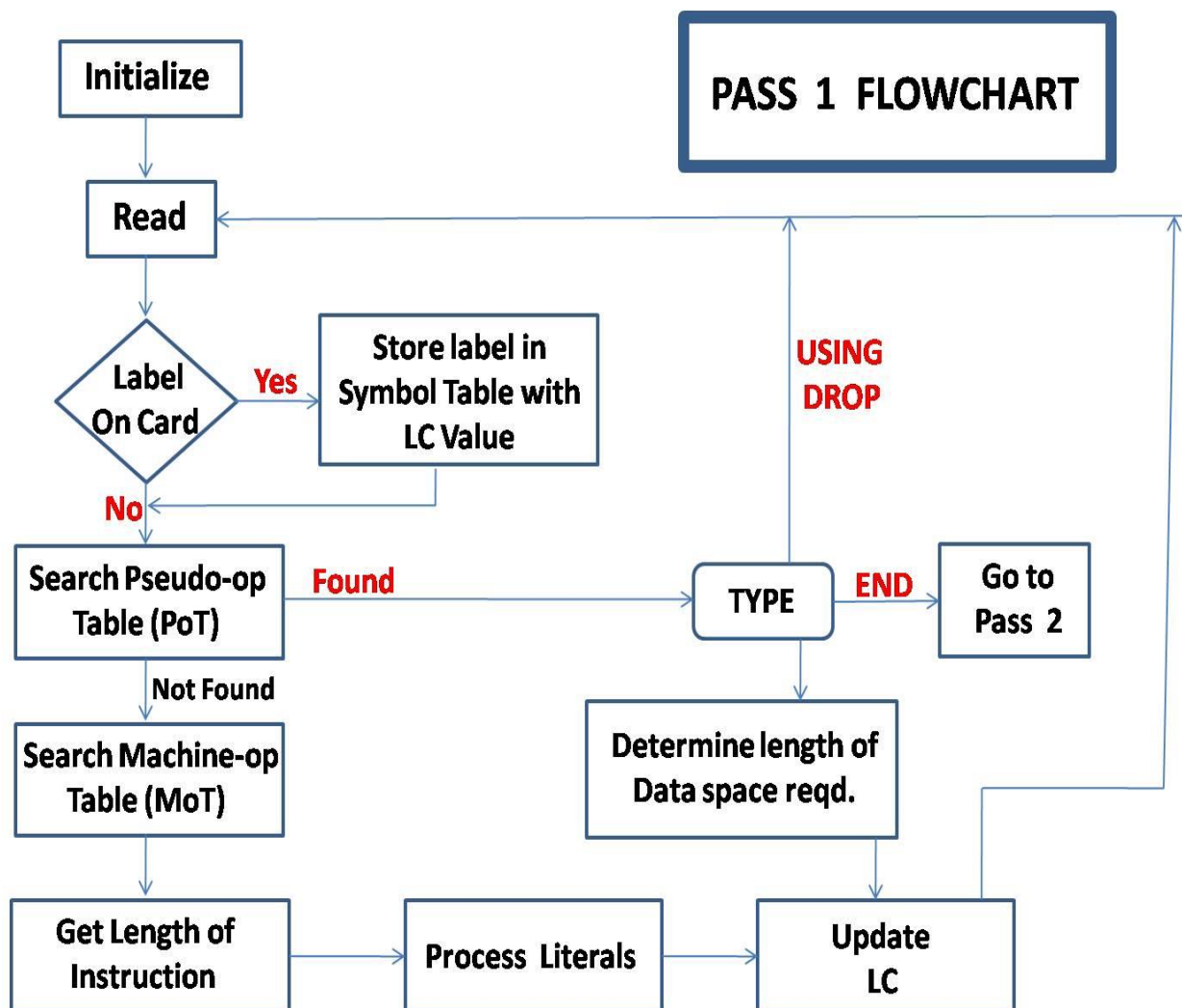
A Simple Two Pass Assembler Implementation



2.2 Pass One

As mentioned earlier in pass 1 the assembler checks for any errors. By errors we mean any syntactic errors. If an error is generated the assembly of the program terminates by flagging an error. The variable names that are declared in the variable segment are entered in the symbol table. The output of the first pass is an intermediate file. In this intermediate file all the instructions are assigned an address. Assembler directives are processed. Here it checks the syntactic errors. The goals of pass 1 are mentioned below ,

- ❖ Assign addresses to all statements in the program.
- ❖ Save the values assigned to all labels for use in Pass 2.
- ❖ Process some assembler directives.



