Madhuben and Bhanubhai Patel Institute of Technology

(A constituent of CVM University)

(System Software)

Project definition: Implement Assembler Pass-1 Target Code.

Submitted by:

Name:

1) Briyansi Dabhi (180630107009)
2) Astha Mehrotra (180630107029)
3) Rutvi Shah(180630107091)

Class: 6CE-1
Subject: System Software (3160715)
Academic Year: 2020-2021

Aim: To implement Assembler Pass-2 Target Code.

➔ **What does Assembler mean?**

Assembler is a program for converting instructions written in low-level assembly code into relocatable machine code and generating along information for the loader.

➔ Assembler generates instructions by evaluating the mnemonics (symbols) in operation field and find the value of symbol and literals to produce machine code. Now, if assembler do all this work in one scan then it is called single pass assembler, otherwise if it does in multiple scans then called multiple pass assembler.

➔ Here assembler divide these tasks in two passes:

Pass-1:

- Define symbols and literals and remember them in symbol table and literal table respectively.
- Keep track of location counter
- Process pseudo-operations

Pass-2:

- Generate object code by converting symbolic op-code into respective numeric op-code
- Generate data for literals and look for values of symbols

Now, let us try to implement the Assembler Pass-2 code.

➔ **Implementation and Working of Assembler Pass-1?**

Step-1: START 200 (here no symbol or literal is found so both table would be empty)
Step-2: MOVER R1, ='3' 200 (='3' is a literal so literal table is made)

Literal	Address
='3'	– – –

Step-3: MOVEM R1, X 201

X is a symbol referred prior to its declaration so it is stored in symbol table with blank address field.

Symbol	Address
X	– – –

Step-4: L1 MOVER R2, =’2’ 202

L1 is a label and =’2’ is a literal so store them in respective tables

Symbol	Address
X	– – –
L1	202

Literal	Address
=’3’	– – –
=’2’	– – –

Step-5: LTORG 203

Assign address to first literal specified by LC value, i.e., 203

Literal	Address
=’3’	203
=’2’	– – –

Step-6: X DS 1 204

It is a data declaration statement i.e X is assigned data space of 1. But X is a symbol which was referred earlier in step 3 and defined in step 6.This condition is called Forward Reference Problem where variable is referred prior to its declaration and can be solved by back-patching. So now assembler will assign X the address specified by LC value of current step.

Symbol	Address
X	204
L1	202

Step-7: END 205

Program finishes execution and remaining literal will get address specified by LC value of END

instruction. Here is the complete symbol and literal table made by pass 1 of assembler.

Symbol	Address
X	204
L1	202

Literal	Address
=’3’	203
=’2’	205

Now tables generated by pass 1 along with their LC value will go to pass-2 of assembler for further processing of pseudo-opcodes and machine op-codes.

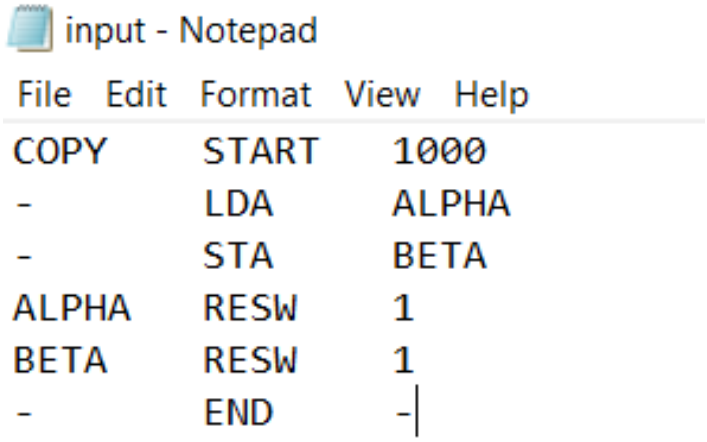
CODE:

```
#include<stdio.h>
#include<stdlib.h>
#include<conio.h>
#include<string.h>
void main()
{
  FILE *f1,*f2,*f3,*f4,*f5;
  int lc,sa,i=0,j=0,m[10],pgmlen,len,k,len1,l=0;
  char name[10],opnd[10],la[10],mne[10],s1[10],mne1[10],opnd1[10];
  char lcs[10],ms[10];
 char sym[10],symaddr[10],obj1[10],obj2[10],s2[10],q[10],s3[10];
 clrscr();
 f1=fopen("input.txt","r");
 f2=fopen("optab.txt","r");
 f3=fopen("symtab.txt","w+");
 f4=fopen("symtab1.txt","w+");
 f5=fopen("output.txt","w+");
 fscanf(f1,"%s%s%s",la,mne,opnd);
 if(strcmp(mne,"START")==0)
 {
   sa=atoi(opnd);
   strcpy(name,la);
   lc=sa;
 }
 strcpy(s1,"*");
 fscanf(f1,"%s%s%s",la,mne,opnd);
 while(strcmp(mne,"END")!=0)
 {
  if(strcmp(la,"-")==0)
  {
    fscanf(f2,"%s%s",mne1,opnd1);
    while(!feof(f2))
    {
      if(strcmp(mne1,mne)==0)
      {
   m[i]=lc+1;
   fprintf(f3,"%s\t%s\n",opnd,s1);
   fprintf(f5,"%s\t0000\n",opnd1);
   lc=lc+3;
   i=i+1;
   break;
s%s",sym,symaddr);
    while(!feof(f3))
    {
      if(strcmp(sym,la)==0)
      {
   itoa(lc,lcs,10);
```

