EXPERIMENT NO 1

Date of performance :
Date of submission :

Aim: To implement a two pass assembler.
Software Used: c,c++,java language
Theory:

If the source program is in assembly language of a machine, the object is in the machine language of the same machine and translator is executing on the same machine. The assemblere generates 2 files. The first file called the object file and is given .obj extension. The object file contains the binary code for instruction and information about object instruction. The second file generated by assembler is called list file with .lst extension. The list file contains assembly language statements, binary code for each instruction and offset for each instruction.

There are two phases of assembler design:
1. Analysis phase
2. Synthesis phase

The task performed by the two phases are:-
Analysis phase:

1. isolated label, mnemonic opcode,operands and comment of statement.
2. check validity of mnemonic opcode by consulting MOT.
3. check the number of operands required for an instruction by consulting MOT.
4. process the labels and symbols appropriately and fill ST.
5. update LC appropriately by consulting MOT for the length of insturction.
6. ignore comments.
7. takeb proper actions for pseudo opcodes by consulting POT.

Synthesis phase:

1. obtain machine opcode consulting to MOT corresponding to mnemonic.
2. Fill in address for symbols or labels at appropriate places by consulting ST.
3. write the above information in output object file.

1)ALGORITHM:
Pass1: Purpose- define symbols and literals

1. Determine length of machine instructions (MOTGET1)
2. Keep track of Location Counter (LC)
3. Remember values of symbols until Pass2 (STSTO)
4. Process some pseudo ops(POTGET1)
5. Remember literals (LITSTO)

Pass2: Purpose- generate object program

1. Look up value of symbols(STGET)
2. Generate instructions(MOTGET2)
3. Generate Data
4. Process pseudo ops(POTGET2)

2) PROBLEM SPECIFY

Source program			FIRST PASS Relative Address	Mnemonic instruction		SECOND PASS Relative Address	Mnemonic instruction	
JOHN	START	0						
	USING	*,15						
	L	1,FIVE	0	L	1,-(0,15)	0	L	1,16(0,15)
	A	1,FOUR	4	A	1,-(0,15)	4	A	1,12(0,15)

	ST	1,TEMP	8	ST	1,-(0,15)	8	ST	1,20(0,15)
FOUR	DC	F'4'	12	4		12	4	
FIVE	DC	F'5'	16	5		16	5	
TEMP	DS	1F	20			20		
	END							

3) FORMAT OF DATABASE

1) PSEUDO OPCODE TABLE(POT) FOR PASS1 AND PASS2

Mnemonic Op-code (4 byte) (character)	Binary Op-code (1-byte) (Hexadecimal)	Instruction Length (2-bite) (binary)	Instruction Format (3 bit) (binary)	Not Use in this design (3 bit)
"Abbb"	5A	10	001	
"AHbb"	4A	10	001	
"ALbb"	5E	10	001	
"ALRb"	1E	01	000	
"ARbb"	1A	01	000	
	….	…..	…….	
"MVCb"	D2	11	100	

2) MACHINE-OP TABLE(MOT) FOR PASS1 AND PASS2

Pseudo-op (5-byte) (character)	Address of routine to process pseudo-op (3-byte=24bit addrss
"DROPb" "ENDbb" "EQUbb" "START" USING"	P1DROP P1END P1EQU P1START P1USING

3) SYMBOL TABLE(ST) FOR PASS1 AND PASS2

Symbol (8-bytes) (character)	Value (4-bytes) (hexadecimal)	Length (1-bye) (hexadecimal)	Relocation (1-byte) (character)
"JOHNbbbb"	0000	01	"R"
"FOURbbbb"	000C	04	"R"
"FIVEbbbb"	0010	04	"R"
"TEMPbbbb"	0014	04	"R"

4) SAME FIELD USE FOR TABLE FOR LITERAL (LT)

