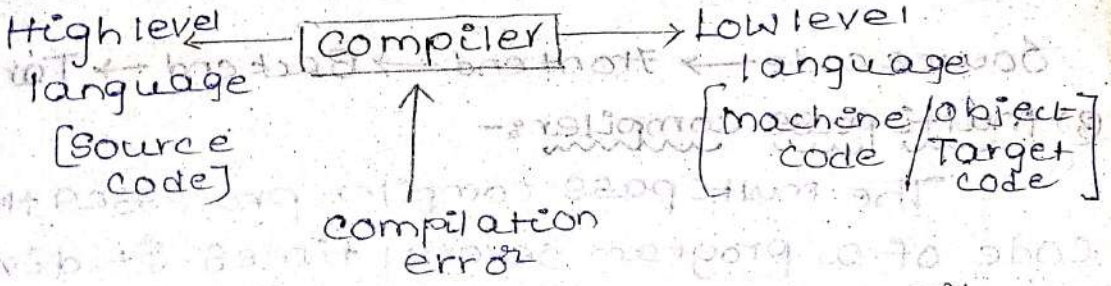
UNIT-3

COMPILERS:-

Basic compiler functions:-

Definition of compiler:-

* It is a system software that translates source code into machine code or assembly or object code also called as target code.
* It also makes the ENDcode efficient [out] which is optimised for execution & memory space.
* The compilation process includes basic translation mechanism & error detection
* compiler process goes through lexical, syntax and symantic analysis at front end and code generation and optimization at back end.



Features or characterstics of compiler:-

There are some important characterstics of a compiler that makes the compiler to compile program very fastly. They are

(1) correctness
(2) speed of compilation
(3) pressure the correct meaning of code
(4) The speed of target code

Types of compiler.

Depending upon the program compilation there are different types of compiler such as

(1) Single pass compiler
(2) Two pass compiler
(3) multi pass compiler

(1) single pass compiler:-

In single pass compiler the source code is directly transformed in to machine code

Ex:- PASCAL Language

(2) Two-pass compiler:-

TWO pass compiler is divided into two section such as

(a) Front end:-

It maps legal code in to intermediate representation [IR]

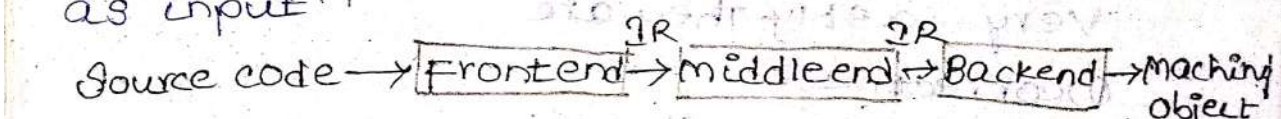
(b) Back end:-

It maps intermediate representation onto the target machine

Source code → Front end → Back end → Target code

(3) Multi pass compiler:-

The multipass compiler processes the soure code of a program several times. It divides the large program into multiple small programs and process them. It develops multiple intermediate codes. All of these multiple pass take the output of previous phase as input.



Various tasks of compiler:-

(1) It Breaks up the source program into pieces and impose gramatical structure on them.
(2) It allows us to construct the desired target programs from the intermediate representation & also creates a Symbol table.
(3) It compiles source code and detect errors in it.

(4) It manages storage of variables and codes & support for seperate compilation.
(5) It reads, analyse the entire program & translate to semantically equivalent.
6) It translates the source code into object code depending upon the type of machine

*uses & purpose of compiler:-

(1) It verifies entire program so that there are no syntax & symantic errors.
2) The executable file is optimised by the compiler so that it executes faster
(3) It allows us to create internal structure in the memory
4) There is no need to execute the program on the same machine it was built
5) It translates entire program into other language
6) It generates various files on disk such as .obj & .exe.
7) It links the file into an executable format.
8) It helps to enhance understanding of language semantics
9) It helps to handle language performance issues.

