Name: Aditya Somani | TE (IT) – A | Roll No. T1851061 | PRN No. 71901204L

Assignment No. 1 and 2

1. Write a program to implement Pass-I of Two-pass assembler for Symbols and Literal processing considering following cases:

   i. Forward references

   ii. DS and DC statement

   iii. START, EQU, LTORG, END.

   iv. Error handling: symbol used but not defined, invalid instruction/register etc.

2. Write a program to implement Pass-II of Two-pass assembler for output of Assignment 1.

Aim: Write a program to implement II pass assembler. (For hypothetical instruction set from Dhamdhere)

a. Consider following cases only (Literal processing not expected)
b. Forward references
c. DS and DC statement
d. START, EQU
e. Error handling: symbol used but not defined, invalid instruction/register etc.

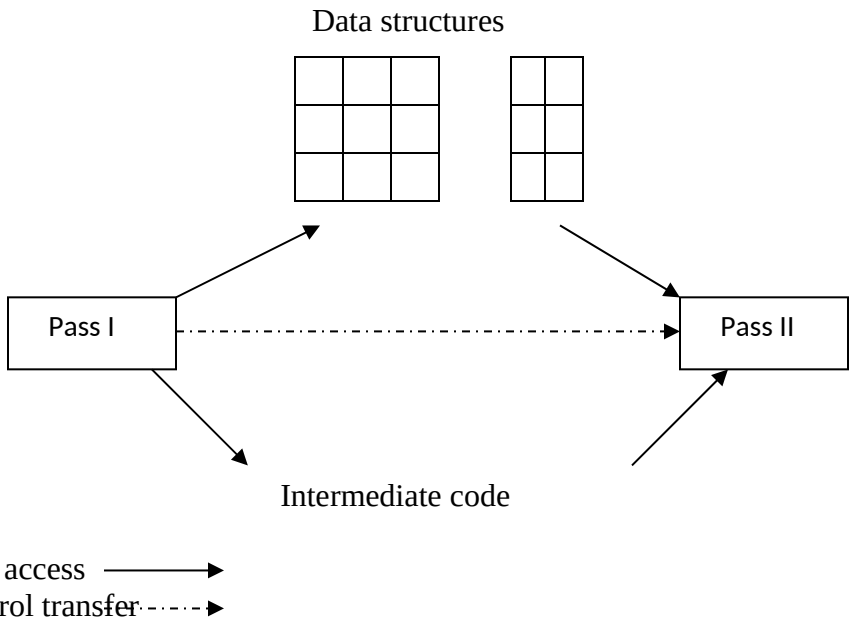
OBJECTIVE:

1. To study basic translation process of assembly language to machine language.
2. To study two pass assembly process.

THEORY:

A language translator bridges an execution gap to machine language of computer system. An assembler is a language translator whose source language is assembly language.

Language processing activity consists of two phases, Analysis phase and synthesis phase. Analysis of source program consists of three components, Lexical rules, syntax rules and semantic rules. Lexical rules govern the formation of valid statements in source language. Semantic rules associate the formation meaning with valid statements of language. Synthesis phase is concerned with construction of target language statements, which have the same meaning as source language statements. This consists of memory allocation and code generation.



Over view of two pass assembly

Analysis of source program statements may not be immediately followed by synthesis of equivalent target statements. This is due to forward references issue concerning memory requirements and organization of Language Processor (LP).

Forward reference of a program entity is a reference to the entity, which precedes its definition in the program. While processing a statement containing a forward reference, language processor does not posses all relevant information concerning referenced entity. This creates difficulties in synthesizing the equivalent target statements. This problem can be solved by postponing the generation of target code until more information concerning the entity is available. This also reduces memory requirements of LP and simplifies its organization. This leads to multi-pass model of language processing.

Language Processor Pass: -

It is the processing of every statement in a source program or its equivalent representation to perform language-processing function.

Assembly Language statements: -

There are three types of statements Imperative, Declarative, Assembly directives. An imperative statement indicates an action to be performed during the execution of assembled program. Each imperative statement usually translates into one machine instruction. Declarative statement e.g. DS reserves areas of memory and associates names with them. DC constructs memory word containing constants. Assembler directives instruct the assembler to perform certain actions during assembly of a program,
e.g. START<constant> directive indicates that the first word of the target program generated by assembler should be placed at memory word with address <constant>

Function Of Analysis And Synthesis Phase:

Analysis Phase: -

Isolate the label operation code and operand fields of a statement.

