❖ SYSTEM PROGRAMMING PROJECT

- NAMES : PARTHIV SARKAR , PROTYAY RAY , ABHIJIT CHAWDHURY , BINOY SIKDAR SUDIPTO MISTRY
- ROLL Nos. : 1134 , 1140 , 1139 , 1142 , 1141
- CLASS : BCSE UG-3
- SECTION : A3
- SUBJECT : SYSTEM PROGRAMMING

- **ASSIGNMENT NO.** :- **2**

- **PROBLEM STATEMENT** :- Design of a 8086 Simulator/Assembler which supports different loop constructs in C Programming language (e.g., while, do-while, for) following the working principle of Two Pass Assembler .

- THEORY ANALYSIS :-

Two Pass Assembler :

A Two Pass Assembler typically goes through the source code in two passes:

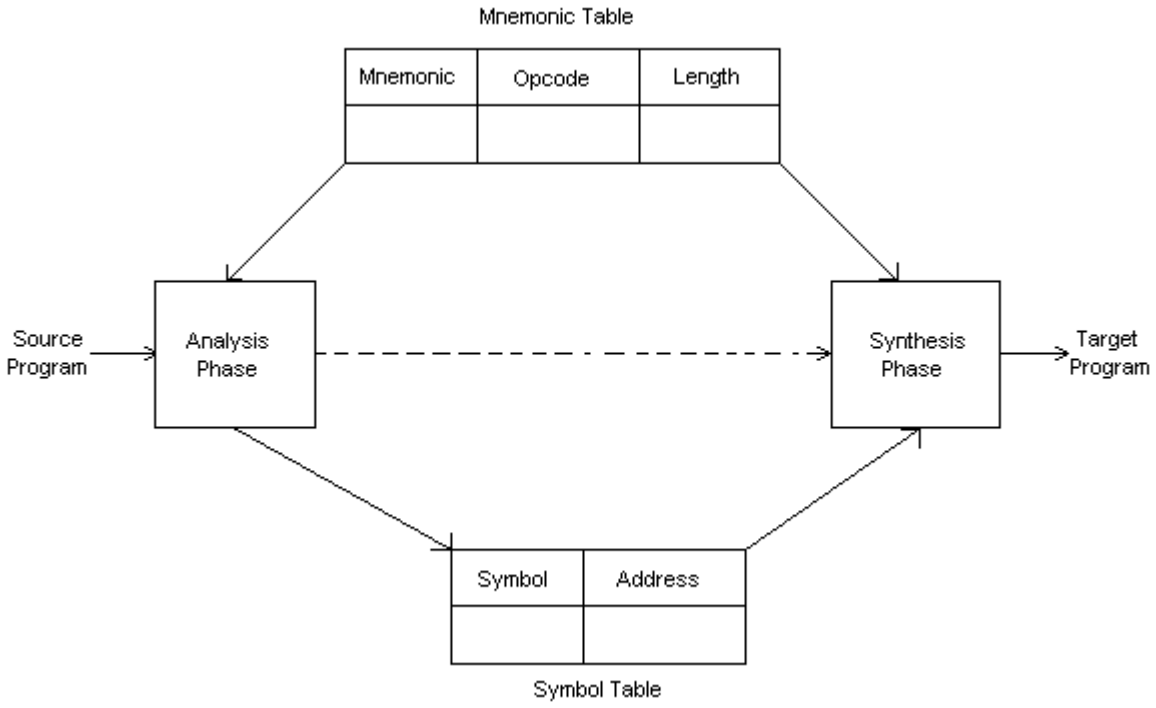
1. **First Pass**: In the first pass, the assembler reads the entire source code to collect information about labels and addresses. It generates a symbol table that includes the addresses of labels and variables.
2. **Second Pass**: In the second pass, the assembler generates machine code. It uses the symbol table from the first pass to replace labels and variables with their corresponding addresses.

Steps for Designing an 8086 Simulator/Assembler:

- **Lexical Analysis**: Tokenize the source code to identify symbols, labels, instructions, and operands.
- **First Pass**: Identify and store labels and their corresponding addresses. Calculate the size of the code and data sections. Generate a symbol table.
- **Intermediate Representation**: Create an intermediate representation that represents the instructions and operands in a form suitable for machine code generation.
- **Second Pass**: Generate machine code based on the intermediate representation. Replace labels and addresses using the symbol table. Resolve forward references.
- **Execution/Simulation**: Simulate the execution of the generated machine code on an 8086 virtual machine. Implement support for different loop constructs (e.g., while, do-while, for).

Supporting Loop Constructs:

- **Identify Loop Constructs**: Recognize loop constructs in the source code during the first pass.
- **Generate Intermediate Representation**: Represent loop constructs in the intermediate representation. Include information about loop conditions, loop bodies, and loop exits.
- **Machine Code Generation for Loops**: During the second pass, generate machine code for the identified loop constructs. Implement the necessary logic for loop execution.