2.3 Pass Two

In pass 2, the code is assembled into object program. The input of pass 2 is the output of pass 1. Instruction is fetched from the intermediate file and the object code is generated for each instruction. The opcode for each mnemonic is available in the object table and the address of the variable is available in the symbol table. Hence in pass 2 the following operations are going to take place,

- ❖ Assemble instructions.
- ❖ Generate data values defined by BYTE, WORD, etc.
- ❖ Process the assembler directives not done in Pass 1.
- ❖ Write the object program and the assembly listing

2.4 .Data Structures

Three simple data structures are used. They are the Symbol table, Operation code table and the location counter.

OPTAB: This table consists of the representations of the various mnemonics that are used in the source code. This table is mainly used to validate the correctness of the mnemonic used and to fetch the operation code of the mnemonic used.

SYMTAB: This table consists of the name and the address of the various variables used in the source code. In the second pass the address of each variable is retrieved in order to assemble the object program.

LOCCTR: This data structure is used to point at the succeeding instruction of the current instruction that is being processed.

Chapter 3

ALGORITHMS

3.1 Pass 1 Algorithm

Start

The input file assembly language code is divided into label ,opcode and operand

Read first input line

If OPCODE = `START` then

Save # [OPERAND] as starting address

Initialize LOCCTR to starting address

write line to intermediate file

read next input line

end start loop

else

initialize LOCCTR to 0

while OPCODE = `END` do

if this is not a comment line i.e the line starting with semicolon in input file then

if there is a symbol in the LABEL field then

search SYMTAB for LABEL

if found then

set error flag if duplication of symbol

else

add to symbol table which contain locctr ,value

end if there is symbol loop

search OPTAB for OPCODE

if found the

if OPCODE = `WORD` then

add 3 to LOCCTR

else if OPCODE = `RESW` then

add 3* #[OPERAND] to LOCCTR

else if OPCODE = `RESB` then

add # [OPERAND] to LOCCTR

else if OPCODE = `BYTE` then

find length of constant in bytes

add length to LOCCTR

end if it is byte

else

addlocctr =locctr +3 i.e is length of instruction

else

set error flag (invalid operation code)

end if not comment

write line to intermediate file

read next input line

write last line to intermediate file

save (LOCCTR – starting address) as program length

end

3.2 Pass 2 Algorithm

Start

The input file assembly language code is divided into label ,opcode and operand

Read first input line

If OPCODE = `START` then

Save # [OPERAND] as starting address

Initialize LOCCTR to starting address

write line to intermediate file

read next input line

end start loop

else

initialize LOCCTR to 0

while OPCODE = `END` do

if this is not a comment line i.e the line starting with semicolon in input file then

if there is a symbol in the LABEL field then

search SYMTAB for LABEL

if found then

set error flag if duplication of symbol

else

add to symbol table which contain locctr ,value

end if there is symbol loop

search OPTAB for OPCODE

if found the

if OPCODE = `WORD` then

add 3 to LOCCTR

else if OPCODE = `RESW` then

add 3* #[OPERAND] to LOCCTR

else if OPCODE = `RESB` then

add # [OPERAND] to LOCCTR

else if OPCODE = `BYTE` then

find length of constant in bytes

add length to LOCCTR

end if it is byte

else

addlocctr =locctr +3 i.e is length of instruction

else

set error flag (invalid operation code)

