PART A

(PART A : TO BE REFFERED BY STUDENTS)

Experiment No. 2 (B)

Aim : To design and implement second pass of a two pass assembler for IBM 360/370 Processor

Objective: Develop a program to implement second pass:

- **a.** To generate Base table
- **b.** To generate machine code

Outcome: Students are able to design and implement pass 2 of two pass assembler.

Theory:

Pass 2: Purpose - To generate object program

1) Look up value of symbols (STGET)
2) Generate instruction (MOTGET2)
3) Generate data (for DC, DS)
4) Process pseudo ops (POT, GET2)

Data Structures:

1) Copy of source program from Pass1
2) Location counter
3) MOT which gives the length, mnemonic format opcode
4) POT which gives mnemonic & action to be taken
5) Symbol table from Pass1
6) Base table which indicates the register to be used or base register
7) A work space INST to hold the instruction & its parts
8) A work space PRINT LINE, to produce printed listing
9) A work space PUNCH CARD for converting instruction into format needed by loader
10) An output deck of assembled instructions needed by loader.

Format of database:

Base Table:

Assembler uses this table to generate proper base register reference in machine instructions and to compute offset. Then the offset is calculated as:

offset= value of symbol from ST - contents of base register

4 bytes per entry

	Availability indicator (1-byte) characters	Designated relative address contents of base register (3 bytes= 24 bits address) hexadecimal
1	"N"	
	"N"	
2...	....	....
	"Y"	

Y : Register specified in USING pseudo-op

N: Register never specified in USING pseudo-op or made unavailable by DROP pseudo-op.

Let us consider the same example as experiment no. 1 and the base table after statement 2:

Base register	Contents
15	0

After statement 7:

Base register	Contents
15	10

Code after pass2:

stmt no	Relative address	Statement		
3	0		A	1, 12 (0,15)
4	4		A	2, 16(0,15)
6	8		A	3, 10(0,15)
8	12		4	
9	16		5	
10		-		

Algorithm:

1. Initialize the location counter as: LC=0
2. Read the statement from source program
3. Examine the op-code field: If match found in MOT then

 a. From the MOT entry determine the length field i.e. L=length, binary op-code and format of the instruction.

 Different instruction format requires different processing as described below:

 1. **RR Instruction : (Register to Register)**
 Both of the register specification fields are evaluated and placed into second byte of RR instruction
 2. **RX Instruction : (Register to Index)**
 Both of the register and index fields are evaluated and processed similar to RR instruction. The storage address operand is evaluated to generate effective address

(EA). The BT is examined to find the base register. Then the displacement is determined as:

D=EA- Contents of base register.

The other instruction formats are processed in similar manner to RR and RX.

b. Finally the base register and displacement specification are assembled in third and fourth bytes of instruction.
c. The current value of location counter is incremented by length of instruction.

4. If match found in POT then
   a. If it is EQU pseudo-op then EQU card is printed in the listings.
   b. If it is USING pseudo-op then the corresponding BT entry is marked as available.
   c. If it is DROP pseudo-op then the corresponding BT entry is marked as unavailable.
   d. If it is DS or DC pseudo-op then various conversions are done depending on the data type and symbols are evaluated. Location counter is updated by length of data.
   e. END pseudo-op indicates end of source program and then pass2 is terminated. Before that if any literals are remaining then the code is generated for them.
5. After assembling the instruction it is put in the format required by loader.
6. Finally a listing is printed which consist of copy of source card, its storage location and hexadecimal representation.
7. Go to step 2.

