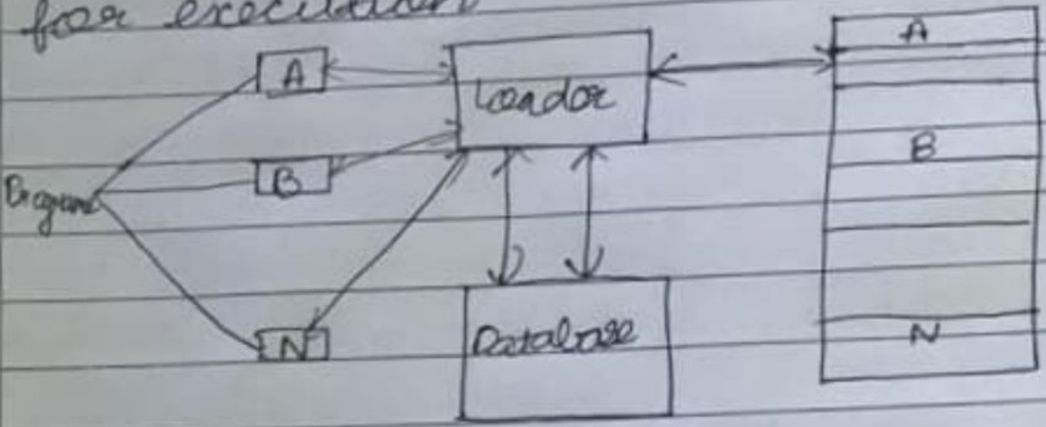
Enlist the different types of errors that are handled by Pass I and Pass II of assembler
Error messages generated during an assembly may originate from the assembler or from a higher level language such as C or from operating system.
Errors indicate that the assembler is unable to interpret or implement the intent of a source line.
Errors and warnings in Pass I assembler:
duplicate label
RESW or RESB has invalid operand location counter exceeds 1,048,575 unrecognized entry in op code field
warning: unknown qualifier for operand
warning: missing or invalid START statement
warning: missing END statement
warning: symbol too long - truncated to SYMBSIZE-1
Errors and warnings in Pass 2 assembler
out of range for PC-relative addressing and BASE is not in effect
out of range for PC-relative and BASE-relative addressing
operand not found
invalid register id
indexed, immediate and indirect addressing are not allowe

Define loader. What are different functions of loade
A loader is a system program, which takes the object code of a program as input and prepares it

Scanned with CamScanner

for execution



The loader performs the following functions:
Allocation:
It is used to allocate space in memory for the object programs. Translators cannot allocate space since overlap may occur or large wastage of memory takes place.
Linking:
It combines two or more seperate object programs and resolve symbolic references between object decks. It supplies information needed to allow reference between them.
Relocation: It modifies the object program so that it can be loaded at an address different from the location originaly specified and adjusts all address dependent locations.
Loading: Physically it places the machine instructions and data into the memory for the execution.

Scanned with CamScanner

	Top up Parser	Bottom up parser
	1) it is a parsing strategy that first looks at the highest level of the parse tree and works down the parse tree by using the rules of grammar	1) It is a parsing strategy that first looks at the lowest level of the parse tree and works up the parse tree by using the rules of grammar
2)	Top down parsing attempts to find the left most derivations for an input string	2) Bottom up parsing can be defined as an attempts to reduce the reduce the input string to start symbol of a grammar
	In this parsing technique we start parsing from top to down in top-down manner	3) In this parsing technique we start parsing from bottom to up in bottom up manner
	This parsing technique uses left most Derivation	4) This parsing technique uses right most Derivation.
	It's main decision is to select what production ule to use in order to onstruct the string.	5) It's main decision is to select when to use a production rule to reduce the string to get the starting symbol

hat is the need of Intermediate code generation? Explain

[illegible]
language.

$$\begin{tikzcd}
\boxed{\text{Parser}} \arrow[r] & \boxed{\text{Static checker}} \arrow[r] & \boxed{\text{Intermediate code generator}} \arrow[r, "\text{Intermediate code}"] & \boxed{\text{Code generator}}
\end{tikzcd}$$

Position of intermediate code generator

Using the intermediate code, the second phase of the compiler, synthesis phase is changed according to target machine.

The two intermediate code generation forms are as follows

Postfix Notation: No parentheses are needed in postfix notation because the position and arity of the operators permit only one way to decode a postfix expression.
The postfix representation of the expression $(a-b)*(c+a)+(a-b)$ is $ab-ca+*ab-+$.