Enter the symbol found in label field (if any) and address of next available machine word into symbol table.
Validate the mnemonic operation code by looking it up in the mnemonics table.
Determine the machine storage requirements of the statement by considering the mnemonic operation code and operand fields of the statement.
Calculate the address of the address of the first machine word following the target code generated for this statement (Location Counter Processing)

Synthesis Phase:
Obtain the machine operation code corresponding to the mnemonic operation code by searching the mnemonic table.
Obtain the address of the operand from the symbol table.
Synthesize the machine instruction or the machine form of the constant as the case may be.

Design of a Two Pass Assembler: -

Tasks performed by the passes of two-pass assembler are as follows:

Pass I: -
Separate the symbol, mnemonic opcode and operand fields.
Determine the storage-required foe every assembly language statement and update the location counter.
Build the symbol table and the literal table.
Construct the intermediate code for every assembly language statement.

Pass II: -

Synthesize the target code by processing the intermediate code generated during

Data structures required for pass I:

1. Source file containing assembly program.
2. MOT: A table of mnemonic op-codes and related information.
   It has the following fields
   Mnemonic : Such as ADD, END, DC
   TYPE : IS for imperative, DL for declarative and AD for Assembler directive
   OP- code : Operation code indicating the operation to be performed.
   Length : Length of instruction required for Location Counter Processing

Hash table Implementation of MOT to minimize the search time required for searching the instruction.

Index	Mnemonic	TYPE	OP-Code	Length	Link
0	ADD	IS	01	01	-1
1	BC	IS	07	01	-1
2	COMP	IS	06	01	-1
3	DIV	IS	08	01	5
4	EQU	AD	03	-	7
5	DC	DL	01	-	6
6	DS	DL	02	-	-1

7	END	AD	05	-	-1

Hash Function used is ASCII Value of the First letter of Mnemonic – 65. This helps in retrieving the op- code and other related information in minimum time. For Example the instruction starting with alphabet 'A' will be found at index location 0, 'B' at index 1, so on and so forth. If more instructions exist with same alphabet then the instruction is stored at empty location and the index of that instruction is stored in the link field. Thus instructions starting with alphabet 'D' will be stored at index locations 3,5,and 6. Those starting with E will be stored at 4 and 7 and the process continues.

1. **SYMTB: The symbol table.**
   Fields are Symbol name, Address (LC Value). Initialize all values in the address fields to -1 and when symbol gets added when it appears in label field replace address value with current LC. The symbol if it used but not defined will have address value -1 which will be used for error detection.

Symbol	Address
Loop	204
Next	214

4. **LITTAB: and POOLTAB :** Literal table stores the literals used in the program and POOLTAB stores the pointers to the literals in the current literal pool.

Literal	Address
= '5'	
= '1'	
= '1'	

5. **Intermediate form used Variant 1 / Variant 2**
   Students are supposed to write the variant used by them.

Data Structure used by Pass II:
1. OPTAB: A table of mnemonic opcodes and related information.
2. SYMTAB: The symbol table
3. LITTAB: A table of literals used in the program
4. Intermediate code generated by Pass I
5. Output file containing Target code / error listing.

Algorithm
1 Open the source file in input mode.
2. if end of file of source file go to step 8.
3. Read the next line of the source program
4. Separate the line into words. These words could be stored in array of strings.
5. Search for first word in mnemonic opcode table, if not present it is a label , add this as a symbol in symbol table with current LC. And then search for second word in mnemonic opcode table.