Types of special compilers:-

There are three types of special compilers, such as

(1) Native code compiler
(2) Cross compiler
(3) Source to source compiler.

(1) Native compiler:-

The compiler used to compile a source code for same type of platform only

ex:- Machine code → machine code

(2) Cross compiler:-

The compiler used to compile for different kinds of platform.

ex:- Machine A → machine B

(3) Source to source compiler:-

The compiler that takes high level language as input and outputs source code of another high level language

* Basic compiler functions:-

It describes fundamental operations that are necessary in compiling a typical high level programming language. For example PASCAL program, However the concept and approaches can also be applied to the compilation of programs in other languages. A high level programming language is described in terms of grammer. This grammar specifies the form or syntax, Legal statements in the language.

An example of a pascal program is as follows.

```
1. PROGRAM STARTS
2. VAR
3. SUM, SUMSQ, I, VALUE, MEAN, VARIABLE, INTEGER
4. BEGIN
5. SUM:=0
6. SUMSQ:=0
7. FOR:=1 to 100 DO
8. BEGIN
9. READ(VALUE);
10. SUM:=SUM+VALUE;
11. SUMSQ:=SUMSQ+VALUE *VALUE;
12. END;
13. MEAN:=SUM DIV 100;
14. VARIANCE:=SUM SQ. DIV100 - MEAN*MEAN;
```

15. WRITE(MEAN, VARIANCE);
16. END

An assignment statement (:=) might be defined by the grammar, a variable name followed by an assignment operator followed by an expression.

It is convinent to a source program statement as a sequence of programs rather than as a string of characters program tokens may be taught as a fundamental building blocks of the language. for example,

A token might be a keyword, variable name, integer etc.

The basic elements of a simple compilation process illustrating its application to the example program is as follows,

1) GRAMMAR
2) Lexical analysis

1. Grammar:-

A Grammar for a programming language is a formal description of the syntax of the programs and individual statements written in the language. The Grammar doesn't describe the semantics or meaning of various statements such as knowledge must be supplied.

The following example illustrates the difference between syntax and semantics

I := J + K

and

X := Y + I

Where X and y are real variables and I, J, k are integer variables. This two statements have identical syntax. In the given expression the two variable names are seperated by '+' operator the first statement specifies the variable that are added by using integer arthimetic operations and the second statement specifies floating point addition.

A Number of different notations are used for writing Grammar the one we describe is called as Back as Naurform [BNf]. BNf is not the most powerful syntax description tool available. Infact it is totally adequate for the description of Real programming languages. A BNf Grammar consists of set of rules which defines the syntax of some constructs in programming languages

for Ex:-

1. <prog> : : = program<prog-name><dec-list> BEGIN<stmt list>END
2. <prog-Name> : : = id
3. <dec-list> : : <declaration list> : <type>
4. <dec> : : = <id-list> : <type>
5. <type> : : = INTEGER
6. <id-list> : : id / <id-list> id
7. <stmt-list> : : = <stmt> / <stmt-list> ; <start>
8. <stmt> : : <assign> / <read> / <write> / <for>.
9. <assign> : : id : = <exp>
10. <exp> : : <Term> / <exp> / <terms> / <exp> - <term>
11. <Term> : : = <factor> / <Term> * <factor> / <Term> DIV <factor>
12. <factor> : : id / int (<exp>)
13. <read> : : = = READ (<id-list>)

14. <write> ::= WRITE (<id-list>)
15. <for> ::= for <Index-Exp> TO <body>
16. <Index-exp> ::= id := <exp> to <EXP>
17. <body> ::= <Start> / BEGIN <Start-list> END

In the line 13, it indicates the definition of syntax of PASCAL Read statement that is denoted in + Grammar as <read>. The symbol ":: =" can be read as "is defined to be" on the left of the symbol is the language construct being defined as <read> and on the right side it is the Description of the syntax being defined. characterstics of enclosed between "< >" Angular braces which are called non-terminal symbols.