Three Address Code:

A statement involving no more than three references that is two for operands and one for result is known as three address code.

The three address code for expression $a+b*c+d$:

$T1 = b*c$ $\quad$ $T2 = a+T1$

$T3 = T2+d$

T1, T2, T3 are temporary variables.

Q.2.a) What is left factoring? Find FIRST & FOLLOW for

S → Aa   A → BD

B → b|ε   D → d|ε

⇒ If RHS of more than one production starts with the same symbol, then such a grammar is called as grammar with common prefixes and ~~this~~ the process by which the grammar with common prefixes is transformed to make useful for Top-down Parser is known as Left Factoring.

Here, • we make one production for each common prefix

• The common prefix may be terminal or a non terminal or a combination of both

• Rest of the derivation is added by new productions.

The grammar obtained after this is called left factored gramm

S → Aa	First(S) = First(A) ie First(BD)
A → BD	First(B) = {b, ε}
B → b|ε	First(D) = {d, ε}
D → d|ε	∴ First(A) = {b, d}
	First(S) = {b, d, a}

Follow(S) = {$} because S is starting symbol.

Follow(A) = {a} because a is followed by A.

Follow(B) = First(D) = {d, a}

+ Follow(A)

Follow(D) = Follow(A) = {a} ∵

Ans 6] The phases of compiler are given as follows

1] Lexical Analyzer

2] Syntax Analyzer

3] Semantic Analyzer

4] Intermediate Code Generator

5] Code optimizer

6] Code generator

Statements : int a, b, c = 1;

a = a* b - 5*3/c

Source Program:

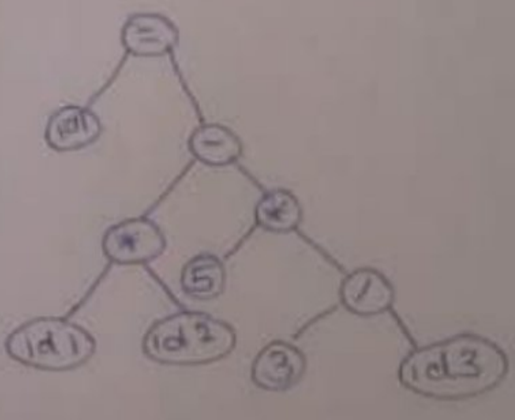
Lexical analysis: First the initialization of all variables will take place and the symbol table will be generated

	Attribute	Value
1 -	id	1 (a)
2	id	- (b)
3	id	- (c)

Now for a = a*b - 5*3/c

⇒ id1 = id1 * id2 - 5 * 3 / id3 ⇒ Stream of tokens

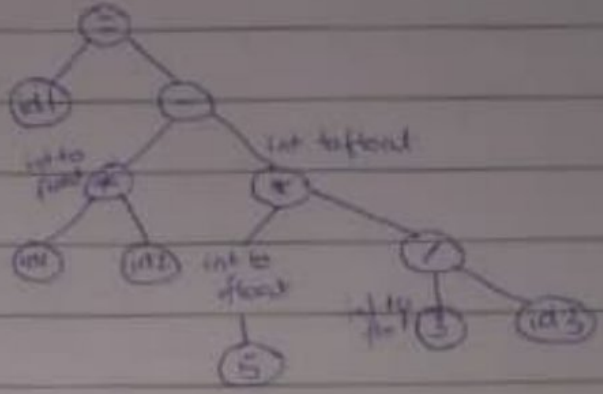
Syntax analysis - Here the parse tree would be generated.



P.T.O

Scanned with CamScanner

Semantic analyser: Here the input is parse tree given by Syntax analyser and output is semantically correct parse tree



Intermediate code generator:

Using 3-Address Code Format:

$t1 = 3 / id3$

$t2 = 5 * t1$

$t3 = id1 * id2$

$t4 = t_3 - t2$

$id1 = t_4$

Code optimizer:

$t1 = 3 / id3$; $t2 = 5 * t1$

$t3 = id1 * id2$; $id1 = t_3 - t_2$

Code generator

```
LD   R1, id3                         R1 -> id3
DIV  R2, 3                           R4 -> id1
MUL  R3, R2, 5                       R5 -> id2
LD   R4, id1                         R2 -> t1
   LD  R5, id2                       R3 -> t2
   MUL R6, R4, R5
   SUB R4, R6, R3
```

Scanned with CamScanner

Q3.

a. Explain YACC in detail

→ i) YACC stands for yet another compiler compiler.