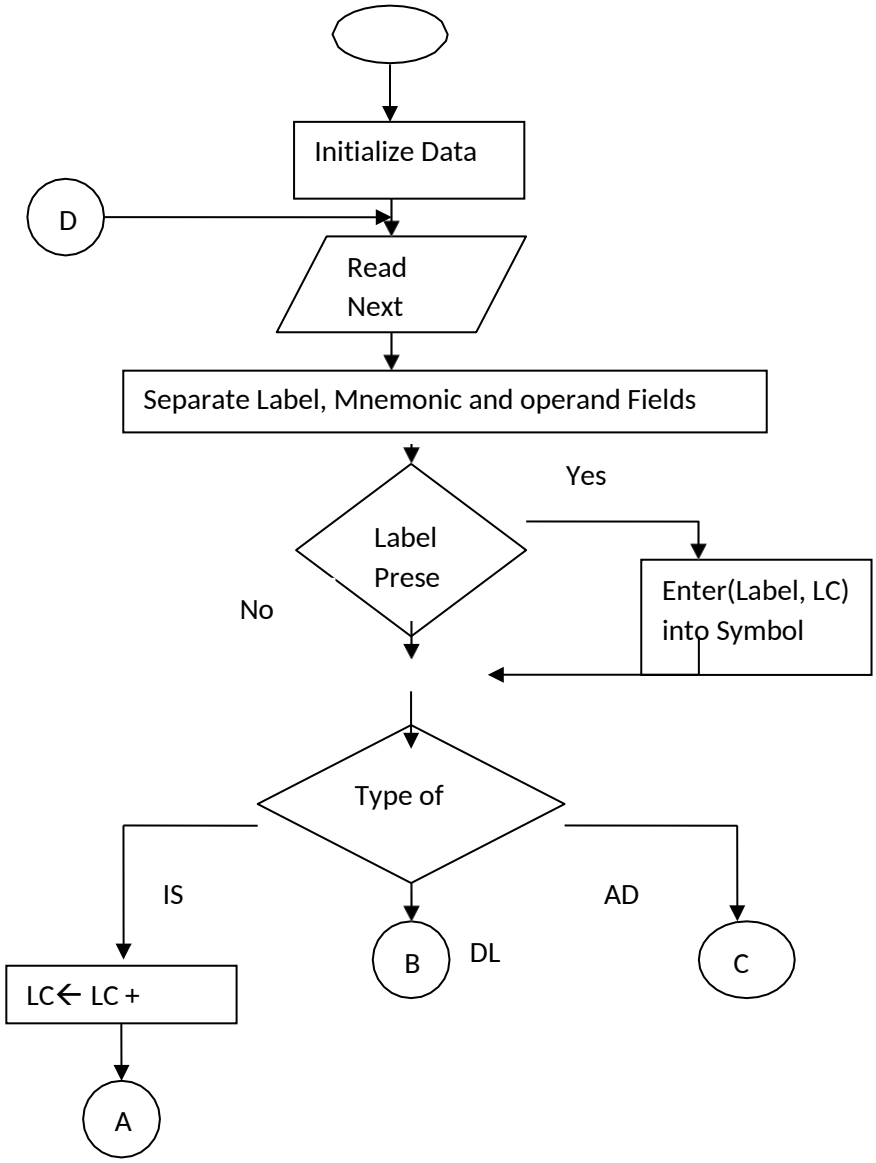
6. If instruction is found
   case 1 : imperative statement
   case 2: Declarative statement
   case 3: Assembler Directive
   Generate Intermediate code and write to Intermediate code file.
7. go to step 2.
8. Close source file and open intermediate code file
9. If end of file (Intermediate code), go to step 13
10. Read next line from intermediate code file.
11. Write opcode, register code, and address of memory(to be fetched from literal or symbol table depending on the case) onto target file. This is to be done only for Imperative statement.
12 go to step 9.
13. Close all files.
14. Display symbol table, literal table and target file.