In the line 6 <id-list> ::= id <id-list> id. The first alternative specifies "<id-list>", id may consist of simply a Token id. The second syntax alternative is <id-list> followed by the token, id.

2. Lexical Analysis:-

Lexical Analysis Involves scanning the program to be compiled and recognising the tokens that make up that source statement. scanner are usually designed to recognise keywords, operators, Identifiers Datatype, symbols, character string, and other similar items that are be written as a part of source program.

Items such as Identifiers and Integers usually recognised as single Tokens. These tokens could be defined as a part of the Grammar.

For Example:-

An identifier might be defined as follows:-

<iden>::<letter>/<Ident><letter>/<ident><digit>

<letter>::=A/B/C/D/----/z or /a/b/c/d/---/z/

<digit>::=0/1/2/3/----/9

The scanner would recognise tokens as single characters like A, B, a, b, 0, 1------ However this approach would require the parser [translator] to recognise simple identifiers using general parsing techniques. A special purpose routine such as scanner can perform the same function more efficiently. similarly the scanner recognise both single and multiple character directly.

Eg ex:-

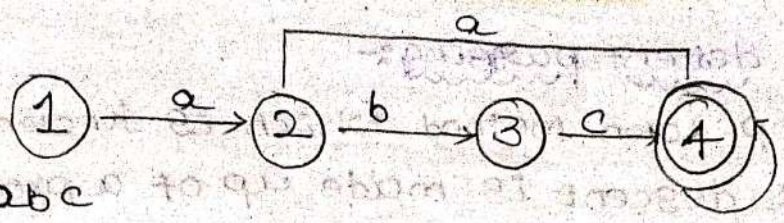
The character lead would be interpreted as a single token rather than as a sequence of tokens

READ → R/E/A/D

In addition to its primary function of recognising tokens are scanner is responsible for reading the lines of source program. comments are ignored by the scanner so they are effectively removed from the source statements before passing begins.

Modeling scanner as a finite automata:-

The Tokens of most programming languages can be recognised by finite Automation. mathematically, a finite automation consist of set of states and set of Transitions from one statement to another. one of the state is designated as final states. finite automata are represented graphically as follows in the below diagram.



(1) abc
(2) abcabc

In the above diagram states are represented by circles and transition by arrows from one state to Another. Each Arrow is labelled with a character.

The first line Recognise identifiers and keywords that Begin with a letter and continue with any sequence of letters and digits. The Notation A to z shows a finite automaton that recognises Identifiers. This automation does not allow the Identifiers begin with underscore or that contains two consecutive underscore. The second line shows, a finite Automation the recognises identifiers of these type

Syntatic Analysis:-

In syntatic Analysis, the source statements are recognised as language constructs described by the grammar passing technique are divided in to two general classes. They are:-

(1) Bottom-up Approach
(2) Top-down Approach

(1) Bottom-up Approach:-

This method begin with the terminal node of the tree and combines these into higher nodes until the root is reached.

(2) Top-down Approach:-

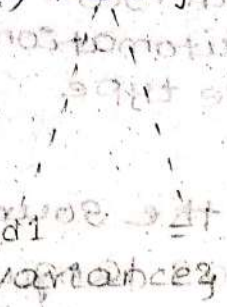
This method begin with the rule of the Grammar that specifies the goal of analysis.

Recursive descent parsing:-

A Top-down method which is known as recursive descent is made up of a procedure for each non terminal symbol in the grammar When the procedure is called, it attempts to find a substring of the input begening with the current token that can be interpreted as a non terminal. If the procedure is unable to find a substring that can be interpreted as a non terminal, it returns an indication of failure or invokes an error.

The following example illustrates a recursive descent parsing of assignment statement is as follows.

a) Assign



b) Assign

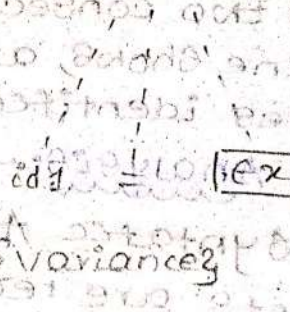


Diagram 'a' represents the procedure for non terminal symbols that are envolves in parsing this statement.