end if not comment

write line to intermediate file

read next input line

write last line to intermediate file

save (LOCCTR – starting address) as program length

end

Chapter 4

PROGRAMS

4.1 PROGRAM TO IMPLEMENT PASS 1

```
main.c X main.c X
1  #include<stdio.h>
2  #include<conio.h>
3  #include<string.h>
4  void main()
5  {
6  FILE *f1,*f2,*f3,*f4,*f5,*f6;
7  int lc,sa,l,op1,o,len;
8  char m1[20],la[20],op[20],otp[20];
9  //clrscr();
10 f1=fopen("input.txt","r");
11 f3=fopen("symtab.txt","w");
12 f5=fopen("inter.txt","w");
13 fscanf(f1,"%s %s %d",la,m1,&op1);
14 if(strcmp(m1,"START")==0)
15 {
16     sa=op1;
17     lc=sa;
18     fprintf(f5,"\t%s\t%s\t%d\n",la,m1,op1);
19 }
20 else
21     lc=0;
22 fscanf(f1,"%s %s",la,m1);
23 while(!feof(f1))
24 {
25     fscanf(f1,"%s",op);
26     fprintf(f5,"\n%d\t%s\t%s\t%s\n",lc,la,m1,op);
27     if(strcmp(la,"-")!=0)
28     {
29         fprintf(f3,"\n%d\t%s\n",lc,la);
30     }
31     f2=fopen("optab.txt","r");
32     fscanf(f2,"%s %d",otp,&o);
33     while(!feof(f2))
34     {
35         if(strcmp(m1,otp)==0)
36         {
```

```
69         if(strcmp(m1,"END")==0)
70         {
71             fprintf(f5,"Program length =\n%d",lc-sa);
72         }
73         fclose(f1);
74         fclose(f3);
75     fclose(f5); // getch();
76     }
77
```

4.2. PROGRAM TO IMPLEMENT PASS 2

```
1  #include<stdio.h>
2  #include<string.h>
3  #include<ctype.h>
4  main()
5  {
6  FILE *f1,*f2,*f3,*f4;
7  int opl[10],txtlen,txtlen1,i,j=0,len;
8  char add[5],symadd[5],op[5],start[10],temp[30],line[20],label[20],mne[10],operand[10],symtab[10],opmne[10];
9  f1=fopen("input.txt","r");
10 f3=fopen("length.txt","r");
11 f2=fopen("optab.txt","r");
12 f4=fopen("symbol.txt","r");
13 fscanf(f1,"%s%s%s%s",add,label,mne,operand);
14 if(strcmp(mne,"START")==0)
15 {
16 strcpy(start,operand);
17 fscanf(f3,"%d",&len);
18 }
19 printf("H^%s^%s^%d\nT^00%s^",label,start,len,start);
20 fscanf(f1,"%s%s%s%s",add,label,mne,operand);
21 while(strcmp(mne,"END")!=0)
22 {
23 fscanf(f2,"%s%s",opmne,op);
24 while(!feof(f2))
25 {
26 if(strcmp(mne,opmne)==0)
27 {
28 fclose(f2);
29 fscanf(f4,"%s%s",symadd,symtab);
30 while(!feof(f4))
31 {
32 if(strcmp(operand,symtab)==0)
```

```

33 {
34     printf("%s%s^", op, symadd);
35     break;
36 }
37 else
38     fscanf(f4, "%s%s", symadd, symtab);
39 }
40 break;
41 }
42 else
43     fscanf(f2, "%s%s", opmne, op);
44 }
45 if ((strcmp(mne, "BYTE")==0) || (strcmp(mne, "WORD")==0))
46 {
47     if (strcmp(mne, "WORD")==0)
48         printf("0000%s^", operand);
49     else
50     {
51         len=strlen(operand);
52         for(i=2; i<len; i++)
53         {
54             printf("%d", operand[i]);
55         }
56         printf("^");
57     }
58 }
59 fscanf(f1, "%s%s%s%s", add, label, mne, operand);
60 f2=fopen("optab.txt", "r");
61 fseek(ftab, SEEK_SET, 0);
62 }
63 printf("\nE^00%s", start);