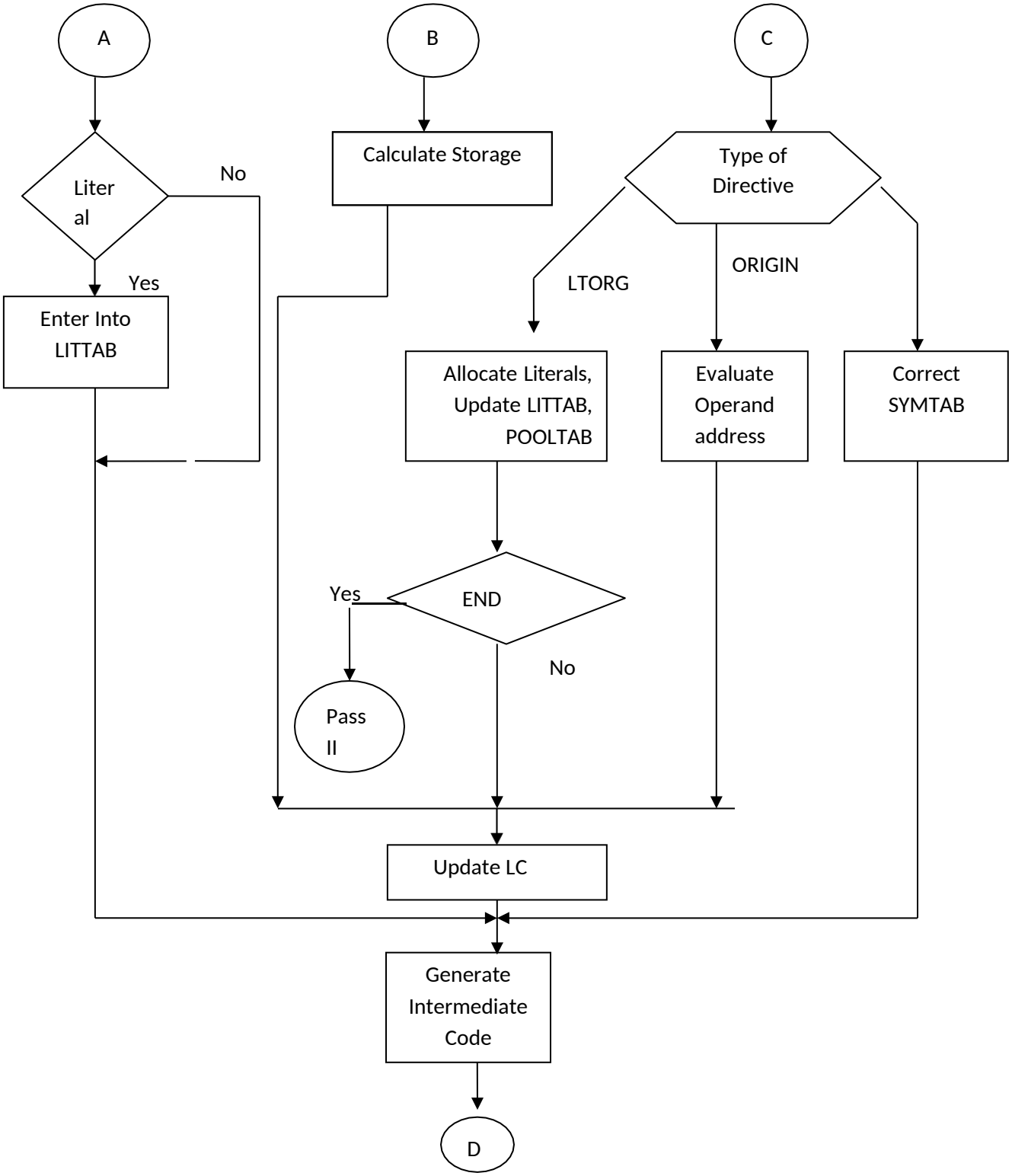
Imperative statement case :

1. If opcode >= 1 && opcode <=8 (Instruction requires register operand)
   a. Set type as IS, get opcode, get register code, and make entry into symbol or literal table as the case may be. In case of symbol, used as operand, LC field is not known so LC could be -1. Perform LC processing LC++. Updating of symbol table should consider error handling.
2. if opcode is 00 (stop) :
   Set all fields of Intermediate call as 00. LC++
3. else register operand not required (Read and Print)
   Same as case 1, only register code is not required, so set it to zero. Here again update the symbol table. LC++

On similar lines we can identify the cases for declarative and assembler directive statements based on opcode.

Flowchart for first pass of two pass assembler


Initialize Data
D
Read
Next
Separate Label, Mnemonic and operand Fields
Label
Prese
Yes
No
Enter(Label, LC)
into Symbol
Type of
IS
AD
B
DL
C
LC← LC +
A



List of hypothetical instructions:

Instruction Opcode	Assembly mnemonic	Remarks
00	STOP	stop execution
01	ADD	first operand modified condition code set
02	SUB	first operand modified condition code set
03	MULT	first operand modified condition code set
04	MOVER	register memory
05	MOVEM	memory ← register
06	COMP	sets condition code
07	BC	branch on condition code
08	DIV	analogous to SUB
09	READ	first operand is not used.
10	PRINT	first operand is not used.

Sample Input & Output: -

SAMPLE INPUT FILE		SAMPLE OUTPUT FILE OF INTERMEDIATE CODE Using Variant One
	START 202	
	MOVER AREG, =’5	202) (IS,04)(1)(L,1)
	MOVEM AREG , A	203) (IS,05)(1)(S,1)
LOOP	MOVER AREG, A	204) (IS,04)(1)(S,1)
	MOVER CREG, B	205) (IS,04)(3)(S,3)
	ADD CREG, = ‘1’	206) (IS,01)(3)(L,2)
	MOVEM CREG, B	207) (IS,05)(2)(S,3)
	SUB CREG,A	208) (IS,02)(3)(S,1)
	BC ANY, NEXT	209) (IS,07)(6)(S,4)
	LTORG	210) (AD, 04)
		211)
	ADD CREG,B ’	212) (IS,01)(3)(S,2)
	BC LE LOOP	213) (IS,07)(2)(S,2)
NEXT	SUB AREG, = ‘1’	214) (IS,02)(1)(L,3)
	BC LT, BACK	215) (IS.07)(1)(S,5)
	STOP	216) (IS,00)
	ORIGIN 219	217) (AD,03)
	MULT CREG, B	219) (IS,03)(3)(S,3)
A	DS 1	220) (DL,02) (C,1)
BACK	EQU LOOP	221) (AD,02)
B	DS 1	221) (DL,02) (C,1)
	END	222) (AD,05)

