PART B
EXPERIMENT NUMBER 7

Aim: To design and implement the first pass of a two-pass assembler for IBM 360/370 Processor.

(PART B: TO BE COMPLETED BY STUDENTS)

(Students must submit the soft copy as per the following segments within two hours of the practical. The soft copy must be uploaded at the end of the practical)

Roll No. 50	**Name:** AMEY THAKUR
Class: Comps TE B	**Batch:** B3
Date of Experiment: 30/04/2021	**Date of Submission:** 30/04/2021
Grade:	

B.1 Software Code written by a student:

(Paste your code completed during the 2 hours of practice in the lab here)

- **SPCC-7.C**

```
#include<stdio.h>
#include<string.h>
#include <stdlib.h>

int main()
{
FILE *f1,*f2,*f3,*f4;
int lc,sa,l,op1,o,len;
char m1[20],la[20],op[20],otp[20];
f1=fopen("INPUT.txt","r");
f3=fopen("SYMTAB.txt","w");
fscanf(f1,"%s %s %d",la,m1,&op1);
if(strcmp(m1,"START")==0)
{
 sa=op1;
 lc=sa;
 printf("\t%s\t%s\t%d\n",la,m1,op1);
 }
 else
 lc=0;
fscanf(f1,"%s %s",la,m1);
```

```
while(!feof(f1))
{
 fscanf(f1,"%s",op);
 printf("\n%d\t%s\t%s\t%s\n",lc,la,m1,op);
 if(strcmp(la,"-")!=0)
 {
 fprintf(f3,"\n%d\t%s\n",lc,la);
 }
 f2=fopen("OPTAB.txt","r");
 fscanf(f2,"%s %d",otp,&o);
 while(!feof(f2))
 {
  if(strcmp(m1,otp)==0)
  {
    lc=lc+3;
    break;
  }
  fscanf(f2,"%s %d",otp,&o);
  }
  fclose(f2);
  if(strcmp(m1,"WORD")==0)

    {
   lc=lc+3;
   }
   else if(strcmp(m1,"RESW")==0)
   {
    op1=atoi(op);
    lc=lc+(3*op1);
    }
    else if(strcmp(m1,"BYTE")==0)
    {
    if(op[0]=='X')
      lc=lc+1;
      else
      {
      len=strlen(op)-2;
      lc=lc+len;}
    }
    else if(strcmp(m1,"RESB")==0)
    {
     op1=atoi(op);
     lc=lc+op1;
```

```
 }
 fscanf(f1,"%s%s",la,m1);
 }
 if(strcmp(m1,"END")==0)
 {
 printf("Program length =\n%d",lc-sa);
 }
 fclose(f1);
 fclose(f3);
 return 0;
 }
```

- **INPUT.TXT**

```
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     _
```

- **OPTAB.TXT**

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

- **SYMTAB.TXT**

```
1012   ALPHA

1017   ONE

1019   TWO

1022   BETA

1025   _
```

B.2 Input and Output:

```
C:\Users\ameyt\Desktop\SPCC-7>GCC SPCC-7.C

C:\Users\ameyt\Desktop\SPCC-7>a.exe
        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     _
Program length =
25
```

B.3 Observations and learning:

(Students are expected to comment on the output obtained with clear observations and learning for each task/ subpart assigned)

We have learnt about the two passes of the assembler and implemented the first pass.

B.4 Conclusion:

(Students must write the conclusion as per the attainment of individual outcome listed above and learning/observation noted in section B.3)

Hence, we have successfully implemented the program for the first pass of a two-pass assembler.

B.5 Question of Curiosity

(To be answered by a student based on the practical performed and learning/ observations)

A. Define Data Structures

Ans:

1. **Mnemonic Operation Table**

   This table indicates the symbolic mnemonic for each instruction and its length.