5) BASE TABLE(BT) F0R PASS2

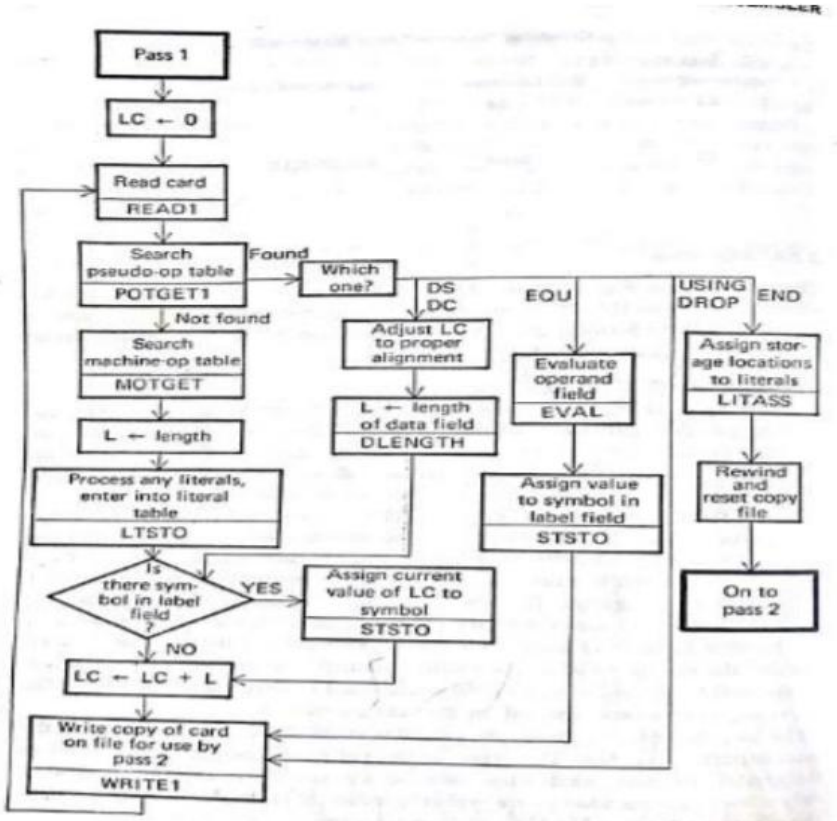
Availability indicator (1-byte) (character)	Designated relative address contents of base register(3-byte=24bit address) (hexadecimal)
"N" "N" . . "N" "Y"	- - - 00 00 00

Y=register specified in USING pseudo-op

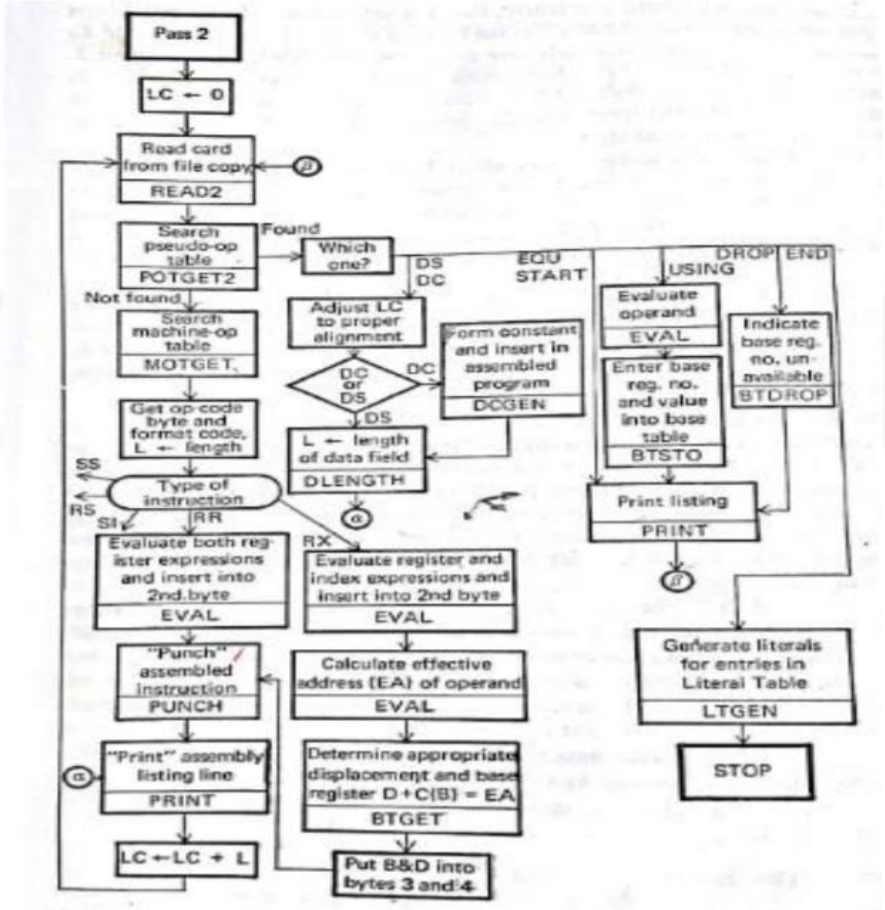
N=register never specified in USING pseudo-op/subsequently made unavailable by the DROP pseudo-op