```

```

57 }
58 }
59 fscanf(f1, "%s%s%s%s", add, label, mne, operand);
60 f2=fopen("optab.txt", "r");
61 fseek(ftab, SEEK_SET, 0);
62 }
63 printf("\nE^00%s", start);
64 fclose(f1);
65 fclose(f2);
66 fclose(f4);
67 fclose(f3);
68 }
69 }
70 }
71 }

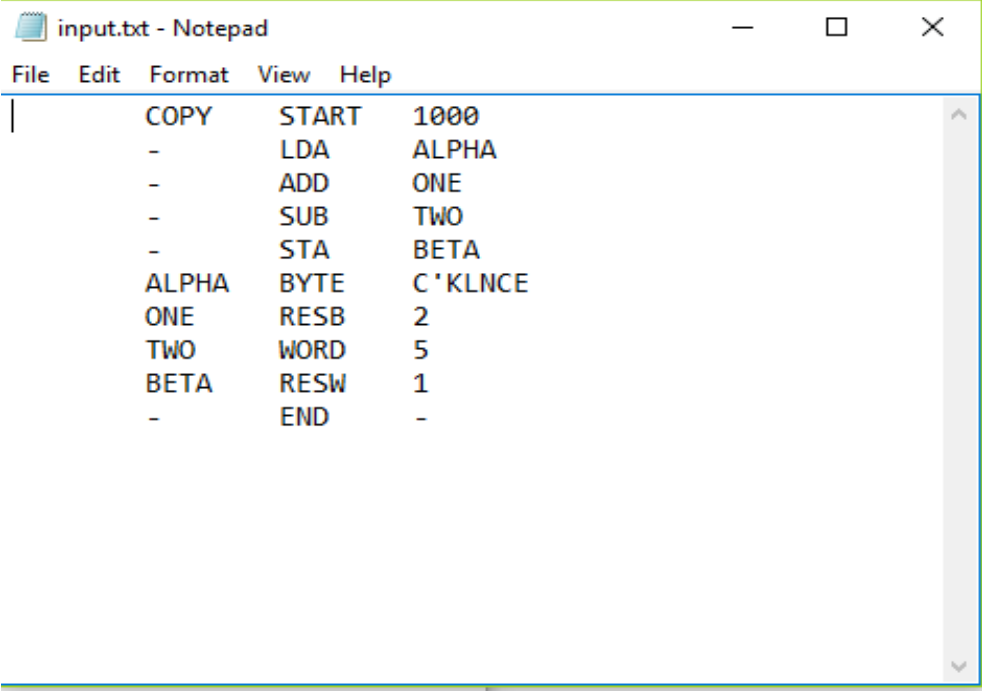
```