Diagram 'b' is a step by step representation of procedure calls at tokens.

Machine dependent compiler features:-

The purpose of a compiler is to translate programs written in a high level language into machine language most high level programming languages are designed to the relatively independent to the machine being used these means that the syntax of programs written in this languages should be relatively machine independent The real machine dependencies of a compiler are related to the generation optimisation of the object code.

Ex:-

code generation routines from the Grammar

<prog>::=<program name><declaration list>
BEGIN<stmt list>
END.....

<prog name>::=id
generate[START,0]
generate[EXTREF,XREAD,XWRITE]
generate[STL RETA-DDR]
add 3 to loc CTR
generate[RETADDR RESW1]

All code generation is machine dependent because we must know the instruction set of a computer to generate a code for it. However these are many complex issues that involves to improve the efficiency of execution. such type of code optimisation are considered by an intermediate form of the program being compiled. There are also many machine independent techniques for code optimisation that use a intermediate representation of a program.

There are 2 different types of machine dependent compiler features, they are:

(1) Intermediate form of program
(2) machine dependent code optimisation

(1) Intermediate form of program:-

There are many possible way of representing a program in intermediate form for code analysis and optimisation. It represents the executable instruction of the program with a sequence of operations. Each operation consist of set of operands such as op_1, op_2, result

The above operations can be performed by the object code OB1, OB2 are the two operands for this operation and the resulting value is performed.

Ex:-

Sum := sum+value;

It can be represented as quadraples like +, sum, value, i, :=, i1, sum.

It designates and intermediate result, the second quadraples assigns the value of intermediate result to the sum.

Similarly the statement

Variance := SUMSQ DIV100 - MEAN * MEAN

It could be represented with quadraples as div, sumsq, #100, $1, mean, mean, i2 -, i1, i3 :=, variance

for example, it can be rearranged to eliminate redundant load and store operation and the immediate result It can be assigned to registers & to the temporary variables to make their view as efficient as possible.

The following above example shows a quadraple corresponding to source program

2. Machine dependent code optimisation:-

It performs on many computers where there are a number of general purpose register that may be used to hold constants value of variables, intermediate results and so on. These are also be used for addressing.

machine instruction uses register as operands that are usually faster than corresponding instruction that refers to locations in memory. The register keeps all the variables and intermediate result that will be used later in the program. Each time a value is fetched from memory are calculated as an immediate result that can be assigned to some registers

This approach avoids unnecessary movement of values b/w memory and registers.

consider an example the quadraple the value used once in quadruple and twice in quadruple q. It is possible to fetch this values once again. Machine dependent code optimisation increases the execution speed and performance of computer and the only problem is wastage of memory is increased

* machine Independent compile features:-

It describes some common compiler features that are largely independent of particular machine being used. It describes the method for handling structured variables such as arrays. It also contains code optimization. It describes some alternative way of performing storage allocation for the compiled programs.

There are 3 different types of machine independent compiled features. They are:- (1) structured variable

(2) machine independent code optimisation

(3) storage Allocation

(1) Structured variable:-

The compilation of program that use structured variables such as Array, string, records and sets. We are primarily concerned with the allocaton of storage space for such variables and with the generation of code to reference them. The some principle can also be applied to other types of structured variables for example, consider the pascal array declaration as.

A: Array[1-----10] of Integer

In the above example, each integer variable occupies 1 word of memory. Then we must allocate 10 words to store this array.

Generally if an array is declared as

A: Array [l ---- u] of integers, then we must allocate (u-l)+1 words of storage for the array.

