Sanjivani Rural Education Society's

SANJIVANI COLLEGE OF ENGINEERING,

KOPARGAON - 423 603

(An Autonomous Institute)

CERTIFICATE

Department ECE *Subject* _____

Roll No. 73. *Univ. Seat No.* UEC20E2009

Year 2023-24 *Sem I / II* II

This is to certify that Mr./Miss. Wakte Sakshi Vijay

has satisfactorily completed _____ *out of* _____ *experiments*

in the Practical / Term work prescribed by Savitribai Phule Pune University, Pune in

_____ *Laboratory.*

STAFF MEMBER INCHARGE **HEAD OF DEPT.** **DIRECTOR**

Page No.
Date

Aim:
1. Study of lexical Analyzer.
2. Write C Program to implement lexical Analyzer for simple arithmetic operation involving equal to ε) P arithmetic operators (+, -) Expected o/p to create
   i) Identifier Table.
   ii) Literal Table
   iii) Symbol Table
   iv) Uniform Symbol Table.

Learning Objectives:

Study of lexical Analyzer.

Equipment Required:

PC (personal Computer), Turbo C software. etc

Theory:
a. lexical Phase of Compilation.
b. Figure for Interaction compiler phases.
c. LEX and YAcc Compiler.

a. lexical Phase of Compilation:

Lexical Analysis identifies the lexical unit in a source statement It then classifies the units into different lexical classes, eg. ids, constant

Sanjivani Rural Education Society's

SANJIVANI COLLEGE OF ENGINEERING,

KOPARGAON - 423 603

(An Autonomous Institute)

Department ECE Engineering

PRACTICAL / TERM-WORK REPORT

Subject SPOS

Name of the Student Wakte Sakshi Vijay

Class TY ECE Division -

Batch No. T3 Roll No. 73

Name of the Experiment / Title of Term Work Study of lexical Analyzer

Experiment / Term - Work No. 01.

Conducted on : 04/03/2024

Expected date of Submission : 11/03/2024.

Actual date of Submission : 11/03/2024.

Checked by Prof. N. Y. Siddiqui.
(Name of the Teacher)

Remarks 09/10

Signature 11/03/24 Date

Sanjivani Rural Education Society's

SANJIVANI COLLEGE OF ENGINEERING, KOPARGAON

(An Autonomous Institute)

_____ Laboratory Journal

Name Wakte Sakshi Vijay Class TYECE Roll No. 73

Examination Seat No. UEC22E2009 Year _____

❖ INDEX ❖

Sr.No.	Title of Experiment	Date of Conduction	Page No.	Remark / Initial of Faculty
01	Study of lexical Analyzer	04/03/2024		
02	Design of PassI of 2 pass			
	assembler for subset of 8086	11/03/24		
03	Design of PassII of 2	11/03/24		
	pass assembler.			
04	Design of a MACRO PassI	18/03/24		
05	Design of MACRO	25/03/24		
	PassII			
06	Implement of Bankers	01/04/24		
	Alg. for Deadlock Detecti			
	& Aviodance			
07	Implementation Page	08/04/24		
	Replacement alg.			
	1. FIFO 2. LRU			
08.	Write Interactive Shell	15/04/24		
	program on UNIX/LINUX			
09	Implement Job	29/04/24		
	scheduling Alg. FIFO,			
	Shortest path First, RR			
10	Disk scheduling			
	Algorithm for FCFS.			

reserved id's, etc. and enters them into different tables. This classification may be based on the nature of a string or on the specification of the source language. (For example, while an integer constant is a string of digits with an optional sign, a reserved id is an id whose name matches one of the reserved names mentioned in the language specification). Lexical analysis builds a descriptor called a token. For each lexical unit. A token contains two fields class code, and number in class. Class code identifies the class to which a lexical unit belongs number in class is the entry number of the lexical unit in the relevant table. We depict a token as code #no, eg Id #10.
The IC for a statement is thus a string of tokens.

Ex: The statement a := b + i; is represented as the string of tokens.

Id #2	OP #5	Id #3	OP #5	Id #1	OP # 10

where Id #2 stands for 'identifier occupying entry #2 in the Symbol table', ie the id a. OP #5 stands for the operator ':=', etc.

Page No.
Date

```
/* Program for lexical Analyser */

#include <stdio.h>
#include <string.h>
 int main()
{
 int i=0, j=0, x=0, t=0, ss=0, ss_print
[15], kw_print=0, ss_flag=0, kw_flag=0,
 Wrdfnd=0, key_print[10], idn_pri[10];

char ch, kw[10], mac;[15]
[15], spl_symb[] = {'#', '+', '-', '*', '/',

 '>', '<', '(', ')', ',', ';', '{', '}'};

char keywrds[10]
 [10] ={"include", "void", "main",

        "stdio.h", "canio.h", "getch",
        "int", "char", "float"};

FILE *in;
  in = fopen("test.c", "r);
 for (i=0; i<15; i++)
    {
        ss_print[i]=0;
        key_print[i]=0;
        idn_pri[i]=0;
        maci[i][0]=0;
    }
```

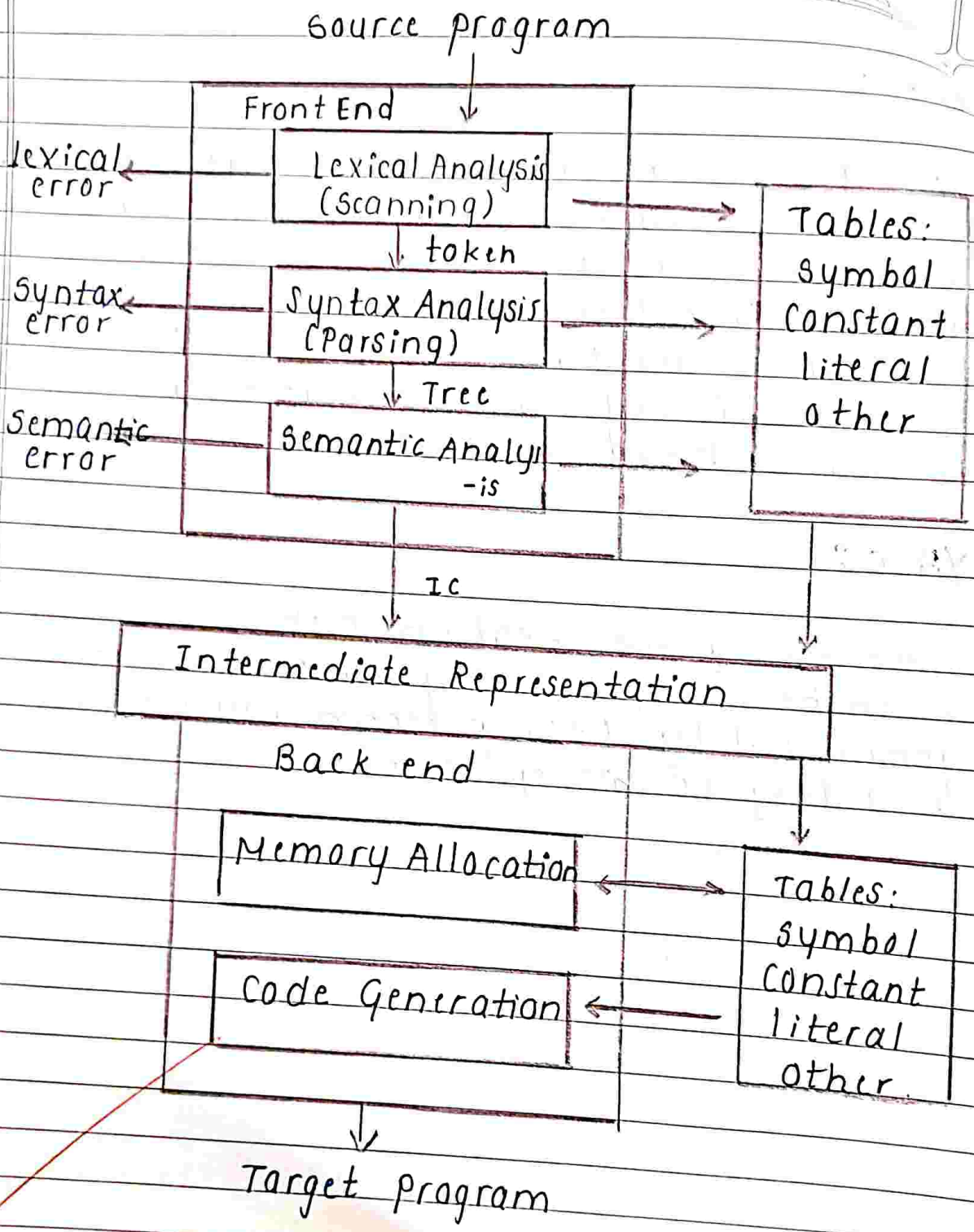


fig: Toy Complier.

Page No.
Date

* LEX :

LEX accepts an inputs specification which consists of two component. The first component is a specification of string representing the lexical units in L eg. id's and constants. The second component is a specification of sematic action aimed at building an IR.

YACC :

Each string specification in the input to YACC resembles a grammer production. The parser generated by YACC perform reduction according to this grammer.

```
        x++;
      }
    }
  }
  j=0;
}
}while (ch != EOF);
printf ("\n Symbol Table \n");

for (i=0; i<14; i++)

printf ("%d \t %c \t %d \n", i, spl_
        symb[i], ss_print[i]);

printf ("no of special symbol %d \n
   Uniform Symbol Table \n", ss);

for (i=0; i<14; i++)

 if (ss_print[i] != 0)

printf ("%d \t %c \t \n", i+1,
    spl_symb[i]);

printf ("\n keyword Table \n");

for (i=0; i<6; i++);
 if (key_print[i] != 0)
```

Page No.
Date

Conclusion:

From this experiment we study the Lexical analysis, now we are able to do lexical analysis and segregate the token or Symbol.