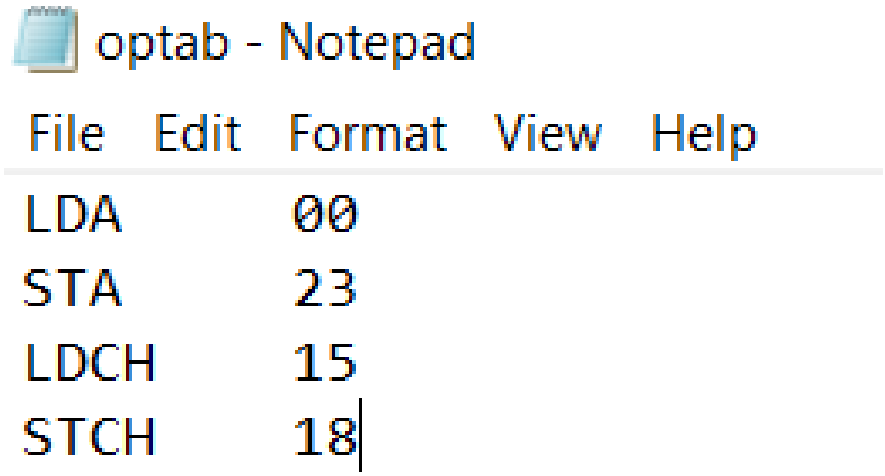
```
  fprintf(f4,"%s\t%s\n",la,lcs);
  itoa(m[j],ms,10);
  j=j+1;
  fprintf(f5,"%s\t%s\n",ms,lcs);
  i=i+1;
  break;
    }
    else
  fscanf(f3,"%s%s",sym,symaddr);
  } //f3
  if(strcmp(mne,"RESW")==0)
   lc=lc+3*atoi(opnd);
  else if(strcmp(mne,"BYTE")==0)
  {
   strcpy(s2,"-");
   len=strlen(opnd);
   lc=lc+len-2;
   for(k=2;k<len;k++)
   {
   q[l]=opnd[k];
   l=l+1;
   }
   fprintf(f5,"%s\t%s\n",q,s2);
   break;
  }
  else if(strcmp(mne,"RESB")==0)
   lc=lc+atoi(opnd);
  else if(strcmp(mne,"WORD")==0)
  {
    strcpy(s3,"#");
    lc=lc+3;
    fprintf(f5,"%s\t%s\n",opnd,s3);
    break;
  }
 } // else la=-
  fseek(f2,SEEK_SET,0);
  fscanf(f1,"%s%s%s",la,mne,opnd);
}
fseek(f5,SEEK_SET,0);
pgmlen=lc-sa;
printf("H^%s^%d^0%x\n",name,sa,pgmlen);
printf("T^");
printf("00%d^0%x",sa,pgmlen);
fscanf(f5,"%s%s",obj1,obj2);
while(!feof(f5))
{
 if(strcmp(obj2,"0000")==0)
   printf("^%s%s",obj1,obj2);
 else if(strcmp(obj2,"-")==0)
   {
   printf("^");
   len1=strlen(obj1);
   for(k=0;k<len1;k++)
   printf("%d",obj1[k]);
   }
```