Allocation of multi-dimensional array is not more difficult.

for example, consider the 2 dimensional array as

B: Array [0 ---- 3, 1 ---- 6]

Here the first subscript can be taken out four different values (0 ---- 3) and second subscript can take six different values (1 ---- 6). Now we need to allocate a total of (4×6=24) words to store on array.

The following examples shows a possible ways of storing Array 'B'.

(a)

0,1	0,2	0,3	0,4	1,1	1,2	1,3	1,4	1,5	2,1	2,2	2,3	2,4	2,5	2,6	3,1	3,2	3,3	3,4	3,5	3,6

1st ROW 2nd ROW 3rd ROW 4th ROW

b)

0,1	1,1	2,1	3,1	0,2	1,2	2,2	3,2	0,3	1,3	2,3	3,3	0,4	1,4	2,4	3,4	0,5	1,5	2,5	0,6	1,6

a) In the above array 'a' the array elements have same value of the first subscript that are stored in location. This is called "Row major order"

b) In the above array 'b'. An array elements that have same value of the second subscript that are stored together is called as "column major order"

2. Machine Independent code optimization:-

It consist of the most important type of machine code optimisation. We assume that the source program has already been translated in to a sequence of quadraples. one important source of code optimisation is the elemination of sub-expression appear more than one point in the program and that compute the same value.

The term i,2*j is a sub expression. The optimising compiler generates a code generater. Thus multiplication is performed and the result is used in both the places.

for example x,y,Array[1---10,1----10] of enteger

for i=1 to 10 do

x[I, 2*j-1] := y[I, 2*J-1]

It can be written as T_1: 2*j;

T_2: T_1-1;

for i:=1 to 10 do

x[I, T_2] := y[T_1, I];

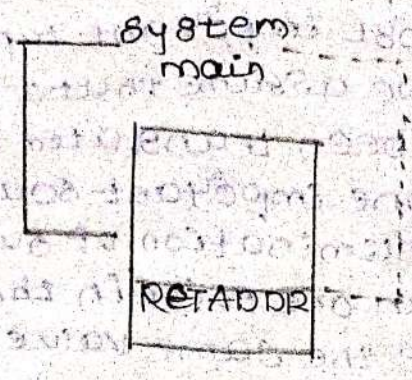
However this would achieve a part of the benefits realised by the optimisation process that has been just described.

3. storage allocation:-

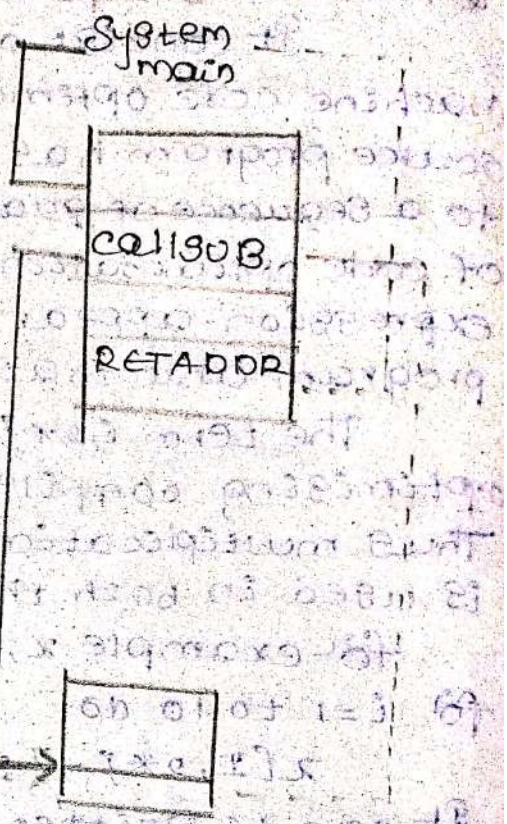
It includes temporary variables, save the return address and also assign the fixed addresses with in the program. The simple type of storage assignment is usually called static Allocation. It is often used for languages that do not allow the recursive use of procedures & sub-routines and do not provide the dynamic allocation of storage during execution. If procedures may be called recursively static allocation cannot be used.