Page No.
Date

```
#include <stdio.h>
#include <conio.h>
   void main()
 {
   int a,b;
   float rat;
    char c;
    c = a+b;
    rat = b-a;
    getch();
 }
```

Symbal Table

0	#	2
1	+	1
2	-	1
3	*	0
4	/	0
5	>	2
6	<	2
7	(	2
8	)	2
9	=	2
10	,	1
11	;	6
12	{	1
13	}	1

no of Special symbal 123

Uniform Sym Table

1	#
2	+
3	-
6	>
7	<
8	(
9	)
10	=
11	,
12	;
13	{
14	}

keyward Table

1	2	include
2	1	void
3	1	main
4	1	stdio.h
5	1	conio.h
6	1	getch

Data Types table

1	1	int
2	1	char
3	1	float

Identifier Table

1	a	2
2	b	2
3	rat	1
4	c	1

```
printf("%d \t %d \t %s \n", i+1,
    key_print[i], keywrds[i]);

printf ("\n Data Types table \n");

for (i=6 ; i<9; i++)

  if (key_print [i] )=0)
printf("%d \t %d \t %s \n", i-5,
  key_print [i], keywrds [i]);

printf ("\n Data Types table \n");
for (i=6 ; i<9; i++)

  if (key_print ("%d \t %d \t %s \n, "
      i-5, key_print [i], keywrds[i]);

printf ("\n Identifier Table \n");

for (i=0; i<10; i++)

  if (idn_pri[i] ! =0)

  printf ("%d \t %s \t %d \n", i+1,
      mac1[i], idn_pri [i])

  false (in);
  return;
}
```

Sanjivani Rural Education Society's

SANJIVANI COLLEGE OF ENGINEERING,

KOPARGAON - 423 603

(An Autonomous Institute)

Department ECE. Engineering

PRACTICAL / TERM-WORK REPORT

Subject System Programming and Operating System.

Name of the Student Wakte Sakshi Vijay.

Class TY ECE Division -

Batch No. T3 Roll No. 73

Name of the Experiment / Title of Term Work Design of Pass I of two pass assembler for a subset of 8086.

Experiment / Term - Work No. 2.

Conducted on : 11/03/24

Expected date of Submission : 18/03/24

Actual date of Submission : 11/03/24

Checked by Prof. N. Y. Siddiqui
(Name of the Teacher)

Remarks 08/10

Signature 11/03/24 Date

Page No.
Date

Assignment NO-02

Aim :
Design of PASS I of two pass assembler for a subset of 8086.

Learning Objectives :
Implementation and working principle of Assembler.

Equipment Required :

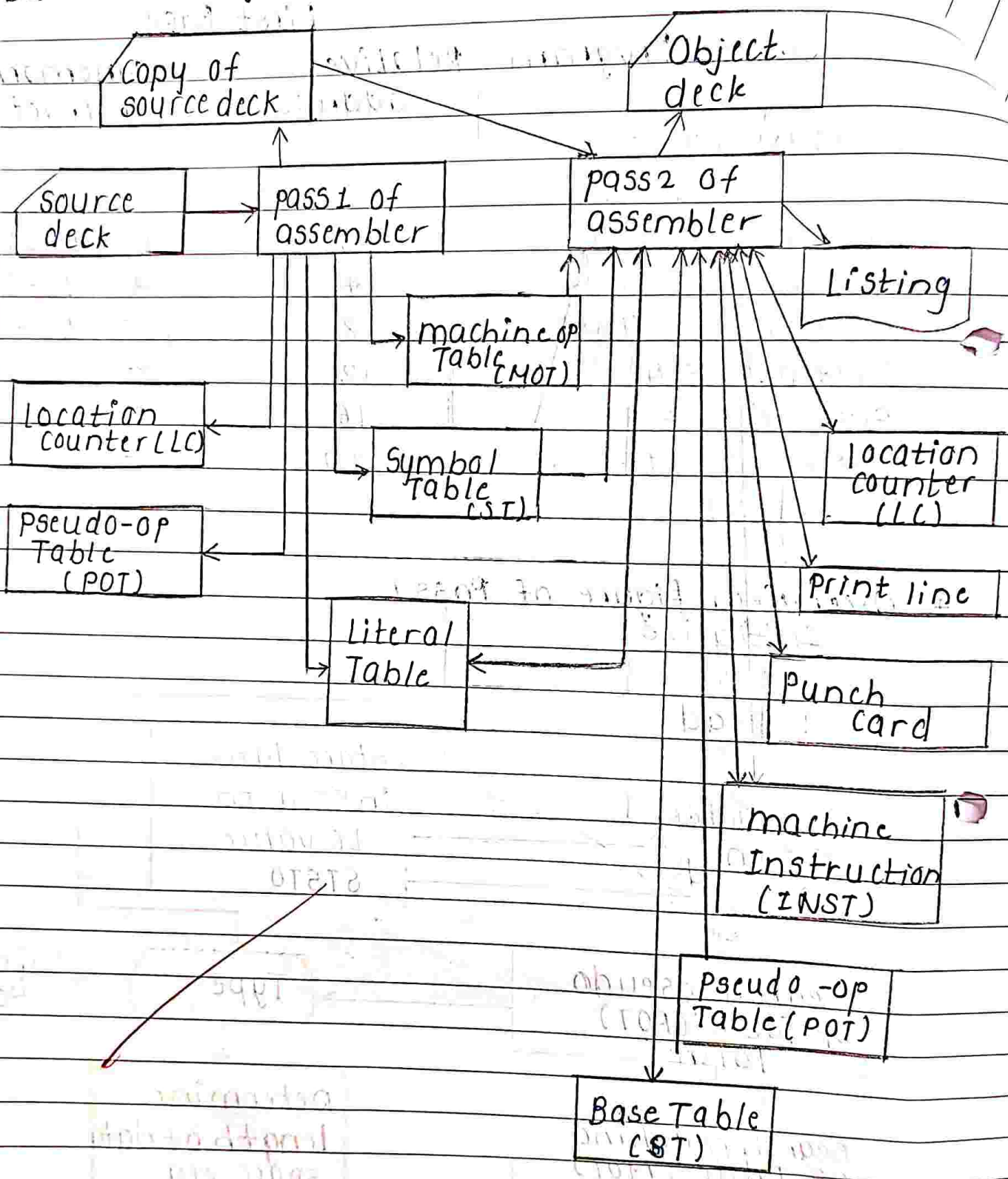
PC (Personal Computer), Turbo c software, etc.

Theory :

1. What is Assembler ?
An Assembler is a type of computer Program that takes in basic instruction and convert them into a pattern of bits that the computers processor can use to perform basic operation.
The assembly language code into machine code that the computer can then read and execute

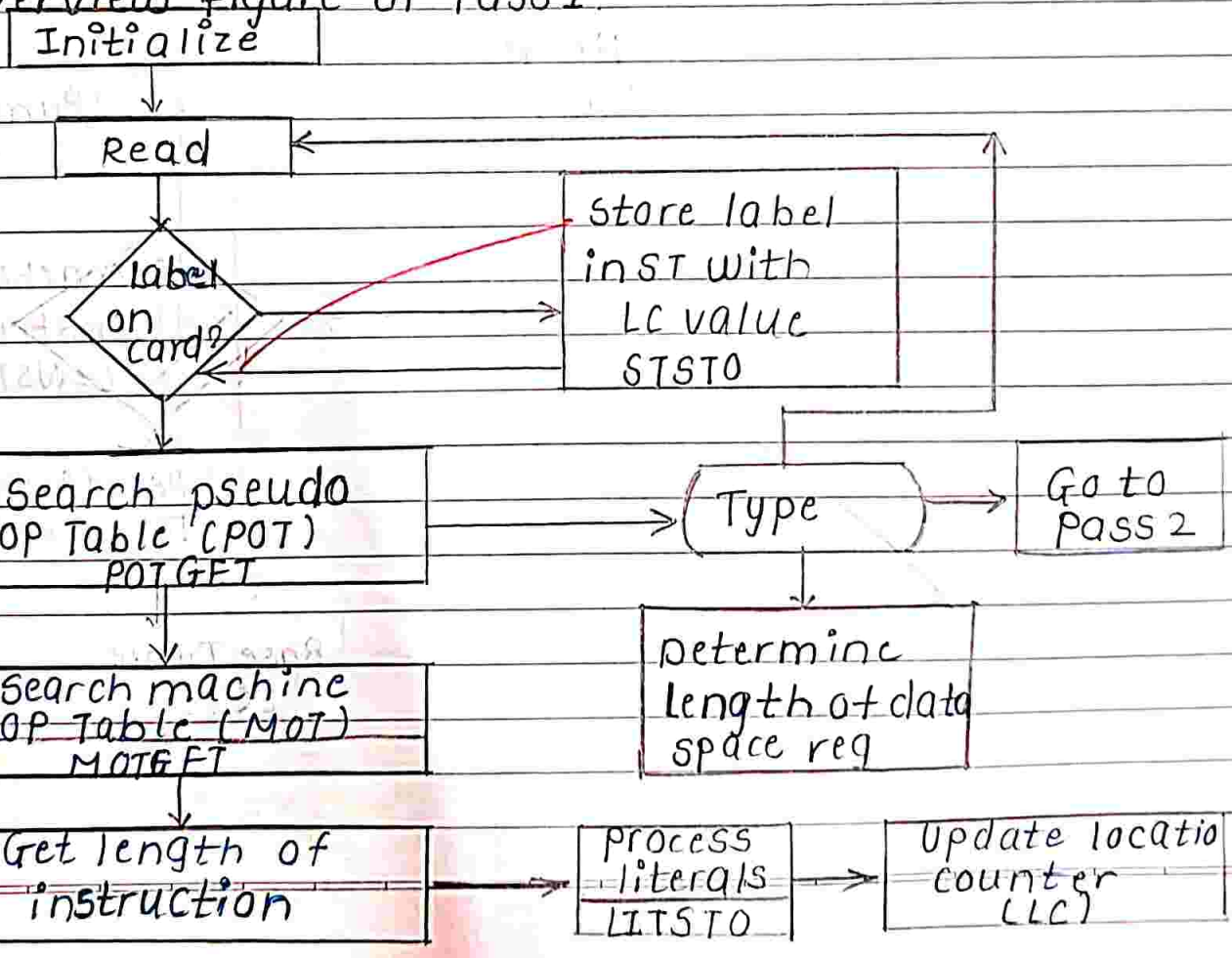
2. General Design Purpose :
Assembler is a low-level programming language designed to match the architecture of a computer central processing unit. Its primary purpose is to provide a human-readable Representa

5. Data bases by Assembler Phase Figure.



Source Program	First Pass	
	Relative address.	Mnemonic instruction.
JOHN START 0		
USING *, 15		
L 1, FIVE	0	L 1, -(0, 15)
A 1, FOUR	4	A 1, -(0, 15)
ST 1, TEMP	8	ST 1, -(0, 15)
FOUR DC F'4'	12	4
FIVE DC F'5'	16	5
TEMP DS 1F	20	-
END		

4. Overview figure of Pass 1.



of machine code instruction, making it easier for programmers to write code that directly correspond to the computer hardware. It help bridge the gap between high-level programming language & machine code.