ii) YACC provides a tool to produce a parser for a given grammar.

iii) YACC is a program designed to compile a LALR(1) gramma

iv) It is used to produce the source code of the syntatic analyzer of the language produced by LALR(1) grammar

v) The input of YACC is the rule or grammar and the output is a C program

6. Input file:
YACC input file is divided in 3 parts

```
/* definitions */
...
%%
/* rules */
...
%%
/* auxillary routines */
...
```

7. Input file:-
The definition part includes information about the tokens used in the syntax definition.

```
% token NUMBER
% token ID
```

8. YACC automatically assigns number for token, but it can be overridden by

```
% token Number 621.
```

9. YACC also recognizes single characters as tokens. Therefore assigned token number should no overlap ASCII codes.

10. The definition part can include C code external to the defintion of the parser and variables declarations, with %{ and %} in 1st column.

Scanned with CamScanner

It can also include the specification of the starting symbol in the grammar
% start non terminal
Input file :
The rules part contains grammar definition in a modified BNF form.
Actions is C code in { } can be embedded inside.
Input file : Auxiliary routine :
The auxillary routine part is only C code.
It includes functions definitions for every function needed in rules part.
It can also contain the main ( ) function definition if the parser is going to be run as a program.
The main () function must call funⁿ yyparse ( ).
input file: generally finishes with .y
output : A parser y.tab.c (yACC)
The o/p file "file.output" contains parsing tables.
The file "file.tab.h" contains declarations.
The parser called the yyparse ( )

Explain machine independent code optimization techniques.
code optimization in compiler design: ~~the~~ In the synth phase is a program transformation technique, which tries to improve the intermediate code by making it consume fewer resources so that faster-running machine code will result. compiler optimizing process should meet the following objectives :-
The optimization must be correct, it must not, in any way, change the meaning of program
optimization should increase the speed & performance
compilation time must be kept reasonable.
process should not delay overall compiling process.

Scanned with CamScanner

optimization process is of 2 types
machine dependent optimization
machine independent optimization
In machine independent optimization code optimizatio
phase attempts to improve the intermediate code to get
a better target code as o/p.
eg:-

```
do
{
item =10;
 value = value + item;
}
 while (value <100);
```

This code involves repeated assignment of identifier
item, which if we put this way:

```
item =10;
do
{
value = value + item;
}
while (value <100);
```

should not only save CPU cycles, but can be used on
any processor.
intermediate code generation process introduces many
efficiencies, extra copies of variables, etc.
eg:-
copy propagation
loop unrolling
function inlining.

Scanned with CamScanner

Q1)

a)

Basic for comparison	Compiler	Interpreter
1) Input	It takes an entire program at a time	It takes a single line of code or instruction at a time
2) Output	It generates intermediate object code	It does not produce any intermediate object code
3) Working mechanism	The compilation is done before execution.	Compilation and execution take place simultaneously
4) Speed	Comparatively faster	Slower
5) Memory	Memory requirement is more due to	It requires less memory as it does not create intermediate object code
6) Errors	Display all errors after compilation, all at the same time	Display error of each line one by one.
7) Error detection	Difficult	Easier comparatively

Scanned with CamScanner

b) The grammar after eliminating left recursion is

$$S \rightarrow (L) / a$$
$$L \rightarrow SL'$$
$$L' \rightarrow SL' / \epsilon$$

c) Dynamic linker loader a special part of an operating system that loads external shared libraries into a running process and then binds those shared libraries dynamically to the running process. This approach is also called dynamic linking or late linking. It retrieves the address of function and variables contained in the library, execute those functions or access those variables, and unload the library from memory.

(i) It ~~is~~ provides the ability to load the routines only when they are needed so lot of time and memory is saved if subroutine are large with lots of external references

(ii) It helps in not loading the entire library for execution

(iii) In dynamic linking loader is used to load the main program.

Steps to accomplish the actual loading and linking of a called procedure.

1) The symbolic ~~na~~ name of the routine in the program is used to make the load and call service request

Scanned with CamScanner

to the operating system

(ii) The operating System checks its internal tables to determine whether or not the routine is already loaded

(iii) Control is then being passed from the operating system to the routine being called. When the called subroutine completes its processing, the operating system then returns the control of the program that issued this request.

Implementation of Dynamic Linking loader.

1. Dynamic Link library, or DLL, is Microsoft's Implementatio of the Shared Library concept in the Microsoft windows and OS/2 operating Systems. These libraries usually have the file extension DLL, OCX (for libraries containing Active X controls) or DRV (for legacy System drivers}.