SYMBOL TABLE

Index	Symbol	Address
1	A	220
2	LOOP	204
3	B	221
4	NEXT	214
5	BACK	204

POOL TABLE

LIT_Ind
01
03

LITERAL TABLE

Index	LITERAL	ADDRESS
1	5	210
2	1	211
3	1	222

SAMPLE INPUT FILE

```
          START 202
          MOVER AREG, =’5
          MOVEM AREG , A
LOOP      MOVER AREG, A
          MOVER CREG, B
          ADD CREG, = ‘1’
          MOVEM CREG, B
          SUB CREG, A
          BC ANY, NEXT
          LTORG

          ADD CREG, B      ’
          BC LE LOOP
NEXT      SUB AREG, = ‘1’
          BC LT, BACK
          STOP
          ORIGIN 219
          MULT CREG, B
A         DS     1
BACK      EQU  LOOP
B         DS     1
          END
```

SAMPLE OUTPUT FILE FOR TARGET CODE

```
202) 04 1 210
203) 05 1 220
204) 04 1 220
205) 04 3 221
206) 01 3 211
207) 05 3 221
208) 03 3 220
209) 07 6 214
210) 00 0 005
211) 00 0 001
212) 01 3 221
213) 07 2 204
214) 02 1 222
215)  07 1 220
216)  00 0 000
217)
219)  03 3 221
220)
221)
221)
222) 00 0 001
```

Instructions to the Students: -
Students are supposed to write about the organizations of the different data structures such as array, link list, etc.
Assumptions and limitations if any should be clearly mentioned.
Students are supposed to create two output files, one without errors & one with errors indicating the type of error.

At least following errors must be handled.

Errors: -

Forward reference(Symbol used but not defined): -
This error occurs when some symbol is used but it is not defined into the program.
Duplication of Symbol: -
This error occurs when some symbol is declared more than once in the program.
Mnemonic error:
If there is invalid instruction then this error will occur.
Register error: -
If there is invalid register then this error will occur.
Operand error: -
This error will occur when there is an error in the operand field,

Conclusion:
Thus the program for implementation pass 1 and pass 2 of two pass assembler has been executed successfully.

Program

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

#define max 20
#define ad_max 5
#define is_max 11
#define reg_max 4
#define cond_max 6

struct sym_table{ //structure to store symbol table
        char sym[15];
        int addr;
}s[max];

struct lit_table{ //structure to store literal table
        char lit[5];
        int addr;
}l[max];

int pool_table[max];//to store pooltable;
int sym_count=0,lit_count=0,pool_count=0,token_count=0,lc=-1,i,j,k;
char tok1[max],tok2[max],tok3[max],tok4[max],tok5[max]; //to store token in line
char buffer[80],temp[max];
```

```
	FILE *fp1,*fp2,*fp;

	char reg[4][10] = {"AREG","BREG","CREG","DREG"};
	char ad[5][10] = {"START","END","ORIGIN","EQU","LTORG"};
	char is[11][10] =
{"STOP","ADD","SUB","DIV","MULT","MOVER","MOVEM","BC","COMP","READ","PRI
NT"};
	char cond[6][10] = {"LT","LE","GT","GE","EQ","ANY"};


	int reg_search(char tok[]){
		int i;
		for(i=0;i<reg_max;i++){
			if(strcmp(reg[i],tok)==0)
			  return i;
		}
		return -1;
	}


	int ad_search(char tok[]){
		int i;
		for(i=0;i<ad_max;i++){
			if(strcmp(ad[i],tok)==0)
			  return i;
		}
		return -1;
	}

	int is_search(char tok[]){
		int i;
		for(i=0;i<is_max;i++){
			if(strcmp(is[i],tok)==0)
			  return i;
		}
		return -1;
	}