PART B

(PART B : TO BE COMPLETED BY STUDENTS)

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

Roll. No. A41	Name:Rohit Govardhane
Class: Computer Engineering A	Batch: A2
Date of Experiment:	Date of Submission:
Grade:	

B.1 Software Code written by student:

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

INTERMED.DAT

```
COPY    START   2000
2000    **    LDA    FIVE
2003    **    STA    ALPHA
2006    **    LDCH    CHARZ
2009    **    STCH    C1
2012    ALPHA    RESW    1
2015    FIVE    WORD    5
2018    CHARZ    BYTE    C'EOF'
2019    C1    RESB    1
2020    **    END    **
```

SYMTAB.DAT

```
ALPHA    2012
FIVE    2015
CHARZ    2018
C1    2019
```

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
void main()
{
  char a[10],ad[10],label[10],opcode[10],operand[10],symbol[10],ch;  int
st,diff,i,address,add,len,actual_len,finaddr,prevaddr,j=0;
  char mnemonic[15][15]={"LDA","STA","LDCH","STCH"};
  char code[15][15]={"33","44","53","57"};
  FILE *fp1,*fp2,*fp3,*fp4;
  clrscr();
  fp1=fopen("ASSMLIST.DAT","w");
  fp2=fopen("SYMTAB.DAT","r");
  fp3=fopen("INTERMED.DAT","r");
  fp4=fopen("OBJCODE.DAT","w");
  fscanf(fp3,"%s%s%s",label,opcode,operand);
```

```
while(strcmp(opcode,"END")!=0)
{
 prevaddr=address;
 fscanf(fp3,"%d%s%s%s",&address,label,opcode,operand);
}
finaddr=address;
fclose(fp3);
fp3=fopen("INTERMED.DAT","r");

fscanf(fp3,"%s%s%s",label,opcode,operand);
if(strcmp(opcode,"START")==0)
{
 fprintf(fp1,"\t%s\t%s\t%s\n",label,opcode,operand);
 fprintf(fp4,"H^%s^00%s^00%d\n",label,operand,finaddr);


 fscanf(fp3,"%d%s%s%s",&address,label,opcode,operand);
 st=address;
 diff=prevaddr-st;
 fprintf(fp4,"T^00%d^%d",address,diff);
}
while(strcmp(opcode,"END")!=0)
{
 if(strcmp(opcode,"BYTE")==0)
 {
  fprintf(fp1,"%d\t%s\t%s\t%s\t",address,label,opcode,operand);
  len=strlen(operand);
  actual_len=len-3;
  fprintf(fp4,"^");
  for(i=2;i<(actual_len+2);i++)
  {
   itoa(operand[i],ad,16);
   fprintf(fp1,"%s",ad);
   fprintf(fp4,"%s",ad);
  }
  fprintf(fp1,"\n");
 }
 else if(strcmp(opcode,"WORD")==0)
 {
  len=strlen(operand);
  itoa(atoi(operand),a,10);
  fprintf(fp1,"%d\t%s\t%s\t%s\t00000%s\n",address,label,opcode,operand,a);
  fprintf(fp4,"^00000%s",a);
 }
 else if((strcmp(opcode,"RESB")==0)||(strcmp(opcode,"RESW")==0))
  fprintf(fp1,"%d\t%s\t%s\t%s\n",address,label,opcode,operand);
 else
 {
  while(strcmp(opcode,mnemonic[j])!=0)
   j++;
  if(strcmp(operand,"COPY")==0)

fprintf(fp1,"%d\t%s\t%s\t%s\t%s0000\n",address,label,opcode,operand,code[j]);
  else
  {
   rewind(fp2);
```

```
      fscanf(fp2,"%s%d",symbol,&add);
       while(strcmp(operand,symbol)!=0)
        fscanf(fp2,"%s%d",symbol,&add);

 fprintf(fp1,"%d\t%s\t%s\t%s\t%s%d\n",address,label,opcode,operand,code[j],add
);
      fprintf(fp4,"^%s%d",code[j],add);
     }
    }
    fscanf(fp3,"%d%s%s%s",&address,label,opcode,operand);
   }
   fprintf(fp1,"%d\t%s\t%s\t%s\n",address,label,opcode,operand);
   fprintf(fp4,"\nE^00%d",st);
   printf("\n Intermediate file is converted into object code");
   fcloseall();