3. Design of Assembler?

The most important things which to be concentrated is the generation of symbol table and resolving forward reference

- Symbol table:
  - This is created during pass 1.
  - All the labels of the instructions are symbol
  - Tables has entry for symbol name, address value.

- Forward Reference:
  - Symbol that are defined in the later part of the program are called Forward reference
  - There will not be any address value for such symbols in the symbols table in Pass 1.

Page No.
Date

Program:

```
#include <stdio.h>
 char mot[6][6] =
 {"STOP","ADD","SUB","MUL","MOVER","MOVEM"};
   // mnemonics.

 char r[4][6] ={"AREG","BREG","CREG","DREG"};
    // registers.

 Struct1
    {
      Char name;
     // name of literal
       int add;
     // address of literal
   } list[20];
    struct s
     {
       char name[20];
      // name of symbol
            int add;
      // address of symbol
     } sy[20];

   void main()
   {
       char st[10];
          int lc, n1=0, ns=0, i, j=0, k=0,
     litflag=0, m=0, tempb, flag=0;
```

1. macro prototype statement.
2. One or 2 model statement
3. Macro preprocessor statement.

- A Macro call consist of a name optionally followed by an actual parameter list. The number of parameter in the actual parameter list must be the same as the number of formal parameter specified in the defination of macro If the macro has no formal parameter list, its call must have no actual parameter list.

<macro Name> [& <Actual Parameter Name> [< Actual Parameter kind >].

- The macro expansion can contain load instructions that load parameter in parameter registered and it can contain instruction that load parameter register from register loaded by processor. The processor can also load parameter directly

1. Lexical Expansion / substitution.
2. Flow of control during expansion.

Page No.
Date

6. Data structure in brief:

LOCCR :
It is location counter that maintain tracks machine addresses of symbolic tables
- Initialized with START.
- Raised for all instruction.
- Pseudo instruction are BYTE, WORD, RESB and RESW.
- Machine Instruction.

OPTAB:

In this operation table mnemonic operation code and have its machine language equivalent

ADD	18
LDS	6C

OPTAB can be implemented by using hashing function for fast access.

SYMTAB:

- In symbol table symbols label operand and their consequent machine are keeps address.
- SYMTAB is dynamic, constructed throughout Pass 1:
- SYMTAB can be implemented by using hashing function for fast access.

```
FILE *fip, *fir;
    clrscr();
fip = fopen("input.asm", "r");

// assembler input
   fir = fopen("ir.dat", "w");

// Intermediate Representation
    fscanf(fip, "%s", st);

// read input
        while(!feof(fip))
        {

        up: if(!strcmp(st, "START"))
// if start copy address (lc)
  {
    fscanf(fip, "%d", &lc);
fprintf(fir, "%s%d %s %d\n", "AD", 1,
          "C", lc);
  }
  fscanf(fip, "%s", st);

// read next
    for(i=0; i<6; i++)
//check if it present in mnemonics tabl
    {
```

Page No.
Date

```
strcpy (sy[ns].name,st);
fprintf (fir,"%s %d","S",ns);

    ns++;
    break;
    }
  }
}

//if it is not there is not then check
for symbol definition [DC/DS]

for (m=0;m<ns;m++)
  {
  if(!strcmp(st,sy[m].name))
  {
    fscanf(fip,"%s",st);

    if(!strcmp(st,"DS"))
  {

  fscanf(fip,"%d",&tempb);
  fprintf(fir,"%s %d %d \n","DL",2,
          tempb);
  sy[m].add=lc;
  lc+=tempb;
  goto up;
  } else
```

Page No.
Date

```
// else add new entry in Literal table
   if (litflag==0)
   {
    lit[n1].name = st[k];
    fprintf(fir, "%s %d", "L", n1);

    n1++;
   }
   break;
   }
   else
   // if it is not literal it must be
      symbol
   {

   for(j=0; j<ns; j++)
    {
     // check if said symbol is present
        if symbol table

   if (!strcmp(sy[j].name, st))
   {
     flag=1;
   fprintf(fir, "%s %d\n", "s", j);
    lc++;
    goto up;
   }

   // else add new entry in symbol table
      if (flag==0) {
```

Page No.
Date

```
if (!strcmp (st, mot [i]))
 {
  fprintf (fir,"%.s%.d", "IS", i+1);
          // found in mot

 fscanf (fip, "%.s", st);
  for (j=0 ; j<4; j++)      // now find register

 if (!strcmp (st, r[j]))
  fprintf (fir, "%.d", j+1);
  fscanf (fip , "%.s", st);

  if( !strcmp (st,"= "))
    // if = means literal else
    {
       k=1;

  fscanf (fip , "%.s", st);
  // check if said literal already present
    in literal table.

  for (j=0; j<n1; j++)
   if (!strcmp (lit [j]. name, st [k]))
   {

     litflag =1;

    fprintf(fir , "%.s %.d ", "L", j);
}
```

Page No.
Date

Output:

DATABASE

SYMBOL TABLE
SYMBOL ENTRY

SYMBOL	ADDRESS	
0	B	0
1	A	108

LITERAL TABLE
LITERAL ENTRY
LITERAL ADDRESS

0	112

ir.dat

AD1C100
IS5150
IS6151
IS2110
IS3111
IS6150
DL 15
DL 23
AD 1 C1
AD 2 C2

* Conclusion:-
From this ~~conclusion~~ experiment we understand Design of Pass-I of assembler for a subset of 8086.

```
printf("\n DATABASE:-");
printf("\n SYMBOL TABLE \n SYMBOL
        ENTRY \t SYMBOL \t ADDRESS \n");

for(i=0;i<ns;i++)

printf("\n %d \t %s \t %d", i, sy[i].
        name, sy[i].add);

printf("\n LITERAL TABLE \n LITERAL
 ENTRY \t LITERAL \t ADDRESS \n");

for (i=0 ; i<nl; i++)

printf("\n %d \t %c \t %d", i, lit[i].
   name, lit[i].add);

    getch();
    fclose(fir);    // close files opened
    fclose(fip);
}
```

Program : code I.

```
START 100
MOVER AREG B
MOVEM AREG A
ADD AREG ='1'
SUB AREG ='2'
MOVEM AREG B
 B  DC  5
 A  DS  3
 END
```

Page No.
Date

```
if (!strcmp(st,"DC"))
{
  fscanf(fip,"%d",&tempb);
  fprintf(fir,"%s%d%d\n","DL",tempb);
   sy[m].add=lc;
    goto up;
}
}
}
}
//check for END of assembly statements

if (!strcmp(st,"END"))
    {
      for(i=0;i<n1;i++)
        {
fprintf(fir,"%s%d%s%c\n","AD",2,
    "C",lit[i].name);
  lit[i].add=lc;
    lc++;
}
  break;
}
  lc++;
  fprintf(fir,"\n");
}
//print databases.
```

Sanjivani Rural Education Society's

SANJIVANI COLLEGE OF ENGINEERING,

KOPARGAON - 423 603

(An Autonomous Institute)

Department ECE Engineering

PRACTICAL / TERM-WORK REPORT

Subject SPOS

Name of the Student Wakte Sakshi Vijay

Class TY ECE Division

Batch No. T3 Roll No. 73

Name of the Experiment / Title of Term Work Design of Pass II of two pass assembler for a subset of 8086.

Experiment / Term - Work No. 03

Conducted on : 11/03/24

Expected date of Submission : 18/03/24

Actual date of Submission : 03/05/24

Checked by Prof. N. Y. Siddiqui
(Name of the Teacher)

Remarks 09/10

Signature Date

DATE:

Assignment NO- 03.

Aim:
Design of Pass II of two pass Assembler for a subset of 8086.

Learning Objective:
Implementation and working principle of Assembler.

Equipment Required:
PC (Personal Computer), Turbo C software etc

Theory:
1. What is Assembler?
An Assembler is a type of computer Program that takes in basic instruction and convert them into a pattern of bits that the computer processor can use to perform basic operation. The assembly language code into machine code that the computer can then read and execute.