	int cond_search(char tok[]){
		int i;
		for(i=0;i<cond_max;i++){
			if(strcmp(cond[i],tok)==0)
			  return i;
		}
		return -1;
	}

	int sym_search(char tok[]){
```

```
        int i;
        for(i=0;i<sym_count;i++){
                if(strcmp(s[i].sym,tok)==0)
                  return i;
        }
        return -1;
}

int lit_search(char tok[]){
        int i;
        for(i=pool_table[pool_count];i<lit_count;i++){
                if(strcmp(l[i].lit,tok)==0)
                  return i;
        }
        return -1;
}

void display_sym_table(){
        int i;
        printf("Symbol\tAddress\n");
        for(i=0;i<sym_count;i++)
           printf("%s\t%d\n",s[i].sym,s[i].addr);
}


void display_lit_table(){
        int i;
        printf("Literal Count\tLiteral\tAddress\n");
        for(i=0;i<lit_count;i++)
           printf("%d\t%s\t%d\n",i,l[i].lit,l[i].addr);
}

void display_pool_table(){
        int i;
        printf("Pool_index\tPool_Base\n");
        for(i=0;i<pool_count;i++)
           printf("%d\t%d\n",i,pool_table[i]);
}

void print_file(char fn[]){
        FILE *fp = fopen(fn,"r");

        if(fp==NULL){
                printf("Can't open file\n");
                exit(0);
        }

        while(fgets(buffer,80,fp)){
```

```
                printf("%s",buffer);
        }
        fclose(fp);
}


void pass1_assembler(){

        fp1 = fopen("input.txt","r");
        if(fp1==NULL){
                printf("Can't open file 1\n");
                exit(0);
        }

        fp2 = fopen("output.txt","w");
        if(fp2==NULL){
                printf("Can't open file 2\n");
                exit(0);
        }

        while(fgets(buffer,80,fp1)){ //here fgets function reads line by line
                strcpy(tok1," ");
                strcpy(tok2," ");
                strcpy(tok3," ");
                strcpy(tok4," ");

                token_count    = sscanf(buffer,"%s %s %s %s",tok1,tok2,tok3,tok4);
//MOVER A, B (maximum we can have 4 such tokens so we are scanning 4 tokens)

                switch(token_count){

                        case 1:  //START,STOP,LTORG,END
                                i = is_search(tok1);
                                if(i==0) //STOP
                                {
                                        fprintf(fp2,"(IS, %02d)\n",i);    //(IS,00)
                                        break;
                                }

                                i = ad_search(tok1);
                                if(i==0) //START
                                {
                                        fprintf(fp2,"(AD, %02d)\n",i); //(AD,00)
                                        lc = -1;
//pool table
                                        break;                                          //
                                }
```

```
                                        if(i==1 || i==4)  //LTORG,END
                                        {
                                                fprintf(fp2,"(AD, %02d)\n",i);     //(AD,01)

        for(k=pool_table[pool_count];k<lit_count;k++)
                                                {
                                                        l[k].addr = lc++;
                                                }
                                                pool_table[++pool_count] = lit_count;
                                                lc--;
                                        }
                                        break;


                                case 2:
                                        //START,ORIGIN,PRINT,READ,STOP(with label)
START 200,ORIGIN 105,PRINT A,READ B,NEXT STOP
                                        i = ad_search(tok1);
                                        if(i==0 || i==2)  //START,ORIGIN
                                        {
                                                        lc = atoi(tok2)-1;
                                                        fprintf(fp2,"(AD, %02d) (C,
%02d)\n",i,lc+1);
                                                        break;
                                        }

                                        i = is_search(tok1); //PRINT,READ
                                        if(i==9 || i==10)
                                        {
                                                j = sym_search(tok2);
                                                if(j==-1)
                                                {
                                                        fprintf(fp2,"(IS, %02d) (S,
%02d)\n",i,sym_count);
                                                        strcpy(s[sym_count++].sym,tok2);
                                                        break;
                                                }
                                                else
                                                {
                                                        fprintf(fp2,"(IS, %02d) (S,
%02d)\n",i,j);
                                                }
                                        }
                                        break;

                                case 3:
                                        //ADD-DIV               ADD AREG, B    MOVER
BREG,A
```

```
i = is_search(tok1);
                    //ADD AREG, ='5'
if(i>=1 && i<=9)
{
        if(tok3[0]=='=')
        {
                j = lit_search(tok3);
                if(j==-1)
                {
                        sprintf(temp,"(L,
%02d)",lit_count);

        strcpy(l[lit_count++].lit,tok3);