for example:-

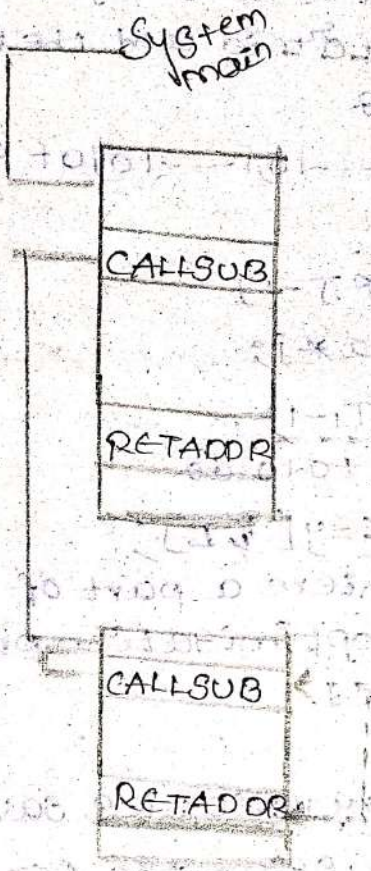
a)



b)



(c)



In the above diagrams, in the diagram 'A', the program main has been called by the operating system & by the loader, the first action taken by the main is to stored, its return address from the register.

In the diagram 'B', the main has called the procedure sub. The return address ~~to this~~ call has been stored at a fixed location.

In diagram 'c' the procedure sub stores the address into RETADDR from the Register. This destroys the return address in the procedu main as a result there is no possibility of making a return to the main.

* compiler design option:-

In this topic we can consider some of possible alternatives on the design and construction of (or) compiler. compiler designed options are categorised in to 3 types. They are

(1) Division into process

(2) Interpreter

(3) P-code compiler

(1) Division into process:-

In this design option, the compiler was driven by the passing process, the lexical parser was called when the parser needed another input token and a code generation routine was invoked as each language was recognised by the parser. All the languages are not translated by one pass compiler in PASCAL, Declaration of variables must be appeared in the programs before the statement that uses these variables.

Ex:- x := y + z

If the all variables x, y, z are of the type integer, the object code for this statement might consist of a simple integer multiplication followed by storage of result.

If the variables are mixture of real and integer types, one or more operations will be needed to include in the object code and floating point arthmetic instructions may be used.

So programming languages requires passes to compile. There are no. of factors that is considered in deciding one pass and multipass compiler design. If the programs are executed many times, for each compilation then the speed of execution becomes more important than the speed of compilation. In such case, we prefer multi-pass compiler that incorporates sophicasted code optimisation technique.

(2) Interpreter:-

An Interpreter process a source program written in high-level language. The interpreter executes the source program directly instead of translating into machine code. An interpreter performs lexical and syntatic analysis functions and then translates the source program before execution.

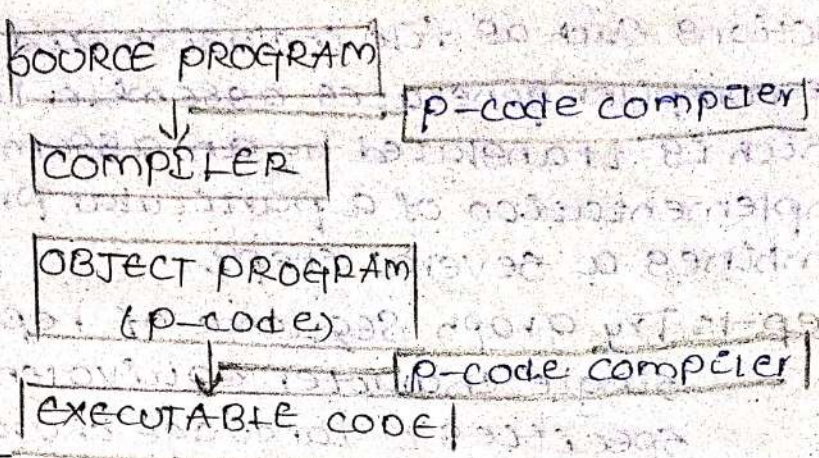
After translating the source program into some internal form, the interpreter executes the operations specified by the program. The process of translating a source program into some internal form is simpler and faster than compiling into machine code.