Algorithm: Assembler Second Pass:
1. code_area_address := address of code_area;
   pooltab_ptr := 1;
   loc_cntr := 0;
2. While next statement is not and END statement
a. Clear machine_code_buffer;

DATE:

```
i++; // skip the next line since it's the
address.
} else if (strcmp(code[i], "LABEL") == 0)
{
  strcpy(symbol_table[symbol_table_size].
  label, code[i+1]);

symbol_table[symbol_table_size].location
  = location_counter;
    symbol_table_size++;
    i++;
  } else {
      location_counter++;
    }
  }
  printf("Symbol Table after Pass 1:\n");

for(int i=0; i<symbol_table_size; i++) {
    printf("%-s\t%d\n", symbol_table[i].
  label, symbol_table[i].location);
  }
}
void second_pass(char code[]
[MAX_LABEL_LEN], int n) {
  printf("Machine code:\n");
  for (int i=0; i<n; i++) {
    if (strcmp(code[i], "LABEL") == 0 ||
    strcmp(code[i], "ORG") == 0) {
```

DATE:

3. (Processing of END statement)
a) Perform step 2 (b) and 2 (f)
b) Write a code_area into output file

Program:

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

#define MAX_LINES 100
#define MAX_LABEL_LEN 10

typedef struct{
 char label[MAX_LABEL_LEN];
  int location;
} SymbolTableEntry;

SymbolTableEntry
symbol_table [MAX_LINES];
int symbol_table_size=0;

void first_pass (char code[]
[MAX_LABEL_LEN],int n){
  int location_counter =0;
  for (int i =0; i<n; i++){
  if (strcmp (code[i], "ORG")==0){
     location_counter = strtol (code[i+1],
     NULL, 16);
```

DATE:

b) If an LTORG statement
i) Process literals in LITTAB [POOLTAB] pooltab_ptr).
LITTAB [POOLTAB [pooltab_ptr+1]]-1 similar to processing of constant in DC statement i.e assembler the literal in machine-code buffer
ii) size := size of memory area required for literals.
iii) pooltab_ptr := pooltab_ptr+1;

c) If a START or ORIGIN statement then
i) loc_cntr := value specified in operand field;
ii) size := 0;

d) If a declaration statement
i) If a DC statement then
Assemble the constant in machine_code_buffer
ii) size := size of memory are required by DC/DS;

e) If an imperative statement.
i) Get operand address from SYMTAB or LITTAB
ii) Assembler instruction in machine_code_buffer
iii) size := size of instruction;

f) If size ≠ 0 then
i) Move content of machine_code_buffer to the address code_area_address + loc_cntr;
ii) loc_cntr := loc_cntr + size;

Sanjivani Rural Education Society's

SANJIVANI COLLEGE OF ENGINEERING,

KOPARGAON - 423 603

(An Autonomous Institute)

Department ECE. Engineering

PRACTICAL / TERM-WORK REPORT

Subject SPOS.

Name of the Student Wakte Sakshi Vijay

Class TY ECE Division -

Batch No. T3 Roll No. 73.

Name of the Experiment / Title of Term Work Design of a MACRO Pass - I

Experiment / Term - Work No. 04.

Conducted on : 18/03/24

Expected date of Submission : 25/03/24

Actual date of Submission : 15/4/24

Checked by Prof. N. Y. Siddiqui.
(Name of the Teacher)

Remarks 09/10

Signature 15/04/24 Date

DATE:

Conclusion:

from this experiment, we undstand design of Pass II of assembler for a subset of 8086.

DATE:

```
        i++;
    }else{
        int found = 0;
        for (int j=0; j< symbol_table_size; j++) {

    if (strcmp(code[i], symbol_table[j].label) ==
    0){
        printf("%d\n", symbol_table[j].location);
            found = 1;
            break;
        }
    }
        if (!found){
            printf("%s\n", code[i]);
    }
}
```

Input file

Ir.dat:

```
AD 1C 100
IS 51 S0
IS 61 S1
IS 24 L0
IS 32 L1
IS 63 S0
DL 1 5
DL 2 3
AD 2 C1
AD 2 C2
```

Database:

Symbol Table

0	B	105
1	A	106

Literal table

0	1	109
1	2	110

Output:

```
51 105
61 106
24 109
32 110
63 105.
```

Page No
Date

Assignment No-03

Aim : Design of a Macro Pass I.

Learning Objectives : Implementation and Principle of working of MACRO Assembler.

Equipments Required : PC (Personal Computer), Turbo C software, etc.

Theory :

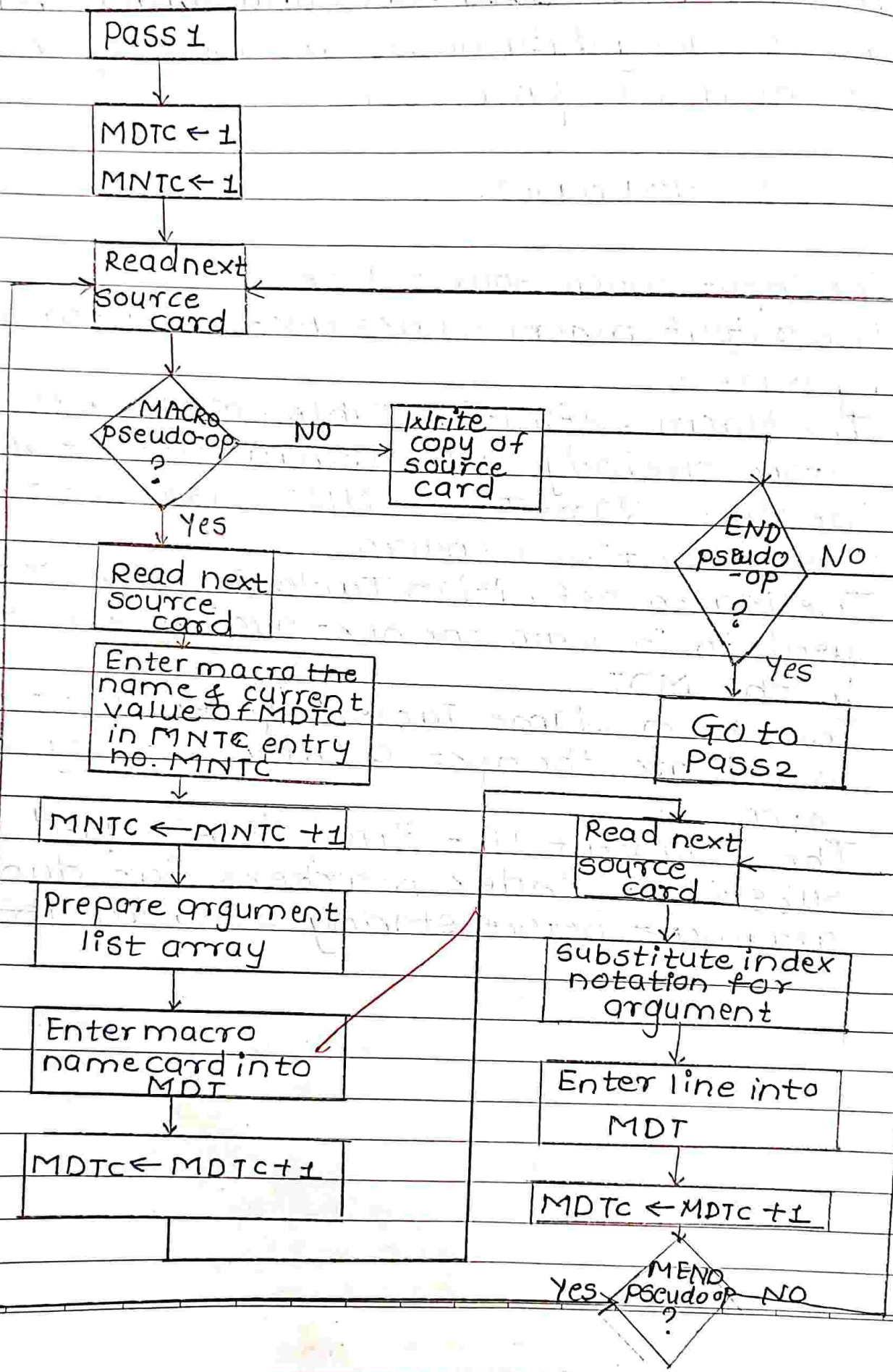
1 What is macro and Macro Processors?

→ A Macro is a unit of specification for program generation through expansion.
A Macro consist of a name, set of formal parameter and a body of code.
The use of macro name with a set of actual parameter is replaced by some code generated from its body. This is called Macro Expansion.

- Necessity:
  In case repetitive use of sequence of instruction.
- Either language processor built in feature or implemented as preprocessor.
- Two type of expansion based on place in Language processor.

1. Lexical : Simply replace string of character.
2. Semantic: Used when code with differ in no sequence & opcode of instruction.

4. Algorithm and flowchart for Pass 1.



Page No.
Date

3. Specification of data bases and format for pass 1.

→ The following data bases are used by the pass 1 of the macro processor.

Pass 1 data bases:

1. The input macro source deck.
2. The output macro source deck copy for use by pass 2
3. The Macro Definition Table (MDT), used to store the body of the macro definition.
4. The Macro Name Table (MNT), used to store the name of defined macros
5. The Macro Definition Table Counter (MDTC) used to indicate the next available entry in the MDT.
6. The Macro Name Table Counter (MNTC), used to indicate the next available entry in the MNT.
7. The Argument List Array (ALA), used to substitute index markers for dummy argument before storing a macro definition.

Page No.
Date

2. Macro Instruction definition, call and Expansion in brief?

→ A Macro definition is enclosed between a macro header statement and a macro end statement.
- Macro definition are typically located at the start of a program.
- Macro definition consist of
1. A Macro prototype statement.
2. One or More Model statement.
3. Macro Preprocessor statement.
- The Macro prototype statement declares the name of a macro and the name and kinds of parameter.
- A Model statement is a statement from which an assembly language statement may be generated during macro expansion
- The Macro prototype Statement has following Syntax:

<macro name> [<formal parameter spec> [,..]]

where <macro name> appear in the mnemonic field of an assembly statement and <formal parameter spec> is form of & <parameter name> [< parameter kind>]

Macro call:
The Macro called by writing macro name in the mnemonic field of an assembly statement
<macro name> [<actual parameter spec>[,..]]

Page No.
Date

```
//check for start of macro or end.

fprintf(fir," %s",word); //if word is not
                          macro and end
fgets(word,20,fip); // paste it in IR
                       file.
if(!strcmp(word,"END\n"))
  {
    j=1;
    break;                //end, close all
  }
if(j!=1)
{
  fgets(word,20,fip);  // name of macro
  fprintf(fmnt,"%d%s",sno,word);
   sno++;
  fprintf(fmdt,"%d%s",mdno,word);
   mdno++;
  fgets(word,20,fip); // paste it in
                         mnt & mdt
}
while(strcmp(word,"MEND\n") &&
      (strcmp(word,"END\n")))
{
                          //paste until end of
                            macro
  fprintf(fmdt,"%d%s",mdno,word);
   mdno++;
   if(!strcmp(word,"END"))
```

Page No.
Date

```
A DS 5
B DC 10
C DC 15
END.
```

Program : code II.