Chapter 5

Input and Output

Input for pass 1

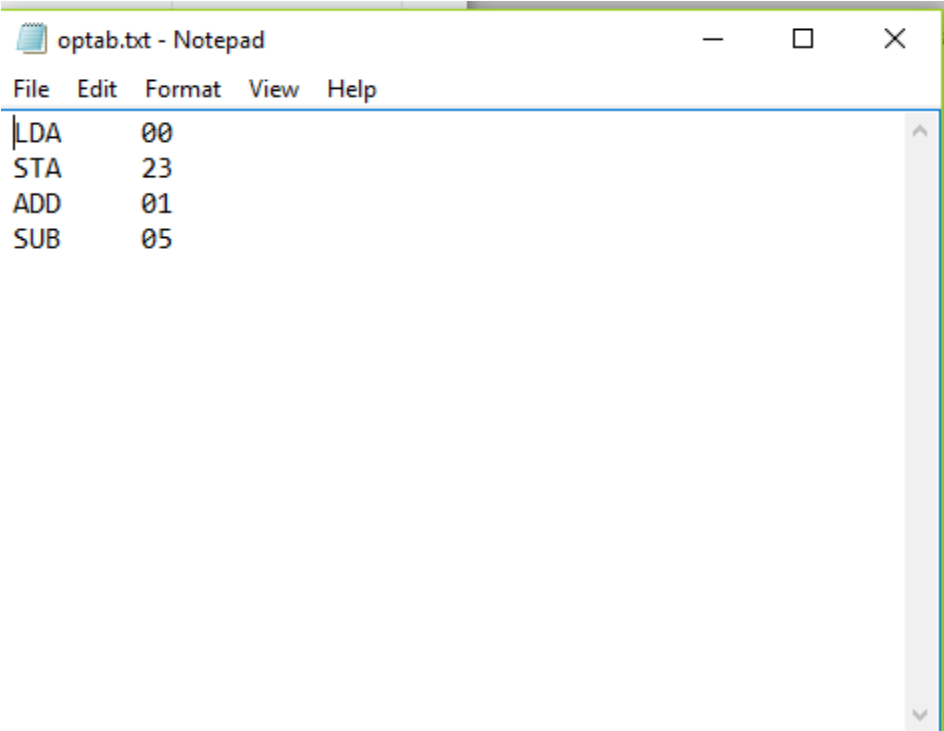
5.1 Input File for Pass 1



A screenshot of a Notepad window titled "input.txt - Notepad". The window contains the following assembly code:

```
COPY      START  1000
-         LDA    ALPHA
-         ADD    ONE
-         SUB    TWO
-         STA    BETA
ALPHA     BYTE   C'KLNCE
ONE       RESB   2
TWO       WORD   5
BETA      RESW   1
-         END    -
```

5.2 Operational Tab(OPTAB)

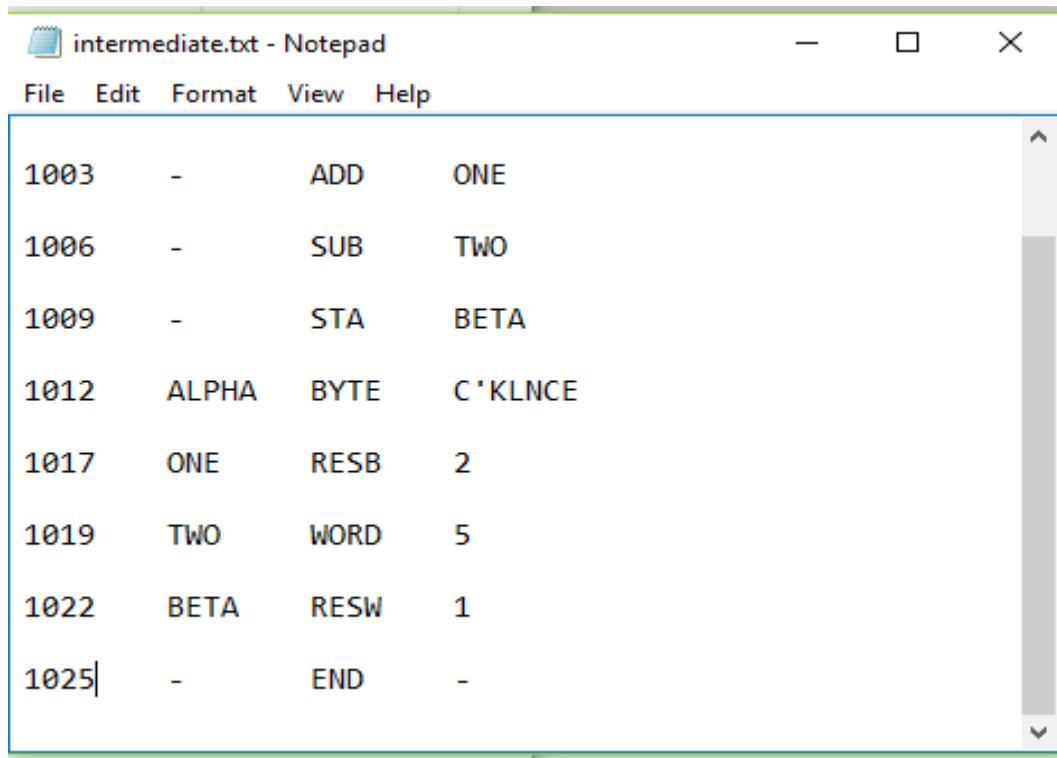


A screenshot of a Notepad window titled "optab.txt - Notepad". The window contains the following Operational Tab (OPTAB) data:

```
LDA      00
STA      23
ADD      01
SUB      05
```