INPUT FILES:

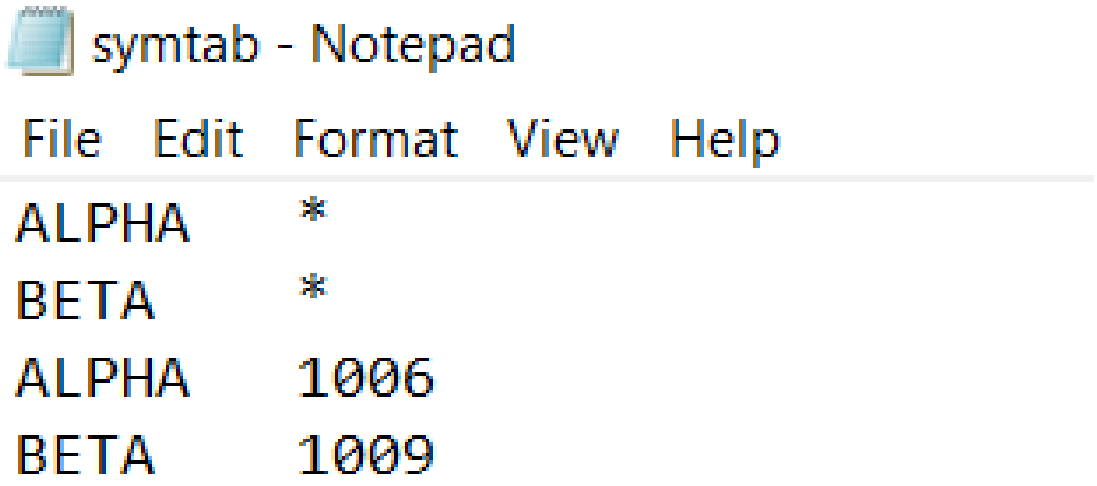
1) **input.txt**



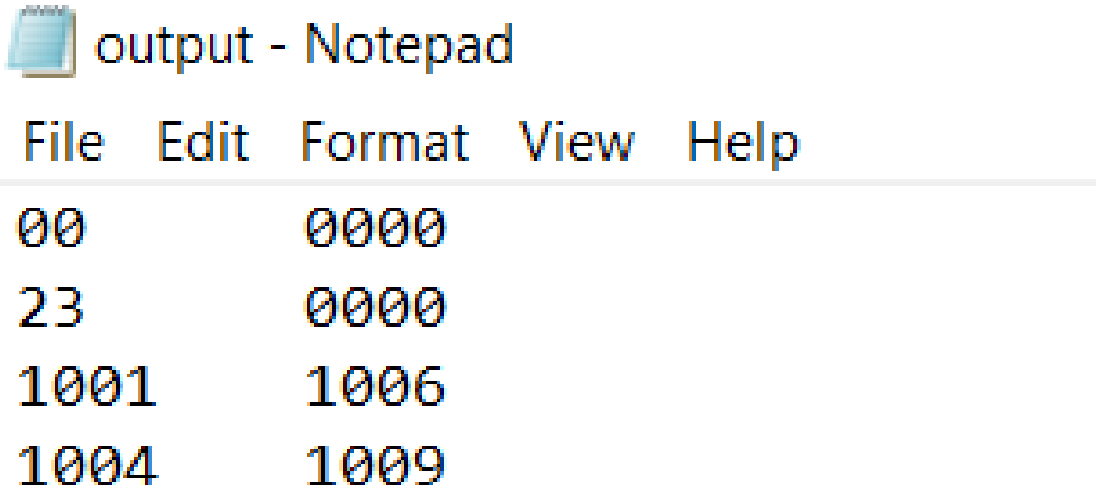
2) **optab.txt**



3) symtab.txt



4) outputl.txt



OUTPUT:

```
H^COPY^1000^0c
T^001000^0c^000000^230000
T^1001^02^1006
T^1004^02^1009
E^001000
```

Conclusion:

Thus, we have implemented the Assembler Pass-2 target code.

End-of-File