6- bytes per entry

Mnemonic -codes (4 bytes) characters	Binary op-codes (1 byte) hexadecimal	Instruction length (2-bits) binary	Instruction format (3-bits) binary	Not used in this design (3 bits)
"Abbb"	5A	10	001	
"AHbb"	4A	10	001	
"ALbb"	5E	10	001	
"ALRb"	1E	01	000	
.....	.....	.....	.....	

b : bank space

Instruction Length

01 =1 half words=2 bytes

10 =2 half words=4 bytes

11 =3 half words=6 bytes

Instruction Format

000= RR

001=RX

010=RS

011=SI

100=SS

2. **Symbol Table**

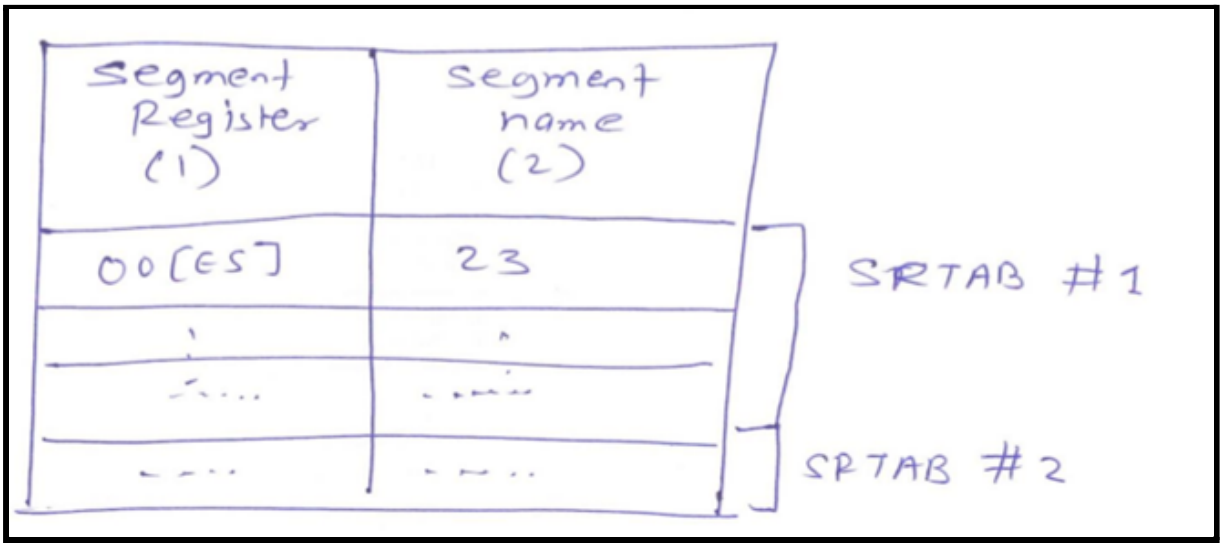
It stores each label along with its value.

14 BYTES PER ENTRY

Symbol (8 bytes) characters	Value (4 bytes) hexadecimal	Length (1 byte) hexadecimal	Relocation (1-byte) character
"JHONbbbb"	0000	01	"R"
"FOURbbbb"	00OC	04	"R"
"FIVEbbbb"	0010	04	"R"
"TEMPbbbb"	0014	04	"R"

3. **Segment Register Table**

It stores information about the segment name and segment register.



4. **Forward Reference Table**

It stores information about forwarding references.

Pointer (2)	SRTAB # (1)	Instruction Address (2)	Usage Code (1)	Source statement # (2)

5. **Cross Reference Table**

It lists out all references to a symbol in ascending order of statements.

Pointer to next entry (2)	Source statement # (2)

B. Comment on the Forward Reference Problem and Remedy.

Ans:

- ➔ In an assembly language program, we can use symbols which are the names associated with data or instructions.
- ➔ The symbols may be referred to before they are defined. This is called a forward reference.
- ➔ One approach to solve this problem is to have two passes over the source program. So the first pass just defines the symbols and the second pass finds the addresses.