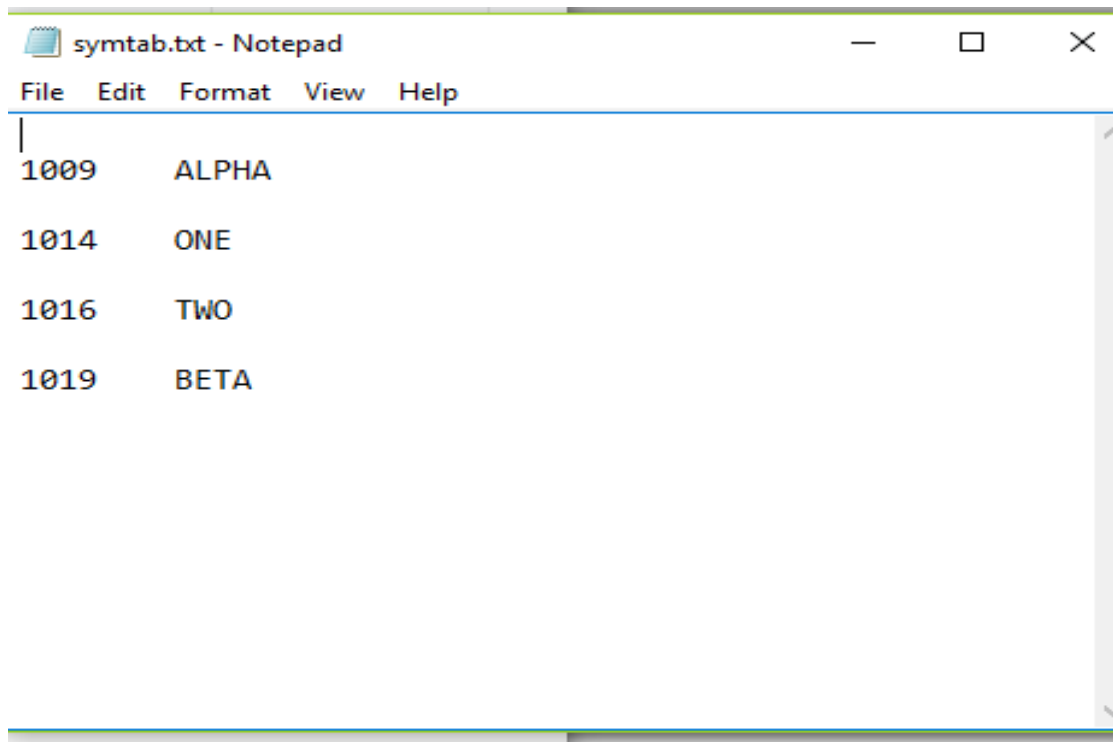

Output of Pass 1

5.3 Output of Pass 1



```
intermediate.txt - Notepad
File Edit Format View Help
1003      -      ADD      ONE
1006      -      SUB      TWO
1009      -      STA      BETA
1012  ALPHA  BYTE  C'KLNCE
1017  ONE    RESB   2
1019  TWO    WORD  5
1022  BETA   RESW   1
1025|  -      END      -
```

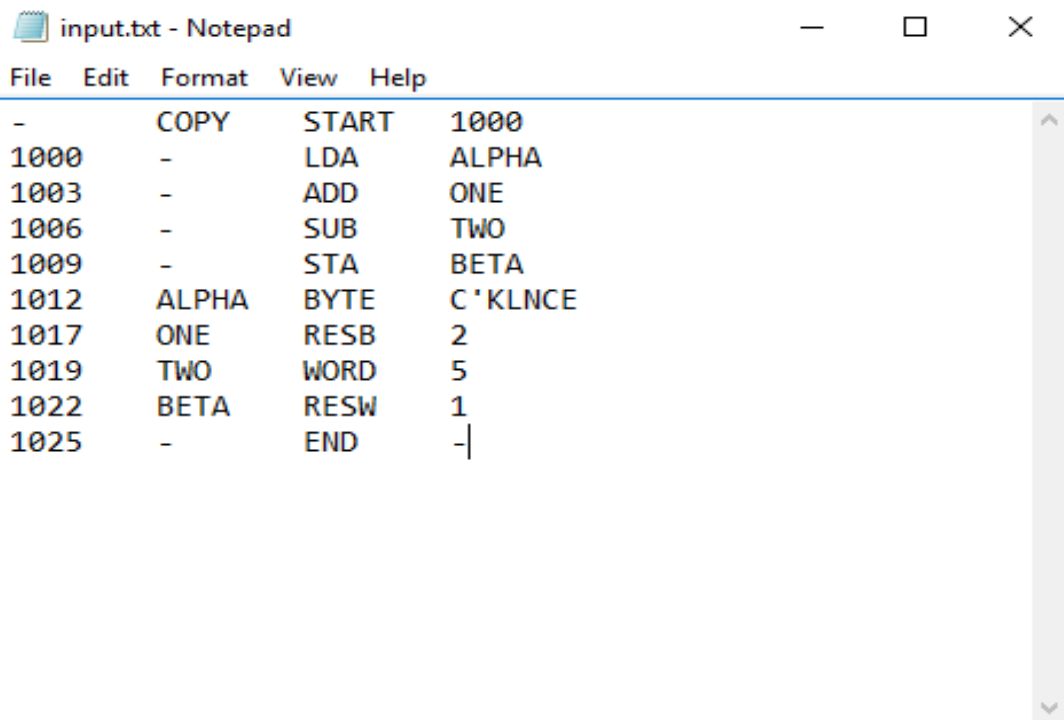
5.4 Symbol Table (SYMTAB)



```
symtab.txt - Notepad
File Edit Format View Help
|
1009  ALPHA
1014  ONE
1016  TWO
1019  BETA
```

