Department of Computer Engineering

Academic Term : Jan-May 23-24

Class : T.E. (Computer)
Subject Name : System Programming and Compiler Construction
Subject Code : (CPC601)

Practical No:	7
Title:	One Pass and Two Pass Assembler
Date of Performance:	
Date of Submission:	
Roll No:	9601
Name of the Student:	Ivan Dsouza

Evaluation:

Sr. No	Rubric	Grade
1	Time Line (2)	
2	Output(3)	
3	Code optimization (2)	
4	Postlab (3)	

Signature of the Teacher :

Experiment No 7

Aim: Write a program to implement Two Pass Assembler.

Learning Objective: Translating mnemonic operation codes to their machine language equivalents. Assigning machine addresses to symbolic labels used by the programmer. Lastly to convert assembly language to binary.

Algorithm:

Pass 1:

1. Start

2.Intialize location counter to zero

3.Read opcode field of next instruction.

4.search opcode in pseudo opcode Table(POT)

5.If opcode is found in POT

5.1 If it is 'DS' or 'DC'

Adjust location counter to proper alignment.

Assign length of data field to 'L'

Go to step 9

5.2 If it is 'EQU'

Evaluate operand field

Assign values to symbol in label field

Go to step 3

5.3 If it is 'USING' or 'DROP' Go to step 3

5.4 If it is 'END'

Assign value to symbol in label field

Go to step 3

6. Search opcode in Machine Opcode Table.

7. Assign its length to ‘L’.

8. Process any literals and enter them into literal Table.

9.If symbol is there in the label field

Assign current value of Location Counter to symbol

10. Location Counter= Location Counter +L.

11.Go to step 3.

12. Stop.

Pass2:

1.Start

2.Intialize location counter to zero.

3. Read opcode field of next instruction.

4.Search opcode in pseudo opcode table.

5.If opcode is found in pseudo opcode Table

5.1 If it is ‘DS’ or ‘DC’

Adjust location counter to proper alignment.

If it is ‘DC’ opcode form constant and insert in assembled program

Assign length of data field to ‘L’

Go to step 6.4

5.2 If it is ‘EQU’ or ‘START’ ignore it. Go to step 3

5.3 If it is ‘USING’

Evaluate operand and enter base reg no. and value into base table

Go to step 3

5.4 If it is ‘DROP’

Indicate base reg no . available in base table . Go to step 3

5.5 If it is ‘END’

Generate literals for entries in Literal Table

Go to step 12

6 Search opcode in MOT

7. Get opcode byte and format code

8. Assign its length to ‘L’.

9. Check type of instruction.

10.If it is type ‘RR’ type

10.1 Evaluate both register expressions and insert into second byte.

10.2 Assemble instruction

10.3 Location Counter= Location Counter +L.

10.4.Go to step 3.

11. If it is ‘RX’ type

11.1 Evaluate register and index expressions and insert into second byte.

11.2 Calculate effective address of operand.

11.3 Determine appropriate displacement and base register

11.4 Put base and displacement into bytes 3 and 4

11.5 Location Counter= Location Counter +L.

11.6 Go to step 11.2

13 Stop.

Implementation Details

1. Read Assembly language input file.
2. Display output of Pass1 as the output file with Op-code Table, Symbol Table.
3. Display output of pass2 as the Op-code Table, Symbol Table , Copy file.

Test Cases:

1 Input symbol which is not defined

2 Input Opcode which is not entered in MOT

Conclusion:

PASS-1

Code:

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

int main(){
    char label[10],opcode[10],operand[10],mnemonic[10];
    int locctr = 0;

    FILE *fp1,*fp2,*fp3,*fp4;

    fp1 = fopen("input.txt","r");    // input
    fp2 = fopen("mot.txt","r");      // input
    fp3 = fopen("output.txt","w");   // output
    fp4 = fopen("st.txt","w");       // output

    fscanf(fp1,"%s %s %s",label,opcode,operand); //opcode start

    fprintf(fp3,"%d %s %s %s\n",locctr,label,opcode,operand);

    fscanf(fp1,"%s %s %s",label,opcode,operand);//opcode using

    fprintf(fp3,"%d %s %s %s\n",locctr,label,opcode,operand);

```

```
    while(strcmp(opcode,"END")!=0){

        if(strcmp(opcode,"DC")==0 || strcmp(opcode,"DS")==0){
            fprintf(fp4,"%s %d\n",label,locctr);
            fprintf(fp3,"%d %s %s %s\n",locctr,label,opcode,operand);
            locctr += 4;
        }
        else{
            fscanf(fp2,"%s",mnemonic);
            fprintf(fp3,"%d %s %s %s\n",locctr,label,opcode,operand);
            while(strcmp(mnemonic,"end")!=0){
                if(strcmp(opcode,mnemonic)==0){
                    locctr += 4;
                    break;
                }
                fscanf(fp2,"%s",mnemonic);
            }
            rewind(fp2);
        }

        fscanf(fp1,"%s %s %s",label,opcode,operand);
    }

    fprintf(fp3,"%d %s %s %s\n",locctr,label,opcode,operand);
    fclose(fp1);
    fclose(fp2);
    fclose(fp3);
    fclose(fp4);

    return 0;
}
```

Input:

Spcc > exp7 > mot.txt

```
A 5A
L 6A
ST 7A
end
```

Spcc > exp7 > input.txt

```
1    PG1 START 0
2    ** USING *,15
3    ** L 1,FIVE
4    ** A 1,FOUR
5    ** ST 1,TEMP
6    FIVE DC F'5'
7    FOUR DC F'4'
8    TEMP DS 1F
9    ** END
```

Output:

Spcc > exp7 > output.txt