   printf("\n\nThe contents of Intermediate file:\n\n\t");
   fp3=fopen("INTERMED.DAT","r");
   ch=fgetc(fp3);
   while(ch!=EOF)
   {
    printf("%c",ch);
    ch=fgetc(fp3);
   }
   printf("\n\nThe contents of Symbol Table :\n\n");
   fp2=fopen("SYMTAB.DAT","r");
   ch=fgetc(fp2);
   while(ch!=EOF)
   {
    printf("%c",ch);
    ch=fgetc(fp2);
   }
   printf("\n\nThe contents of Output file :\n\n");
   fp1=fopen("ASSMLIST.DAT","r");
   ch=fgetc(fp1);
   while(ch!=EOF)
   {
    printf("%c",ch);
    ch=fgetc(fp1);
   }
   printf("\n\nThe contents of Object code file :\n\n");
   fp4=fopen("OBJCODE.DAT","r");
   ch=fgetc(fp4);
   while(ch!=EOF)
   {
    printf("%c",ch);
    ch=fgetc(fp4);
   }
   fcloseall();
   getch();
}
```

B.2 Input and Output:

```
ALPHA    2012
FIVE    2015
CHARZ    2018
C1    2019

The contents of Output file :

        COPY    START   2000
2000    **      LDA     FIVE    332015
2003    **      STA     ALPHA   442012
2006    **      LDCH    CHARZ   532018
2009    **      STCH    C1      572019
2012    ALPHA   RESW    1
2015    FIVE    WORD    5       000005
2018    CHARZ   BYTE    C'EOF'  454f46
2019    C1      RESB    1
2020    **      END     **


The contents of Object code file :

H^COPY^002000^002020
T^002000^19^332015^442012^532018^572019^000005^454f46
E^002000
```

B.3 Observations and learning:

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

From this practical, we have successfully able to design and implement second pass of a two pass assembler for IBM 360/370 Processor.

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)

In this experiment we design and implement first pass of a two pass assembler for IBM 360/370 Processor.

B.5 Question of Curiosity

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

A. Give Example of Working of Two Pass Assembler:

Ans:

A two-pass assembler processes the source code in two passes:

- First Pass: It reads the entire source code, generates a symbol table, and performs preliminary tasks.
- Second Pass: It uses the symbol table to produce the machine or object code.

Example:

Consider the following assembly code:

css

CopyEdit

```
START 1000
MOVER AREG, X
ADD BREG, M
MOVEM C, VAR
STOP
VAR RESW 1
X RESW 1
M RESW 1
```

Pass 1:

In the first pass, the assembler reads the entire source code to create a symbol table:

Label	Address
START	1000
MOVER	1001
AREG	1002
X	1003
ADD	1004
BREG	1005
M	1006
MOVEM	1007
C	1008
VAR	1009
STOP	1010

Pass 2:

In the second pass, the assembler generates the machine code using the symbol table:

Memory Address	Machine Code
1000	(no machine code)
1001	04 02 1003 // MOVER AREG, X
1004	05 03 1006 // ADD BREG, M
1007	06 04 1008 // MOVEM C, VAR
1010	00 // STOP

This machine code can now be loaded into memory for execution.

B. Mention the Advantage of assemblers with Multiple Passes?

Ans:

1. Symbol Resolution: Multiple passes allow the assembler to resolve symbols and addresses. The first pass generates a symbol table, and the second pass uses it to assign final addresses to labels.
2. Forward Referencing: Assemblers with multiple passes can handle forward references, where a label is used before being defined. The first pass identifies all labels, and the second pass resolves the addresses.
3. Optimization: Multiple passes provide opportunities for optimization. The assembler can optimize code in the second pass, such as code rearrangement or size optimization, based on the information gathered in the first pass.
4. Error Detection: The first pass enables syntax checking and error detection. This helps generate meaningful error messages, streamlining the debugging process.