```
#include <stdio.h>
#include <conio.h>
#include <string.h>
void main()
{
 int i,j=0,sno=1,mdno=1;
 char word[20];
 FILE *fip, *fmnt, *fmdt, *fir;
 clrscr();
 fip=fopen("input.dat","r");
                    //open input file.
 fmnt=fopen("mnt.dat","w");
                    //open mnt table.
 fmdt=fopen("mdt.dat","w");
                    //open mdt table.
 fir=fopen("ir.dat","w");
                    //Open Intermediate Represent-
                      ation file.
 while(!feof(fip))  // till the end if
                      input code.
 {
  fscanf(fip,"%s",word);
  while(strcmp(word,"MACRO")&&(strcmp
    (word,"END")))
```

Page No.
Date

Procedure:

- Open the editor window of "Turbo C" to write a code.
- After writing the code save it with ".c" extension Compiler the code with "Alt + F9". You must get error and warning "0". If you will get error or warning then remove that error or warning and again compile. After getting "0" error and warning to run the code press "ctrl + F9" See the output on output window.

Program : Code I.

```
MACRO ADDITION
 LOAD A
 ADD AREG B
 ADD AREG C
 MEND
 MACRO ADDITION1
 LOAD A
 ADD AREG B
 ADD AREG C
 MEND
 START
 MOVER AREG A
 MOVER BREG B
 ADDITION1
 STORE X
```

2. MNT:
It is used for recognizing macro name.

3. MDT:
It is used to perform macro EXPANSION

4. MDTP:
It is used to point to the index of MDT.
The starting index is given by MNT.

5. ALA:
It is used to replace the index notation by it actual value.

2. Algorithm & Flowchart for Pass II.

Pass 2 - Macro calls and Expansion:
The algorithm for pass 2 test the operation mnemonic of each input line to see if it is a name in the MNT. When a call is found, the call processor set a pointer, the Macro Definition Table pointer (MDTP), to the corresponding macro definition stored in the MDT. The initial value of the MDTP is obtained from the "MDT index" field of the MNT entry. The macro expander prepare the Argument List Array (ALA) consisting of a table of dummy argument indices & corresponding argument to the call. The list is simply a succession of symbols ordered to match the dummy argument on the name card

Page No.
Date

Assignment No-04.

Aim:
Design of a MACRO PASS-II.

Theory:

1. Specification of data bases and format for pass II

Pass 2 Data bases:
1. The copy of the input macro source deck.
2. The output expanded source deck to be used as input to the assembler
3. The Macro Definition Table (MDT), created by pass I
4. The Macro Name Table (MNT), created by Pass I.
5. The Macro Definition Table Pointer (MDTP), used to indicate the next line of text to be used during macro expansion.
6. The Argument List Array (ALA), used to substitute macro call argument for the index markers in the stored macro definition.

Database required for pass 2.

In Pass 2 we perform recognize macro call and perform macro expansion.

1. COPY FILE
It is a file it contain the output given form Pass 1.

Sanjivani Rural Education Society's

SANJIVANI COLLEGE OF ENGINEERING,

KOPARGAON - 423 603

(An Autonomous Institute)

Department ECE. Engineering

PRACTICAL / TERM-WORK REPORT

Subject SPOS.

Name of the Student Wakte Sakshi Vijay

Class TY ECE Division -

Batch No. T3 Roll No. 73.

Name of the Experiment / Title of Term Work Design of a MACRO Pass II

Experiment / Term - Work No. 05.

Conducted on : 25/3/24

Expected date of Submission : 1/04/24

Actual date of Submission : 15/04/24

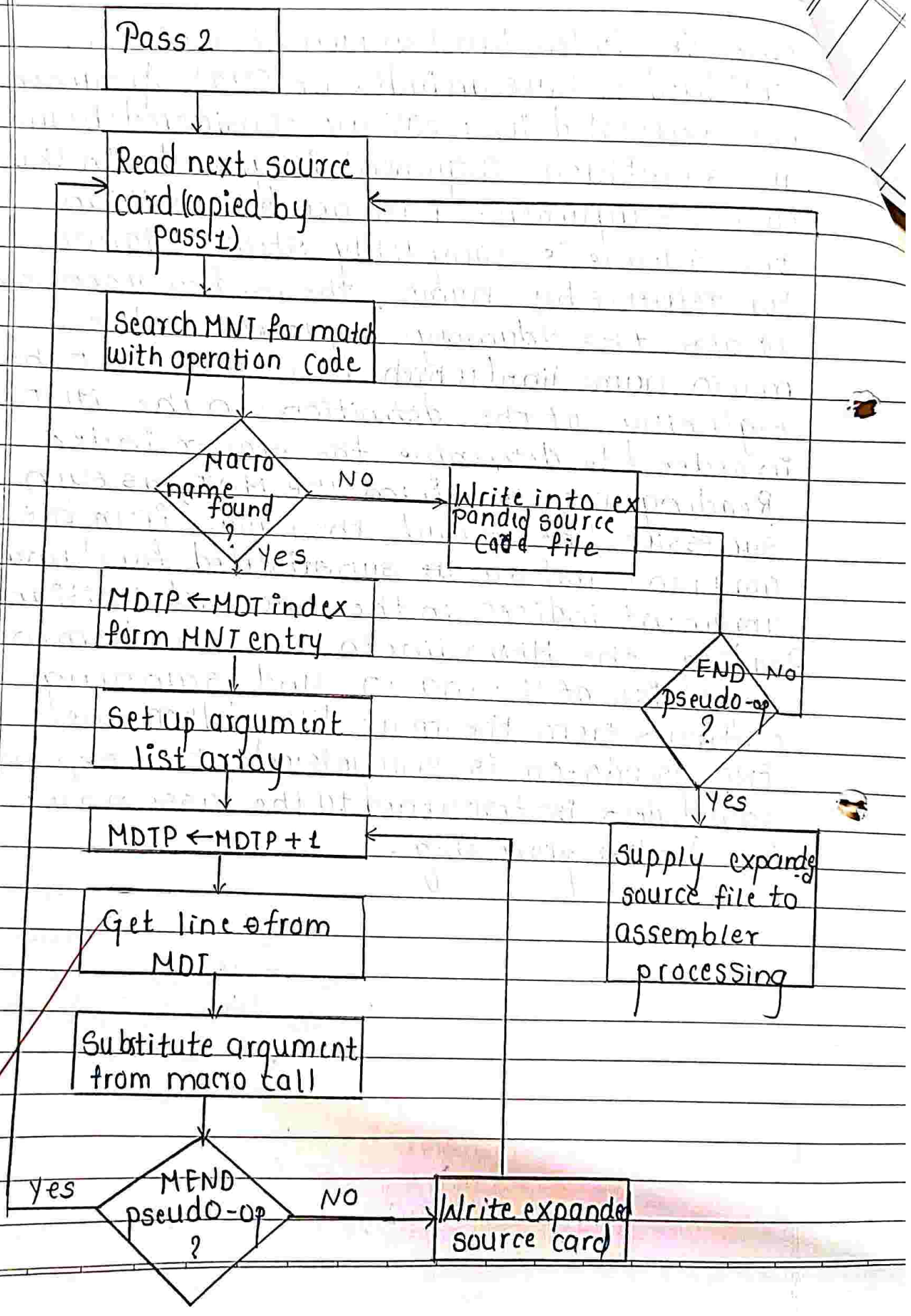
Checked by Prof. N. Y. Siddiqui (Name of the Teacher)

Remarks 09/10

Signature 16/04/24 Date

Page No.
Date

```
#include <stdio.h>
void main()
{
   int j=0,k;
   char mname[10],word[20];
   FILE *fop,*fmnt,*fmdt,*fir;
   fop=fopen("output.dat","w");
    //open output
     fmnt=fopen("mnt.dat","r");
    //open mnt table
     fmdt =fopen("mdt.dat","r");
     //open mdt table
     fir=fopen("ir.dat","r");
   //Intermediate representation
     while(!feof(fir))
     {
      rewind(fmnt);
      //restart mnt table
      fgets(word,20,fir);
      //read instruction
      fscanf(fmnt,"%d",&k);
      //ignore no.
      while(!feof(fmnt))
      {
       fgets(mname,20,fmnt);
       if(!strcmp(word,mname))
        {
          j=1;
          break;
```


Pass 2
Read next source card (copied by pass 1)
Search MNT for match with operation code
Macro name found ?
NO
Write into expanded source code file
Yes
MDTP ← MDT index form MNT entry
Set up argument list array
MDTP ← MDTP + 1
Get line of from MDT
Substitute argument from macro call
MEND pseudo-op ?
yes
NO
Write expanded source card
END pseudo-op ?
No
Yes
Supply expanded source file to assembler processing

(the first is the label argument, which is considered to have an index of zero). Argument not represented in a call are considered blank, and superfluous argument is ignored. In the case of argument reference by position the scheme is completely straight forward For reference by name, the macro processor locates the dummy argument on the macro name line (which is available at the beginning of the definition in the MDT) in order to determine the proper index. Reading proceeds from the MDT, as each successive line is read, the values from the argument list are substituted for dummy argument indices in the macro definition. Reading the MEND line in the MDT terminate expansion of the macro, and scanning continues from the input file. When the END pseudo-op is encountered, the expanded source deck is transferred to the assembler for further processing.

Sanjivani Rural Education Society's

SANJIVANI COLLEGE OF ENGINEERING,

KOPARGAON - 423 603

(An Autonomous Institute)

Department ECE Engineering

PRACTICAL / TERM-WORK REPORT

Subject SPOS.

Name of the Student Wakte Sakshi Vijay

Class TY ECE Division -

Batch No. TY Roll No. 73

Name of the Experiment / Title of Term Work Implement of Bankers Algorithm for Deadlock Detection & Aviodance

Experiment / Term - Work No. 06.

Conducted on : 01/04/24

Expected date of Submission : 08/04/24

Actual date of Submission : 03/5/24

Checked by Prof. N.Y. Siddiqui
(Name of the Teacher)

Remarks 09/10

Signature 3/5/24 Date

IR

```
START
Mover Areg A
mover Breg B
Addition
A DS 5
B DC 10
C DC 15
END
```

MNT

SrNO	MACroName
1	ADDITION
2	ADDITION1

output