```
1    0 PG1 START 0
2    0 ** USING *,15
3    0 ** USING *,15
4    0 ** L 1,FIVE
5    4 ** A 1,FOUR
6    8 ** ST 1,TEMP
7    12 FIVE DC F'5'
8    16 FOUR DC F'4'
9    20 TEMP DS 1F
10   24 ** END 1F
11
```

Spcc > exp7 > st.txt

```
1    FIVE 12
2    FOUR 16
3    TEMP 20
4
```

PASS-2

Code:

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

int main(){
    char label[10],opcode[10],operand[10],mnemonic[10],locctr[10],mlabel[10];

    FILE *fp1,*fp2,*fp3,*fp4,*fp5;

    fp1 = fopen("input.txt","r");        //input
    fp2 = fopen("mot.txt","r");          //input
    fp3 = fopen("output.txt","r");       //input
    fp4 = fopen("outTable.txt","w");     //output
    fp5 = fopen("BT.txt","w");           //output

    fscanf(fp3,"%s %s %s %s",locctr,label,opcode,operand); //START line 1 ignore
    fscanf(fp3,"%s %s %s %s",locctr,label,opcode,operand); //USING Line 2

    fprintf(fp5,"Y  %c%c    00 00 00",operand[2],operand[3]); //Base table

    fscanf(fp3,"%s %s %s %s",locctr,label,opcode,operand);

```

```
    fscanf(fp3,"%s %s %s %s",locctr,label,opcode,operand);

    while(strcmp(opcode,"END")!=0){

        if(strcmp(opcode,"DC")==0){
            fprintf(fp4,"%s\t%c\n",locctr,operand[2]);
        }
        else if(strcmp(opcode,"DS")==0){
            fprintf(fp4,"%s\t_\n",locctr);
        }
        else{
            fscanf(fp2,"%s %s",mnemonic,mlabel);
            while(strcmp(mnemonic,"end")!=0){
                if(strcmp(opcode,mnemonic)==0){
                    fprintf(fp4,"%s\t%s\n",mlabel,operand);
                    break;
                }
                fscanf(fp2,"%s %s",mnemonic,mlabel);
            }
            rewind(fp2);
        }

        fscanf(fp3,"%s %s %s %s",locctr,label,opcode,operand);

    }
    fclose(fp1);
    fclose(fp2);
    fclose(fp3);
    fclose(fp4);
    fclose(fp5);

    return 0;
}
```

Input:

Spcc > pass2 > input.txt

```
1    PG1 START 0
2    ** USING *,15
3    ** L 1,FIVE
4    ** A 1,FOUR
5    ** ST 1,TEMP
6    FIVE DC F'5'
7    FOUR DC F'4'
8    TEMP DS 1F
9    ** END
```

Spcc > pass2 > mot.txt

```
1    A 5A
2    L 6A
3    ST 7A
4    end
```

Spcc > pass2 > output.txt

```
1    0 PG1 START 0
2    0 ** USING *,15
3    0 ** USING *,15
4    0 ** L 1,FIVE
5    4 ** A 1,FOUR
6    8 ** ST 1,TEMP
7    12 FIVE DC F'5'
8    16 FOUR DC F'4'
9    20 TEMP DS 1F
10   24 ** END 1F
```

Output:

Spcc > pass2 > outTable.txt

```
1    6A  1,FIVE
2    5A  1,FOUR
3    7A  1,TEMP
4    12  5
5    16  4
6    20  _
7
```

Spcc > pass2 > BT.txt

```
1    Y  15    00 00 00
```

Post Lab Questions:

1. **Define the basic functions of assembler.**
2. **What is the need of SYMTAB (symbol table) in assembler?**
3. **What is the need of MOT in assembler**
4. **What is meant by one pass assembler?**

Ivan Dsouza
9601
T.E. Comps A

Exp 7

Page No.
Date

Ans 1] An Assembler translates assembly language code into m/c code performing functions like:

i) Parsing: Breaking down assembly ~~instruction~~ instructions into their components

ii) Addressing: Resolving symbolic abstractions to numerical values.

iii) Generating code output: Producing m/c code instructions based on the assembly language input.

iv) Error Handling: Detecting and reporting error.

v) Addressing modes translation

vi) Label and Symbol resolution.

Ans 2) SYM Table is needed for:

i) Tracking symbols (labels, variables, constants) and their associated memory addresses as values.

ii) Resolving references to symbols during assembly process

iii) Ensuring correct generation of m/c code with proper symbol substitution.

Page No.
Date

Ans3] MOT Table is necessary for:

i) Mapping mnemonic instructions to their corresponding m/c code operands.

ii) Providing information about the format and characteristics of instructions.

iii) Assisting in the generation of m/c code by the assembler.

Ans4] One-pass assembler

i) Processes the assembly language program in a single pass as it reads through the source code.

ii) Generates m/c code directly without needing to revisit previously scanned instructions.

iii) Typically used for simpler assembly languages where memory constraints are tight.