2. In Apple Darwin operating system, OS X and iOS operating Systems the dynamically loaded shared libraries can be identified either by the filename suffix, dylib or by their placement inside the bundle for a framework.

3. In Unix-like operating systems using XCOFF, Such a AIX, dynamically-loaded Shared libraries

Scanned with CamScanner

Advantage

1) No ~~one~~ over head is incurred unless the procedure to be called or referenced is actually used.

2) A further advantage is that the system can be dynamically configured.

Disadvantage

1) Considerably overhead and complexity incurred, due to the fact that we have postponed most of the binding process until execution time.

a source program in order to resolve forward references in the program. A forward refrence is defined as a type of instruction in the code segment, that is referencing the label of an instruction, but the assembler has not yet encountered the definition of that instruction.

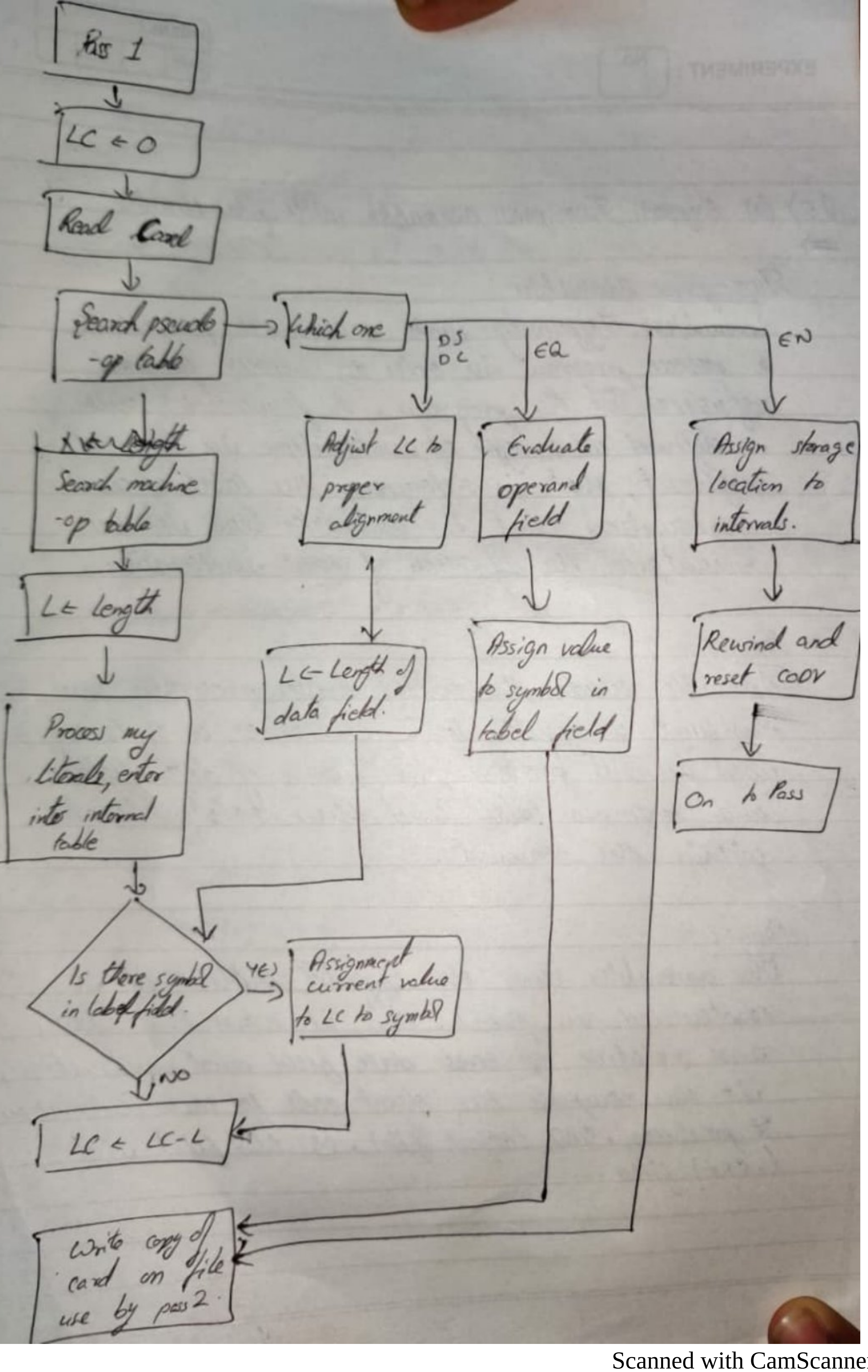
Pass 1:-

Assembler reads the entire source program and constructs a symbol table of names and labels used in the program, i.e name of data fields and programs labels and their relative location within the segment.