```
START
Mover Areg A
Mover Arg B
Load A
Add Arg B
Add Arg C
A DS 5
B DC 10
```

Conclusion:

From this experiment we understand Design of a MACRO pass II. How the database are compare and opcode are shown to the output.

Page No.
Date

```
//found macro call
{
fscanf (fmnt, "%d", &k);
}
   if (j==1)
   {
      j=0;
fscanf (fmdt, "%s", mname);
while (strcmp (word, mname))
fgets (mname, 20, fmdt);
fscanf (fmat, "%d", k);
}
fgets (word, 20, fmdt).
while (strcmp (word, "MEND\n"))
fprintf (fop, "%s", word);
fscanf (fmdt, "%d", &k);
fgets (word, 20, fmdt);
   }
   }
   else
   fprint(fop, "%s", word);
   }
      printf ("action taken");
        getch ();
}
```

Input file

MDT:

ADDITON	MEND
load A	ADDITION
Add Areg B	load a
Add Areg C	Add Areg B
	Add Areg C

DATE:

Banker algorithm for single resource

A	0	6	A	1	6	A	1	6
B	0	5	B	1	5	B	2	5
C	0	4	C	2	4	C	2	4
D	0	7	D	4	7	D	4	7
	(a)			(b)			(c)	

Three resource states:
- safe
- safe
- unsafe.

Banker Algorithm for multiple Resource?

process	tapedrive	plotter	scanner	CDROMS
A	3	0	1	1
B	0	1	0	0
C	1	1	1	0
D	0	1	0	1
E	0	0	0	0

process	tapedrive	plotter	scanner	CDROMS
A	1	1	0	0
B	0	1	1	2
C	3	0	0	0
D	0	0	1	0
E	2	1	1	0

Resource assigned

Resource still needed

E = (6342)
P = (5322)
A = (1020)

DATE:

4. Circular Wait:
We have make circular wait false to pervent the deadlock
To remove the circular wait just give the numbering to all resource
CPU - 1
Printer - 2
Scanner - 3

Deadlock Detection and Avoidance in breif.

Deadlock Detection:
- Method by which the occurance of deadlock the processes & resource involved are detected
- Generally work by detecting a circular wait
- The cost of detection must be considered
- One method is resource allocation graph

Deadlock Avoidance:
- Simplest and most useful model require that each process declare the maximum no. of resource of each type that it may need.
- The Deadlock avoidance alg. dynamically examine the resource allocation state to ensure that there can never be a circular-wait condⁿ

DATE:

Assignment No-06.

Aim:
Implement of Bankers Algorithm for Deadlock Detection and Avoidance.

Theory.

1. What is Deadlock?

Deadlock is a situation where computer process wait for resource which being assign to some another process.

Process is deadlock when it is waiting on an event which will never happen System is Deadlocked when one or more process is deadlocked.

Necessary Condition for deadlock.

1. Mutual Exclusion:
Two or more resource are non-shareable (only one process can use at a time).

2. Hold & Wait:
A Process is holding at least one resource and waiting for resource.

3. No Preemption:
Resource can be preemptive until process released them.

DATE:

```
init_resources()
{
 pmax[0][0]=3; pcurr[0][0]=1; avl[0]=3;
 pmax[0][1]=3; pcur[0][1]=2; avl[1]=1;
 pmax[0][2]=2; pcur[0][2]=2; avl[2]=1;
 pmax[1][0]=1; pcurr[1][0]=1;
 pmax[1][1]=2; pcurr[1][1]=0;
 pmax[1][2]=3; pcurr[1][2]=3;
 pmax[2][0]=1; pcurr[2][0]=1;
 pmax[2][1]=1; pcurr[2][1]=1;
 pmax[2][2]=5; pcurr[2][2]=1;
}
 requestresource(){
// check if it is allocated, will it go to deadlock
 int proc, res[3];
printf("\nEnter which process, you need
      resource? (1-3)!\n");
 scanf("%d", &proc);
 proc--;
 if(running[proc])
{
printf("\n currently this process)
 for(i=j<3; i++)
 printf("%d\t", pmax[proc][i]-pcurr[proc][i]
for(i=0; i<3; i++)
 {
  loop=3;
```

DATE:

```
break;
}
else
{ print ("\n Deadlock prevention using Banker
          Algorithm \n");
  view resources();
  printf("\n1 Request resource(s) for a process\n");
  printf("\n2 View Allocated Resource\n");
  printf("\n3 Exit \n");
  printf("\n Enter your choice \n");
   scanf("%d", &ch);
    if (ch==1)
  {
    requestresource();
    getch();
  }
  else
    if (ch==2){
     view resources();
     getch();
  } else
      if ((ch==3)
        break;
        else
      printf("\n Invalid choice, please try again");
  }
  getch();
}
```

DATE:

Program : Banker_3_process[i]-c

```
//global variable
 int pcurr[3][3];
 int pmax[3][3];
 int avl[] = {6,4,7};
 int avltemp[] = {6,4,7};
int running[3];
int i,j,safe=0; count=0;
main()
{
 int ch;
 for(i=0; i<3; i++)
    running[i] = 1;
  int resourcer();
  while(1)
{
 clrscr();
  count=0;
 for(i=0; i<3; i++)
 {
   if(running[i])
     count++;
 }
   if(count==0)
{
print("\n\n\n\n\n congratulation ! we have
       completed execution of all processes
       successfully without any deadlock!");
getch();
```

DATE:

```
viewresources ()
{
printf ("\n .... Current snapshot of the
        system ....\n");
printf("\n max resource in the system: \n");
printf(" R1\t R2\t R3\n");
  for ( i=0; i<3; i++)
  printf (" %d \t", Maxres[i]
  printf (" \n Current resource available in
        the system: \n");
  printf ("R1 \t R2\tR3 \n");
   for (i=0; i<3; i++)
```

Output:

Deadlock prevention using Banker's Algorithm:
Max. resource in the system : 10
Current resource available in the system : 4

Max. Resources required for Completion Process

P1	7
P2	4
P3	6

Currently allocated resource for process

P1	2
P2	1
P3	3

DATE:

```
pcurr [proc] [i] += res[i];
 avl [i] -= res [i];
}
if (! checksafe())
{
 for (i=0; i<3; i++)
 {
  pcurr [proc [i] == res [i];
    avl [i] -= res[i];
 }
  return 0;
}
  return 1;
}
int checkcompletion (int proc)
{
 if ((pcurr [proc] [0] == pmax [proc] [i] ==
   pmax
     [proc] [i]) && (pcurr [proc] [2] == pmax
           [proc] [2]))
{
 for (i=0; i<8; i++)
 {
  avl[i] += lmax [proc] [i];
 }
  running [proc] = 0;
    return 1;
  }
    return 0;
}
```

DATE:

```
printf("\n Enter no of Resource %d to allocate
to process %d : \n", i+1, proc+1);
 scanf("%d", &res[i];
if ((res[i] > (pmax[proc][i] - pcurr[proc][i]))||
          (res[i] > avl[i]))
{
 printf("\n Invaild value!, try again!");
  goto loop3;
 }
}
getch();
if (allocate (proc, res))
 {
  printf("\n Resource successfully allocated\n"
  if (checkcompletion (proc))
  printf ("\n process %d has completed its
     execution and its resource have been
     released \n" proc+1);
}
else
printf("\n Resource cannot be allocated as it
      may lead to Deadlock\n");
} else {
 printf("\n Invaild process no. \n");
 getch();
}}
 int allocate (int proc, int res[3])
{
 for(i=0; i<3; i++)
```

DATE:

Assignment No-07.

Aim:
Implementation page replacement algorithm
1. FIFO 2. LRU.

Theory:
1. What is paging / Page Replacement?
Page is storage Mechanism. Paging is used to retrive processes from Secondary memory to primary memory.
The main memory is divided into small blocks called pages.
Page Replacement:
- When memory located in secondary memory is needed, it can be retrived back to main memory.
- Process of storing data from main memory to secondary memory swapping out.
- Retrieving data back to main memory.

2. Page Replacement Algorithm FIFO.
Page Replacement Algorithm is the simplest page replacement algorithm. In this algorithm the operating system keep track of all pages in the memory in a queue, the oldest page is in the front of the queue. When a page needs to be replaced page in the front of queue is selected for removal.

Sanjivani Rural Education Society's

SANJIVANI COLLEGE OF ENGINEERING,

KOPARGAON - 423 603

(An Autonomous Institute)

Department ECE Engineering

PRACTICAL / TERM-WORK REPORT

Subject SPOS

Name of the Student Wakte Sakshi Vijay.

Class TY Division -

Batch No. I3 Roll No. 73

Name of the Experiment / Title of Term Work Implementation page Replacement algorithm 1) FIFO 2] LRU

Experiment / Term - Work No. 07.

Conducted on : 08/04/24

Expected date of Submission : 12/4/24

Actual date of Submission : 03/05/24

Checked by Prof. N. Y. Siddiqui

(Name of the Teacher)

Remarks 08/10

Signature 03/5/24 Date

DATE:

For which process, you need resource? (1-3): 3
Currently p3 need 3 resource :
Enter no. of Resource to Allocate to process 3: 3
Resource successfully allocated.

Deadlock prevention using Banker's Algorithm:
Max resources in the system : 10
Current resource available in the system: 1

Max Resource required for Completion
Process

P1 7
P2 4
P3 6

Currently allocated resource for process
P1 2
P2 1
P3 6

Process 3 has completed its exeution and its resource have been released.

Congrulation! Exeution of all processes completed succesfully without any deadlock!

Conclusion : In this practical, we have implement Bankers Algorithm for Deadlock detection and avoidance, Here the program tells us whether it allocated resource gives the safe state or not.

DATE:

```
for (i=0; i<frsize; i++)
    if (fr[i] = FP[k])
        fs[i] = 1;

for (i=0 ; i<3; i++)
  if (fs[i] == 0)
      index = i;
  fr[index] = p[j];
}
else
{  // FIFO Algorithm Login
  fr[index] = p[i];
    index ++;
    if (index == frsize)
     index = 0;
}
 PF ++;
}
 Display();
}
printf ("In no of page faults :- d", PF);
if (ch == 0)
{
  ch = 1;
  cy = 1; PF = 0;
printf("In with LRU Algorithm");
 goto again;
}
getch ();
}
```

DATE:

```
clrscr();
printf("\n Enter Frame size (1-10): ");
scanf("%d", &frsize);
printf("\n Enter string length (1-20): ");
scanf("%d", &stsize);
printf("\n Enter %d element of string":
    stsize);
for(i=0;i<stsize;i++);
scanf("%d", &p[i];
printf("\n WITH FIFO algorithm");
again for(i=0; i<frsize;i++)
    fr[i]=-1;
for(j=0;j<stsize;j++)
{
  flag1=0; flag2=0;
    for(i=0;i<ftsize;i++)
    if( fr[i] == p[j])
    {
      flag1=1;
      flag2=1;
      break;
    }
  if(flag2==0)
  {
    if(ch==1)
    { //LRU Alg. Logic
      for(i=0;i<frsize;i++)
fs[i]=0;
for(k=j-1; i=1; i<frsize-1; i++, k--)
```

3. Page Replacement Algorithm LRU.

The LRU stands for the least Recently Used. It keeps track of page usuage in the memory over a short period of time. It works on the concept that pages that pages that have been highly used in the past are likely to be significantly used again in the future. It removes the page that has not been utilized in the memory for the longest time. LRU is the most widely used ~~to~~ algorithm because it provides fewer page faults than the other method

Program:

```
int fr[10], cy=1, frsize, stsize, p[20];
   //frame void display()
{
   int i;
   printf("\n cycle %d \t Requested page
        %d \t", cy, p[(cy-1]]);
   for(i=0; i<frsize; i++)
   printf("\t %d", fr[i]);
   cy++;
}
void main()
{
   int i, v, ch=0, fs[10];
   int index=0; k, l, flag=0, flag2 =0; PF=0;
```

Sanjivani Rural Education Society's

SANJIVANI COLLEGE OF ENGINEERING,

KOPARGAON - 423 603

(An Autonomous Institute)

Department ECE Engineering

PRACTICAL / TERM-WORK REPORT

Subject SPOS

Name of the Student Wakte Sakshi

Class TY Division

Batch No. T3 Roll No. 73

Name of the Experiment / Title of Term Work Write an Interactive shell program on UNIX / LINUX

Experiment / Term - Work No. 08

Conducted on : 15/04/24

Expected date of Submission : 22/04/24

Actual date of Submission : 03/5/24

Checked by Prof. N. Y. Siddiqui
(Name of the Teacher)

Remarks 09/10

Signature Date

DATE:

Cycle 6:	Requested page:	2	4	1	2
Cycle 7:	Requested page:	5	5	1	2
Cycle 8:	Requested page:	1	5	1	2
Cycle 9:	Requested page:	2	5	1	2
Cycle 10:	Requested page:	3	3	1	2
Cycle 11:	Requested page:	4	3	4	2
Cycle 12:	Requested page:	5	3	4	5

No of page faults: 9

Conclusion:

In conclusion, In this pratical we have implemented page replacement algorithm 1 FIFO 2. LRU.

DATE:

DATE:

Output:

```
Framesize (1-10) : 3
String length (1-20) : 12
Enter 12 element of string : 1 2 3 4 1 2 5 1 2 3 4 5.
With FIFO algorithm
Cycle : 1 Requested page : 1    1  -1  -1
Cycle : 2 Requested page : 2    1   2  -1
Cycle : 3 Requested page : 3    1   2   3
Cycle : 4 Requested page : 4    4   2   3
Cycle : 5 Requested page : 1    4   1   3
Cycle : 6 Requested page : 2    4   1   2
Cycle : 7 Requested page : 5    4   5   2
Cycle : 8 Requested page : 1    1   5   2
Cycle : 9 Requested page : 2    1   5   2
Cycle : 10 Requested page : 3   1   3   2
Cycle : 11 Requested page : 4   4   3   2
Cycle : 12 Requested page : 5   4   5   2

no of page faults : 8

With LRU algorithm

Cycle : 1 Requested page : 1    1  -1  -1
Cycle : 2 Requested page : 2    1   2  -1
Cycle : 3 Requested page : 3    1   2   3
Cycle : 4 Requested page : 4    4   2   3
Cycle : 5 Requested page : 1    4   1   3
```

DATE:

- The Bash shell is away for the user to execute command to the system.
- shell function are a way to group command for later execution using a single name for the group.
- Bash shell (Bourine Again Shell (bash)

```
#program
#!/bin/sh -i

clear
echo 'program to perform Arithmetic
      operation on 2 Integer Number'

ch=1
while [$ch -eq 1]; do

echo -n "Enter First Integer number:"
read a

echo -n 'Enter second Integer number:'
read b
echo -n '1.. ADDITION 2= SUBSTRACTION
      3= MULTIPLICATION 4= DIVISION
      " Enter choice:"

read ch
```

2. CSHell.

2. Shell scripting command:
Shell is special user program that provides an interface to user to use os service. shell accepts human readable command from the user and converts them into something which the kernel can understood
It is a command language interpreter that execute command read. from input device such as keyboard or from files

Displaying the file content on the terminal:
- cat: It is generally used to concatenate the files. It gives the output on the standard output.
- more: It is filter for paging through text one screenful at a time.
- less: It is used to viewing the file instead of opening the file.

3. Bash shell in breif.
- Bash is a Unix shell and command language
- Bash is Unix shell and command language written by Brian Fox for the GNU project as a free software Replacement for the Bourne shell.

The Bash shel
execute comm
shell function

DATE:

Assignment NO-08.

Aim:-
Write an Interactive shell program on UNIX /LINUX.

Theory:
1. Unix shell in brief.
A shell provide you with an interface to the unix system. It gather input from you and execute program based on that input when a program finishes executing it display that program input.
Shell is an environment in which we can run our command program and shell script
There are different flavor of Operating System.
Each flavor of shell has its own set of recognized command and function
1. shell prompt
The prompt $, which is called the command prompt is issued by shell, while the prompt is displayed you can type a command
Syntax:
$ date
Thu Jun 25 08:30:19 MST 2009

Shell Types:
a. Bourne Shell.

Sanjivani Rural Education Society's

SANJIVANI COLLEGE OF ENGINEERING,

KOPARGAON - 423 603

(An Autonomous Institute)

Department FCE Engineering

PRACTICAL / TERM-WORK REPORT

Subject SPOS.

Name of the Student Wakte Sakshi Vijay

Class TYECE Division -

Batch No. T3 Roll No. 73

Name of the Experiment / Title of Term Work Implement Job scheduling Algorithm 1. FIFO 2. shortest path First 3. RR.

Experiment / Term - Work No. 09

Conducted on : 29/04/24

Expected date of Submission : 1.1

Actual date of Submission : 03/5/24

Checked by Prof. N.Y. Siddiqui.
(Name of the Teacher)

Remarks 09/10

Signature Date

DATE:

1= ADDITION 2=SUBTRACTION 3= MULTIPLICATION
4 = DIVISION
Enter choice: 3

RESULT of 8 X 4 = 32

would you like to continue? Yes =1 / No=0 : 0

Conclusion:
Here we conclude that, we have interacted with the shell on Unix | Linux.

```
case $ch in
1) echo " RESULT of $a + $b = $ (($a + $b))" ;;
2) echo "RESULT of $a - $b = $ (( $a - $b ));
3) echo "RESULT of $a x $b = $C( $a * $b ))";
4) echo "RESULT of $a / $b = $ (( $a / $b ))" ;;

echo "you have not entered proper choice"
esac
echo "would u like to continue YES = 1/NO = 0"
read ch
done
exit 0.
```

Outputs:

Program to perform Arithmetic operation on 2 integer numbers

Enter First Integer number: 10
Enter second Integer number: 5

1 = ADDITION, 2 = SUBTRACTION, 3 = MULTIPLICATION, 4 = DIVISION
Enter choice : 1
RESULT of 10 + 5 = 15
would you like to continue? Yes = 1 / NO = 0 : 1

Enter First Integer number: 8
Enter Second Integer number: 4

DATE:

Disadvantage:

1. It has poor response time for process.
2. Machine can freeze up due to bugs.

4 FCFS / FIFO scheduling policy.

First Come First Serve CPU scheduling Algorithm shortly known as FCFS is the First algorithm of CPU process scheduling Algorithm. In First Come First Serve algorithm what we do is to allow the process to execute in linear manner. This means that whichever process enters process enter the ready queue first is executed first. This shows that First come First serve Algorithm follows first In first FIFO principle

5 Shortest Job first / SNJ scheduling policy.

Shortest job first is an optimal scheduling algorithm as it gives maximum throughput and minimum average waiting time and turn around time but it is not practically implementable because the burst time of a process can't be predicted in advance. We may not know the length of the next CPU burst but we may be able to predict its value We expect the next CPU burst will be similar in length to the previous one. By computing an approximate of the length of the next CPU burst, we can pick the process with the shortest predicted CPU brust.

3. Non-Preemptive and Preemptive Scheduling Scheme

Preemptive Scheduling:

Preemptive Scheduling is a method that be used when process switches from running state to ready state & waiting state to ready state. The resources are assign to a process for particular time & then remove. If the resources still have remaining bust time the process is place back in ready queue. The process remain in ready queue until it given a chance to execute again

Advantage:

1. It improve the average response time.
2. Utilize this method in multiprogramming environment is more advantagous.

Disadvantage:

1. It required use of limited computation resource

Non-Preemptive:

It is a method that may be used when process terminate or switches from running to waiting state.

When a processor are assign process they keep the process until it eliminated or reach the waiting state.

It may not interrupt in middle.

Advantage:

1. It provide low scheduling overhead
2. It is very simple method.

DATE:

Assignment NO-05.

Aim:
Implement Job scheduling Algorithm
1. FIFO
2. Shortest Path First
3. Round Robin.

Theory:

1. What is Job Scheduling ?

→ Job Scheduling is the process where different tasks get executed at pre-determined time or when the right event happened. A job schedular is a system that can be integrated with other software system for the purpose of executing or notifying other software component when a pre-defined, scheduled time arrives.

2. scheduling policies:

A Scheduling policy is a set of rules that determine the process to run
eg - Round Robine,
2. Shortest Job first
3. Priority Scheduling
4. FCFS.

DATE:

```
printf("%d|", j);
for(k=0; k<n; k++)
  if(temp1[k]!=0)
  {
    proc[k].turn++;
    proc[k].wait++;
  }
  temp1[i]--;
  proc[i].wait--;
 }
}
printf("\n process \t Bust time \t priority
 \t Arrival time \t Turnaround \t waiting");
for(j=0; j<=n; j++)
for(i=0; i<n; i++)
 if(proc[i].q==j)
{
 printf("\n p %d \t %8d %18d %15d %
  17d %13d", j, proc[i].b, proc[i].p,
  proc[i].a, proc[i].turn, proc[i].wait;
  ta += proc[i].turn;
  wat = pro[i].wait;
 proc[i].turn=0;
 proc[i].wait=0;
}
printf("\n Average turn around time
  is %f", ta/n);
printf("\n Average waiting time is %f",
       wa/n);
```

```
    for (j=0; j<n-1; j++)
{
    if (proc[j].b>proc[j+1].b)
    interchange (j);
    if(proc[j].b==proc[j+1].b)
      if (proc[j].p<proc[j+1].p)
        interchange (j)
}

printf("\n Result with SJF scheduling \n");
 output();
}
 void rr()
{
    int temp1[40], total=0;
    float ta=0, wa=0;
printf ("Result with round robin scheduling
  \n Quantum time=unity \n arrival
   time ignored \n sequence is");
for (i=0; i<n; i++)
{
 temp1[i] =proc[i].b;
 total += proc[i].b;
}
for (j=0; j<total; j++)
    {
     for (i=0; i<n; j++)
       if (temp1[i] != 0)
    {
```

DATE:

```
{ For(i=0;i<n-1;i++)
   for(j=0;j<n-1;j++)
{
if(proc[j].p>proc(j+1).p)
interchange(j);
if(proc[j].p==proc(i+1).p)
      if (proc[j].b>proc[j+1).b)
     interchange(j);
}
printf("\n Result with priority scheduling
  output();
}
 void fcfs()
 {
  for(i=0;i<n;j++)
  for(j=0;j<n;j++)
  {
   if(proc[j].a>proc[j+1].a)
     interchange(j);
   if(proc[j].a==proc[j+1].a)
   if(proc[j].p<proc(j+1).p)
     interchange(j);
}
printf("\n Result with FCFS scheduling \n");
 output();
}
 void sjf()
 {
    for(i=0;i<n;i++)
```

```
void output()
{
 int sum;
 float ta=0, wa=0;
  sum proc[0].a;
  for(i=0; i<n; i++)
  {
   proc[i].wait = sum - proc[i].a;
    sum += proc[i].b;
   proc[i].turn = sum - proc[i].a;
   tat = proc[i].turn;
    wat = proc[i] wait;
  }
   ta = ta/n;
   wa = wa/n;
  printf("\n Process \t Bust time \t priority \t
     arrival time \t Turnaround \t waiting");
 for(j=0; j<=n; j++)
   for(i=0; i<n; i++)
   if(proc[i].q == j)
   printf("\n p%d \t %d \t %d \t %d %d %d",
      j, proc[i].b, proc[i].p,
   proc[i].a, proc[i].turn, proc[i].wait);
  printf("\n Average turn around time
       is %f", ta);
  printf("\n Average waiting time is
       %f", wa);

   getch();
 } void ps()
```

} for(i=
for(j

DATE:

Program:

```
struct
{
int p,b,q,a,turn,wait;
} proc[40], temp;
int n,i,j,k;
void interchange(int i)
{
  temp = proc[i];
  proc[i] = proc[i+1];
  proc[i+1] = temp;
}
void input()
{
  clrscr();
printf("JOB SCHEDULING ALGORITHMS\n");
printf("\n enter no of processes");
scanf("%d", &n);
for(i=0; i<n; i++)
{
printf("\n enter following parameter
        for p%d", i+1);
printf("\n Burst time=");
scanf("%d", &proc[i].b);
printf("\n priority =");
scanf("%d", &proc[i].p);
printf("\n Arrival time=");
  proc[i].q = i+1;
  }
}
```

Program: struct

6. Round Robin scheduling policy::

Implementation

Processes are dispatched FIFO. But are given a fixed time on CPU (quantum time slice)

Characteristic

- Preemptive
- Effective in time sharing enviranment
- penalises I/O bound processes.

Advantage:

1. There is fairness since every process gets an equal share of CPU
2. The newly created process is added to the end of the ~~redy~~ ready queue.

Disadvantage:

1. There is low throughput
2. There is context switches
3. There is larger waiting time and Respone time.

3. Key terms associated with Disk Scheduling

- Seek Time:- Seek time is the time taken to locate the disk arm to a specified track where the data is to be read or written.
- Rotational latency:- Rotational Latency is the time taken by the desired sector of the disk to rotate into a position so that it can access the Read/write heads.
- Transfer Time - Transfer time is the time to transfer the data. It depend on the rotating speed of the disk and the number of bytes to be transfered.
- Disk Access Time.

DAT = Seek Time + Rotational latency + Transfer Time

Total seek Time = Total Head x seek movement Time

- Response Time - Response Time is the average time spent by request waiting to perform its I/O Operation

Title: Disk scheduling Algorithm for FCFS.

Aim: Study the disk scheduling algorithm and its types, for FCFS.

Requirement

Online GDB, Pc.

Theory:

1. What is Disk Scheduling Algorithm?
A process makes the I/O request to operation system to access the disk. Disk scheduling Algorithm manages those requests and decide the order of the disk access given to the request.

2. Why Disk Scheduling Algorithm is needed?
Disk Scheduling Algorithm are needed because a process can make multiple I/O request and multiple process run at same time. The request made by a process may be located at different sector on different tracks. Due to this, the seek time may increase more. These algorithm help in minimizing the seek time by ordering the request made by the processes.

Sanjivani Rural Education Society's

SANJIVANI COLLEGE OF ENGINEERING,

KOPARGAON - 423 603

(An Autonomous Institute)

Department ECE Engineering

PRACTICAL / TERM-WORK REPORT

Subject SPOS

Name of the Student Wakte Sakshi V

Class TY Division

Batch No. T3 Roll No. 73

Name of the Experiment / Title of Term Work Disk Scheduling Algorithm for FCFS.

Experiment / Term - Work No. 10

Conducted on :

Expected date of Submission :

Actual date of Submission :

Checked by Prof. N.Y. Siddiqui.
(Name of the Teacher)

Remarks 08/10

Signature 03/5/24 Date

Page No.
Date

Average turn around time is 9.333333
Average waiting time is 4.333333

Result with ROUND ROBIN Scheduling

Quantum time = unity
arrival time ignored
Sequence is

|0|1|2|0|1|2|0|2|2|2|2|2|2|2|2|2|2|2|2|

Process	Brusttime	Priority	Arrivaltime	Turn around	Waiting
P0	5	2	0	18	7
P1	3	1	1	12	9
P2	7	3	2	22	15

Average turn around time is 15.666667
Average waiting time is 10.333333

Conclusion:

In this practical, we learn and understood how to implement Job Scheduling algorithm like FIFO, Shortest Path first and Round Robin.

Page No.
Date

JOB SCHEDULING ALGORITHMS

Result with Priority Scheduling

Process	Bursttime	Priority	Arrival time	Turn around	Waiting
P1	5	2	0	8	3
P2	3	1	1	7	3
P3	7	3	2	15	8

Average turn around time is 10.000000
Average waiting time is 4.666667.

Result with FCFS Scheduling

Process	Busttime	Priority	Arrival time	Turn around	waiting
P1	5	2	0	5	0
P2	3	1	1	8	5
P3	7	3	2	15	8

Average turn around time is 9.333333
Average waiting time is 4.333333

Result with SJF scheduling

process	Brusttime	Priority	Arrivaltime	Turnaround	waiting
P2	3	1	1	5	2
P1	5	2	0	8	3
P3	7	3	2	15	8

DATE:

```
    getch();
}
void main()
{
    input();
    ps();
    fcfs();
    sjf();
    rr();
}
```

Output:

```
Enter No. of processes: 3

Enter following parameter for P1:
Burst time: 5
Priority: 2
Arrival time: 0

Enter following parameter for P2:
Burst time: 3
Priority: 1
Arrival time: 1

Enter following parameter for P3:
Burst time: 7
Priority: 3
Arrival time: 2
```

Page No.
Date

```
int arr[8] = { 176, 79, 34, 60, 92, 11, 41, 114 };
 int head = 50;

 FCFS (arr, head);

  return 0;
}
```

Output:

Total no of seek operation = 510

Seek sequence is

176
79
34
60
92
11
41
114

Program:

```
#include <stdio.h>
#include <math.h>

int size=8;
void FCFS (int arr[], int head)
{
 int seek_count =0;
 int curr_track, distance;
 for (int i=0; i<size; i++)
  {
   curr_track = arr [i];

 distance = fabs (head_curr_track);
 seek_count += distance;
  head = cur_track;
 }
 printf ("Total no of seek operation:
          %d \n", seek_count);
 printf("seek sequence is \n");
 for(int i=0; i<size; i++)
 {
  printf ("%d \n", arr[i]);
 }
}
int main ()
{
```

4. Dist scheduling Algorithm
FCFS
SSTF Shortest Seek Time First
SCAN Elevator Algorithm
C-SCAN Circular Scan
LOOK
C-LOOK
RSS
LIFO
N Shep SCAN
F-SCAN

FCFS (First come First Serve)

FCFS is the simplest of all Disk scheduling In FCFS the request are addressed in the order they arrive in the disk queue

Advantage of FCFS

(1) Every Request get a fair chance
(2) No indefinite postponement

Disadvantage of FCFS

(1) Does not try to optimize seek time
(2) May not provide the best possible serve