- Here the “Analysis Phase ” denotes “First Pass” .
- The “Synthesis Phase ” denotes “Second Pass” .

IMPLEMENTATION :-

- FILE STRUCTURE :-

    1. **Progcap.asm** :- Here the assembly program is strored .Which is the input file for the assembly.c file.
    2. **Assembler.c** :- Here is the main logic of our program . first pass and Second pass function implementation is done . After running the program it will generate the "symbol Table" And the "Machine Code " for that particular assembly Language .

- CODES :-
    - Progcap.asm :-

```
.MODEL SMALL
.STACK 100H
.DATA
   PROMPT  DB  'The counting from 0 to 9 is : $'
.CODE
  MAIN PROC
    MOV AX @DATA              ; initialize DS
    MOV DS AX

    LEA DX PROMPT             ; load and print PROMPT
    MOV AH 9
    INT 21H

    MOV CX 9            ; initialize CX

    MOV AH 2                  ; set output function
    MOV DL 48                 ; set DL with 0

    @LOOP:                    ; loop label
     INT 21H                  ; print character
     INC DL                   ; increment DL to next ASCII character
     DEC CX                   ; decrement CX
    JNZ @LOOP                 ; jump to label @LOOP if CX is 0
```

```
    MOV AH 4CH                ; return control to DOS
```

```
    INT 21H
  MAIN ENDP
END MAIN
```

- Assembler.c :-

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

#define MAX_LINES 100
#define MAX_LINE_LENGTH 256

typedef struct
{
    char label[32];
    int address;
} Symbol;

typedef struct
{
    char instruction[MAX_LINE_LENGTH];
    int address;
} MachineCode;

Symbol symbolTable[MAX_LINES];
MachineCode machineCode[MAX_LINES];
int symbolTableSize = 0;
int machineCodeSize = 0;

void firstPass(FILE *file)
{
    char line[MAX_LINE_LENGTH];
    char *token;

    // First pass: Collect information about labels and
addresses
    int address = 0;
    while (fgets(line, sizeof(line), file))
```

```
    {
```

```
        token = strtok(line, " \t\n");
        if (token && token[strlen(token) - 1] == ':')
        {
            // Label found
            token[strlen(token) - 1] = '\0'; // Remove the
colon
            strcpy(symbolTable[symbolTableSize].label, token);
            symbolTable[symbolTableSize].address = address;
            symbolTableSize++;
        }
        address++;
    }
}

void secondPass(FILE *file)
{
    char line[MAX_LINE_LENGTH];
    char *token;

    // Second pass: Generate machine code
    int address = 0;
    while (fgets(line, sizeof(line), file))
    {
        token = strtok(line, " \t\n");
        if (token && token[strlen(token) - 1] == ':')
        {
            // Label found, skip
        }
        else if (token)
        {
            strcpy(machineCode[machineCodeSize].instruction,
line);
            machineCode[machineCodeSize].address = address;
            machineCodeSize++;
        }
        address++;
    }
}

int main()
{
    FILE *inputFile = fopen("./progcap.asm", "r");
    if (!inputFile)
    {
        perror("Error opening file");
```

```
        return EXIT_FAILURE;
    }

    // Two-pass assembler
    firstPass(inputFile);
    rewind(inputFile);
    secondPass(inputFile);

    printf("Symbol Table:\n");
    for (int i = 0; i < symbolTableSize; i++)
    {
        printf("%s: %04X\n", symbolTable[i].label,
symbolTable[i].address);
    }

    // Display machine code
    printf("Generated Machine Code:\n");
    for (int i = 0; i < machineCodeSize; i++)
    {
        printf("%04X: %s ", machineCode[i].address,
machineCode[i].instruction);
    }

    fclose(inputFile);
    return 0;
}
```

- Output :-

```
Microsoft Windows [Version 10.0.19045.3693]
(c) Microsoft Corporation. All rights reserved.

E:\system prog project\assignment_2>cd "e:\system prog project\assignment_2\" &&
gcc assembler.c -o assembler && "e:\system prog project\assignment_2\"assembler
Symbol Table:
@LOOP: 0012
Generated Machine Code:
0000: .MODEL 0001: .STACK 0002: .DATA 0003:    PROMPT 0004: .CODE 0005:    MAIN
0006:    MOV 0007:    MOV 0009:    LEA 000A:    MOV 000B:    INT 000D:    MOV
000F:    MOV 0010:    MOV 0013:    INT 0014:    INC 0015:    DEC 0016:    JNZ
0018:
MOV 0019:    INT 001A:    MAIN 001B:  END
```