Flowchart:

Pass1:



Pass2:



Program for TWO PASS ASSEMBLER

```
#include<stdio.h>
#include<string.h>
#include<conio.h>
void main(){
char *code[9][4]={
{"PRG1","START","",""},
{"","USING","*","15"},
{"","L","",""},
{"","A","",""},
{"","ST","",""},
{"FOUR","DC","F",""},
{"FIVE","DC","F",""},
{"TEMP","DS","F",""},
{"","END","",""}
};
char av[2],avail[15]={'N','N','N','N','N','N','N','N','N','N','N','N','N','N','N'};
int i,j,k,count[3],lc[9]={0,0,0,0,0,0,0,0,0},loc=0;
clrscr();
printf("------------------------------\n");
printf("LABLE\t\t\OPCODE\n");
printf("------------------------------\n\n");
for(i=0;i<=8;i++)
{
for(j=0;j<=3;j++)
{
printf("%s\t\t",code[i][j]);
}
printf("\n");}
getch();
printf("-------------------------");
printf("\n VALUES FOR LC:\n\n");
for(j=0;j<=8;j++){
if((strcmp(code[j][1],"START")!=0)&&(strcmp(code[j][1],"USING")!=0)&&(strcmp(code[j][1],"L")!=0))
lc[j]=lc[j-1]+4;
printf("%d\t",lc[j]);
}
printf("\n\nSYMBOL TABLE:\n-----------------------------\n");
printf("SYMBOL\t\tVALUE\t\tLENGTH\t\tR/A");
printf("\n-------------------------\n");
for(i=0;i<9;i++){
if(strcmp(code[i][1],"START")==0){
printf("%s\t\t%d\t\t%d\t\t%c\n",code[i][0],loc,4,'R');}
else if(strcmp(code[i][0],"")!=0){
printf("%s\t\t%d\t\t%d\t\t%c\n",code[i][0],loc,4,'R');
loc=loc+4;}
else if(strcmp(code[i][1],"USING")==0){}
else
{loc=loc+4;}}
printf("-----------------------------------");
printf("\n\nBASE TABLE:\n-----------------------------\n");
printf("REG NO\t\tAVAILIBILITY\tCONTENTS OF BASE TABLE");
printf("\n-----------------------------------\n");
for(j=0;j<=8;j++)
{
if(strcmp(code[j][1],"USING")!=0)
{}
```

```
else{
strcpy(av,code[j][3]);
}}
count[0]=(int)av[0]-48;
count[1]=(int)av[1]-48;
count[2]=count[0]*10+count[1];
avail[count[2]-1]='Y';
for(k=0;k<16;k++){
printf("%d\t\t%c\n",k,avail[k-1]);
}
printf("----------------------------------\n");
printf("Continue..??\n\n");
getch();
printf("PASS2 TABLE:\n\n");
printf("LABLE\t\tOP1\t\tLC\t\t");
printf("\n---------------------\n");
loc=0;
for(i=0;i<=8;i++){
for(j=0;j<=3;j++){
printf("%s\t\t",code[i][j]);}
j=0;
printf("\n");}
printf("-----------------------------");
getch();
}
```

OUTPUT ::-

```
LABLE TABLE::-
------------------------------
LABLE          OPCODE
------------------------------
PRG1           START
               USING          *              15
               L
               A
               ST
FOUR           DC             F
FIVE           DC             F
TEMP           DS             F
               END
-------------------------
 VALUES FOR LC:

0      0      0      4      8      12      16      20      24

SYMBOL TABLE:
------------------------------
SYMBOL         VALUE     LENGTH     R/A
-------------------------
PRG1             0             4             R
FOUR             12            4             R
FIVE             16            4             R
TEMP             20            4             R
-----------------------------------
```

BASE TABLE:

REG NO	AVAILIBILITY	CONTENTS OF BASE TABLE
0		
1	N	
2	N	
3	N	
4	N	
5	N	
6	N	
7	N	
8	N	
9	N	
10	N	
11	N	
12	N	
13	N	
14	N	
15	Y	

Continue..??

PASS2 TABLE:

LABLE	OP1	LC
PRG1	START	
	USING *	15
	L	
	A	
	ST	
FOUR	DC F	
FIVE	DC F	
TEMP	DS F	
	END	

Conclusion: Hence implemented a two pass assembler.

SIGN AND REMARK

DATE

R1	R2	R3	R4	R5	Total (15 Marks)	Signature