                }
                else
                {
                        sprintf(temp,"(L, %02d)",j);
                }
        }
        else
        {
                j = sym_search(tok3);
                if(j==-1)
                {
                        sprintf(temp,"(S,
%02d)",sym_count);

        strcpy(s[sym_count++].sym,tok3);
                }
                else
                {
                        sprintf(temp,"(S, %02d)",j);
                }
        }
        tok2[strlen(tok2)-1] = '\0';


        if(i==7)   //BC              BC LE 5
ADD AREG, B
        {
                j = cond_search(tok2);
        }
        else
        {
                j = reg_search(tok2);
        }
```

```
                        fprintf(fp2,"(IS, %02d) %d %s\n",i,j,temp);
//ADD-DIV
                        break;
                }
                //DC
                if(strcmp(tok2,"DC")==0)                //A DC 5
                {
                        j = sym_search(tok1);
                        if(j==-1)
                        {
                                strcpy(s[sym_count].sym,tok1);
                                s[sym_count++].addr = lc;
                        }
                        else
                        {
                                s[j].addr = lc;
                        }

                        fprintf(fp2,"(DL, 00) (C,
%02d)\n",atoi(tok3));
                        break;
                }

                //DS
                if(strcmp(tok2,"DS")==0)          // A DS 10
                {
                        j = sym_search(tok1);
                        if(j==-1)
                        {
                                strcpy(s[sym_count].sym,tok1);
                                s[sym_count++].addr = lc;
                        }
                        else
                        {
                                s[j].addr = lc;
                        }
                        lc = lc + atoi(tok3)-1;
                        fprintf(fp2,"(DL, 01) (C,
%02d)\n",atoi(tok3));
                        break;
                }

                //EQU
                j = ad_search(tok2);                // A EQU B
(S,00) (AD,03) (S,01)
                if(j==3)
                {
                        i = sym_search(tok1);
```

```
                        k = sym_search(tok3);
                        if(i==-1)
                        {
                                strcpy(s[sym_count].sym,tok1);
                                s[sym_count++].addr = s[k].addr;
                        }
                        else
                        {
                                s[j].addr = s[k].addr;
                        }
                        fprintf(fp2,"(AD, %02d)\n",j);
                        lc--;
                        break;
                }
                break;

        case 4:
                        j = sym_search(tok1); //label          NEXT ADD
AREG, A
                        if(j==-1)
                        {
                                strcpy(s[sym_count].sym,tok1);
                                s[sym_count++].addr = lc;
                        }
                        else
                        {
                                s[j].addr = lc;
                        }

                        i = is_search(tok2);  //ADD-DIV

                        if(i>=1 && i<=9)
                        {
                                if(tok4[0]=='=')
                                 {
                                        j = lit_search(tok4);
                                        if(j==-1)
                                        {
                                                sprintf(temp,"(L,
%02d)",lit_count);

        strcpy(l[lit_count++].lit,tok4);

                                        }
                                        else
                                        {
                                                sprintf(temp,"(L, %02d)",j);
                                        }
```

```
                                        }
                                        else
                                        {
                                                j = sym_search(tok4);
                                                if(j==-1)
                                                {
                                                        sprintf(temp,"(S,
%02d)",sym_count);

        strcpy(s[sym_count++].sym,tok4);
                                                }
                                                else
                                                {
                                                        sprintf(temp,"(S, %02d)",j);
                                                }
                                        }
                                        tok3[strlen(tok3)-1] = '\0';
                                        if(i==7)
                                        {
                                                j = cond_search(tok3);
                                        }
                                        else
                                        {
                                                j = reg_search(tok3);
                                        }
                                        fprintf(fp2,"(IS, %02d) %d %s\n",i,j,temp);
                                        break;
                                }
                                break;

                        }
                        lc++;
                }
                fclose(fp1);
                fclose(fp2);
        }


        void twoPass()
        {
                lc = 0;
           fp2 = fopen("output.txt","r");
           fp = fopen("final.txt","w");
           while(fgets(buffer,80,fp2)){


                        token_count    = sscanf(buffer,"%s %s %s %s
%s",tok1,tok2,tok3,tok4,tok5); //MOVER A, B
```