Pass-2:-

The assembler uses the symbol table that it constructed in pass 1. Now, it knows the length and relative of each data field and instruction it can complete the object code for each instructio It produces .OBJ (object file) . LST (list file) and (.CRF) files.

Scanned with CamScanner


Pass 1
LC ← 0
Read Card
Search pseudo-op table
Which one
DS DC
EQ
EN
Adjust LC to proper alignment
Evaluate operand field
Assign storage location to intervals.
~~L ← Length~~
Search machine-op table
L ← Length
LC ← Length of data field.
Assign value to symbol in label field
Rewind and reset coov
On to Pass
Process my literals, enter into internal table
Is there symbol in label field.
YES
Assignment current value to LC to symbol
NO
LC ← LC-L
Write copy of card on file use by pass 2.

Scanned with CamScanner

Q.5) (b) Explain with example, conditional macro expansion.

=>

The macro processor replaces each instruction with the corresponding group of source language statements. This is called macro expansion or expansion of macros.

Conditional assembly are frequently considered to be the mechanisms that allow a single version of the source code for a program to be used to generate multiple versions of the executable.

Most macro processor can modify the sequen of statements generated for a macro expansion, depending on the arguments supplied in the macro invocation. Conditional assembly is commonly used to describe this feature. It is also referred as a conditional macr expansion.

Conditional assembly can be achieved with the help of AIF & AGO statements.

Scanned with CamScanner

Example :-

(C-programming)

```
# define max(a,b) a>b ? a : b
main ()
{ int x, y;
  x=4; y=6;
  z= max (x,y);
}
```

This macro can be called like any other C-function, using the same syntax. Therefore, after pre-processing z = x>y ? x : y;

∴ After macro

Scanned with CamScanner

* Code Generator issues :-

→ Input data for the code generator :-

- Below mentioned are the following formats used as input for code generator.

i) Three address code (quadruples, triples, indirect triples)

ii) Virtual machine representation (byte code , stack machine)

iii) Linear presentation (postfix, infix, prefix)

iv) Graphical presentation (syntax trees, DAGs , parse tree)

2) Target program :-

→ i) Knowledge of machine architecture and instruction set are the pre-requisite for the design of good code generator

ii) Proper selection of machine architecture helps in producing absolute machine language program and re-locate machine language program.

3) Instruction Selection :-

The complexity of mapping IR program into code sequence depends on :-

i) Level of Intermediate Representation

ii) Type and nature of instruction set.

4) Register allocation and Assignment

→ Selection of set of variables that will reside in registers at each point and choosing specific register helps in faster execution of data

Scanned with CamScanner

5) Evaluation order

Selecting the order in which computations are performed effects the efficiency of the target code.

* Operator Precedence Parser :-

It is a bottom up parser that interprets an operator-precedence grammar. For example, most calculators use operator precedence parsers to convert from the human readable infix notation. The operator precedence parsing technique can be applied to operator grammars.

Operator grammar are defined as the grammars with the following properties :-

i) No epsilon in the right hand side of any production

ii) No adjacent non terminals in the right hand side of any production.

This property enables the implementation of efficient operator precedence parsers. These parsers rely on the following three precedence relations :-

Relation	Meaning
a < b	a yield precedence to b
a = b	a has the same precedence as b
a > b	a takes precedence over b

Scanned with CamScanner

These operator precedence relations allow delimiting the handles in the right sentential forms : <· marks the left end =· appears in the interior of the handle , and ·> marks the right end. Also , a <·b need not imply b·> a

Let us assume that between the symbols ai and ai + 1 there is exactly one precedence relation. Suppose that $ is the end of the string.

Then for all terminals we can write : $ <· b and b ·> $.
If we remove all nonterminals and place the correct precedence relation : <· , =· , ·> between the remaining terminals , there remains strings that can be analyzed by developed parser.

For example , the following operator precedence relations can be ~~b~~ introduced for simple expressions :

	id	+	*	$
id		·>	·>	·>
+	<·	·>	<·	·>
*	<·	·>	·>	·>
$	<·	<·	<·	·>