Input for pass 2

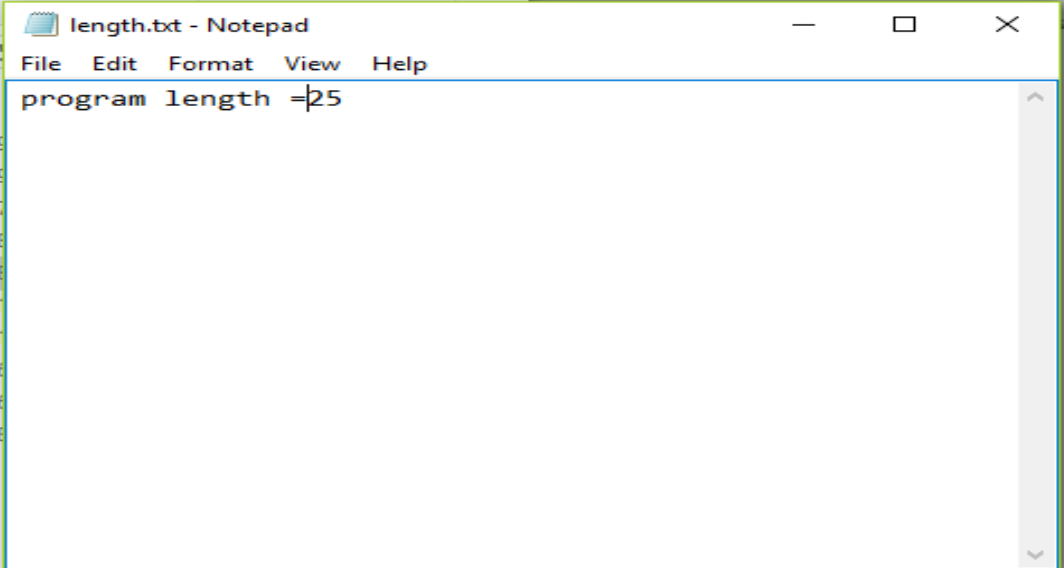
5.5 Input of Pass 2



A screenshot of a Notepad window titled "input.txt - Notepad". The window contains the following assembly code:

```
-      COPY      START  1000
1000   -          LDA   ALPHA
1003   -          ADD   ONE
1006   -          SUB   TWO
1009   -          STA   BETA
1012   ALPHA     BYTE   C'KLNCE
1017   ONE       RESB   2
1019   TWO       WORD   5
1022   BETA      RESW   1
1025   -          END   -
```