```
                tok1[strlen(tok1)-1]='\0';
                tok2[strlen(tok2)-1]='\0';
                //tok3[strlen(tok3)-1]='\0';
                tok4[strlen(tok4)-1]='\0';              //index lit  addr
                tok5[strlen(tok5)-1]='\0';                                      //  0      ='9'  206
                //printf("%s\n",tok1);                  // 1 ='5'  212

switch(token_count){
        case 2:                                         //index base_of_lit
                tok3[strlen(tok3)-1]='\0';                      // 0        0
                if(strcmp(tok1+1,"AD")==0)  // LTORG, END           // 1        1
                {
                for(j=0;j<pool_count;j++)
                 if(l[pool_table[j]].addr==lc)
                        break;
                for(i=pool_table[j];i<pool_table[j+1];i++)
                {
                strcpy(temp,l[i].lit);
                temp[strlen(temp)-1]='\0';                      //='9
                fprintf(fp,"%d) + 00 0 %03d\n",lc++,atoi(strstr(temp,"'")+1));
                }
                        lc--;
                }
                else if(strcmp(tok1+1,"IS")==0)// STOP
                {
                        fprintf(fp,"%d) + 00 0 000\n",lc);
                }
                break;

case 4:
                        //printf("%s",tok1+1);
        tok3[strlen(tok3)-1]='\0';
        if(strcmp(tok1+1,"AD")==0) // START, ORIGIN
        {
                lc = atoi(tok4)-1;
        }

        else if(strcmp(tok1+1,"IS")==0)        // READ, PRINT
        {
                fprintf(fp,"%d) + %02d 0 %03d\n",lc,atoi(tok2),s[atoi(tok4)].addr);
        }
        else if(strcmp(tok1+1,"DL")==0)
        {
                if(atoi(tok2)==1)                                       // DS == 01
                {
                        for(i=0;i<atoi(tok4);i++)
                                fprintf(fp,"%d)\n",lc++);
                        lc--;
```

```
                }
                else if(atoi(tok2)==0)                          // DC
                {
                        fprintf(fp,"%d) + 00 0 %03d\n",lc,atoi(tok4));
                }
        }
        break;

                case 5:
                        /*Tok2[strlen(Tok2)-1]='\0';
                Tok3[strlen(Tok3)-1]='\0';
                Tok5[strlen(Tok5)-1]='\0';*/
                if(tok4[1]=='S')                // ADD-DIV with symbols
                {
                        fprintf(fp,"%d) + %02d %d
%03d\n",lc,atoi(tok2),atoi(tok3),s[atoi(tok5)].addr);
                }
                else if(tok4[1]=='L')           // ADD-DIV with literals
                {
                        fprintf(fp,"%d) + %02d %d
%03d\n",lc,atoi(tok2),atoi(tok3),l[atoi(tok5)].addr);
                }
                break;


        }
        lc++;
}
fclose(fp2);
fclose(fp);

}



void main(){
        pass1_assembler();



        printf("\n\nSOURCE CODE\n\n");
        print_file("input.txt");

        printf("\n\nINTERMEDIATE CODE\n\n");
        print_file("output.txt");

        printf("\n\nSymbol Table\n\n");
        display_sym_table();
```

```
        printf("\n:iteral Table\n");
        display_lit_table();

        printf("\nPool Table\n");
        display_pool_table();

        twoPass();
        print_file("final.txt");


}
```

Input.txt

```
START 100
READ N
A DS 2
MOVER BREG, C
MOVER AREG, B
ADD EREG, R
PRINT R
D EQU A + 1
B DS 2
N DS 2
C DC 2
END
```

Output

SOURCE CODE

```
START 100
READ N
A DS 2
MOVER BREG, C
MOVER AREG, B
ADD EREG, R
PRINT R
D EQU A + 1
B DS 2
N DS 2
C DC 2
END
```

INTERMEDIATE CODE

(AD, 00) (C, 100)
(IS, 09) (S, 00)
(DL, 01) (C, 02)
(IS, 05) 1 (S, 02)
(IS, 05) 0 (S, 03)
(IS, 01) -1 (S, 04)
(IS, 10) (S, 04)
(DL, 01) (C, 02)
(DL, 01) (C, 02)
(DL, 00) (C, 02)
(AD, 01)

Symbol Table

Symbol Address
N 110
A 101
C 112
B 108
R 0
D 107

:iteral Table
Literal Count Literal Address

Pool Table
Pool_index Pool_Base
0 0
100) + 09 0 110
101)
102)
103) + 05 1 112
104) + 05 0 108
105) + 01 -1 000
106) + 10 0 000
107)
108)
109)
110)
111) + 00 0 002

- - - - - - - - - - - - - - - - - - - -