5.6 Length of Program



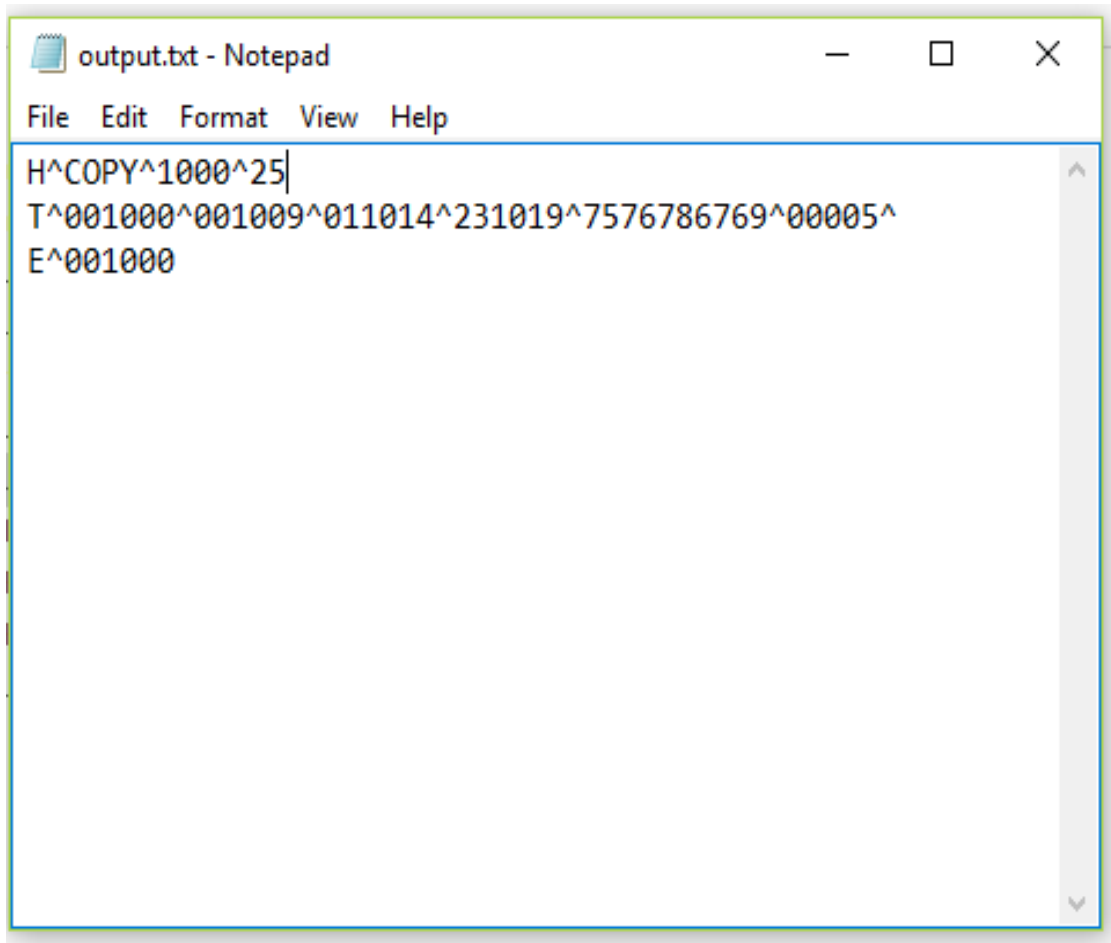
A screenshot of a Notepad window titled "length.txt - Notepad". The window contains the following text:

```
program length =25
```

Note: Operational Table (OPTAB) and Symbol Table (SYMTAB) are also used as Inputs for Pass 2.

Output of Pass 2

5.7 Output of Pass 2 (Object Program)



```
output.txt - Notepad
File Edit Format View Help
H^COPY^1000^25|
T^001000^001009^011014^231019^7576786769^00005^
E^001000
```

CONCLUSION

Finally I want to conclude that our team got good hands on experience on implementing an assembler. This helped our team to gain a lot of knowledge about implementing an assembler and moreover the design of robust soft wares.

We hope that this knowledge helps us in our future. We thank our faculty members in extending our support in order to make this project a huge success.

LITERATURE SURVEY

References:

- System Software: an introduction to systems programming, Leland L Beck, Manjula D, 3rd Edition, Pearson Education Limited, 2013
- System Programming and Operating Systems, D M Dhamdhere, TATA McGraw Hill, 2nd Edition, 2011.

Ebook:

- web.thu.edu.tw/ctyang/ss

Links:

- <https://www.mooc-list.com/tags/system-software>
- <http://web.thu.edu.tw/ctyang/www/course.php>