Advantages:-

The Advantage of an interpreter over a compiler is that it can be easily provided by the de-bugging facilitees. The source line number and source table and other information from the source program are usually retained by the interpreter. Most programming languages can be either compiled & interpreted easily. However some languages are particularly suited to use only interpreter.

(3) P-code compilers-

The p-code compiler also called as byte code compiler or pseudo code compiler. These are also very similar to the concept of interpreters. In both cases, the source program is analysed and converted into an intermediate form is a machine language for Hypothetical computer often called as p-machine. The process of using such p-code compiler is shown below



*Advantages:-

The main advantages of this approach is portability of software. It is not necessary for the compiler to generate different code for different computer. Because p-code object program can be executed on any machine that has a p-code interpreter.

The design of p-machine and associated p-code is often related to the requirement of the language being compiled. For example, p-code for PASCAL compiler include single p-instruction that performs calculation to handle the details of the procedure that is entry and exit and also performs elementary operations on sets.

Implementation Examples:-

In this section we briefly discuss the design and implementation of several compilers. There are different types of implementation examples in compiler they are:-

a) Sun OS compiler:-

It runs on variety of platform including SPARC, X-86 and power pc. It confirms the ANSI standard and also supports ANSI features. This features are accepted by the compilers and also generates the warning message during compilation.

The translation process begins the execution of processor which perform some

functions such as file inclusion and macro processing. The compiler generates assembler language which is translated by an assembler. The implementation of a particular process combines a several steps,

Step-1: Try graph sequence of replaced by their single character equivalence. It also specifies 'c' language character that may not be available on some terminals.

for Ex:- The sequence '??<' is replaced by '{'

Step-2:- Any source line that ends with back slash (\) and a new line is specified together with the following line by deleting backslash (\) newline

Step-3:- pre processing directives are executed and macros are expanded any source line that are included is responsible to # include directives.

Like most implementation of UNIX, SUN OS is largely written in many compilers, editors and other piece of UNIX system softwares are are also written in 'c'. Because of this reason like many other compilers, SUN OS compiler also generates information that supports the operation of debugging.

(2) Java compiler and enviroment:-

Java is a new programming language and operating environment develop by SUN micro system. It is designed to support application in an environment such as internet using Java, we can create high performance application that are portable. It also provides features to make the application more reliable and secure. It is derived from C and c++.

for example, Java has no 'go to' statements and no pointers.

Java is an object oriented language which is stronger than many other languages such as c++ except for a few primitive data types, everything in java are treated as objects.

Java provides building support for multi threads of execution. This feature allows different parts of an application code to be executed concurrently. The Java compiler follows the p-code approach, it does not generate bytecode high level language mission independent language for a hypothetical machine.

The JVM supports a set of primitive data types such as 1, 2, 4 and 8 byte integer single and double precision floating point numbers. This data representation are independent to the target machine. A byte code instructions are machine instructions in JVM that consist of one byte opcode followed by zero (or) more operands.

3. Yet Another compiler compiler (YACC):-

It is a passer generator that is available for UNIX System. YACC has been used in the production of compilers for PASCAL, A, ALGOL, C and other programming languages. It is also used for less convention and applications including type setting language and a document retrival system. A lexical scanner must be supplied with YACC. This scanner is called by PARSER. Whenever a new input token is needed, it returns an integer that identifies the type of token found. The scanner also makes entries in a specific symbol table for the identifiers is as follows:-

(1) Let Return (LET)
(2) * Return (MUL)
(3) = Return (Assigns)
(a-z) (A-z)
(a-z, A-z, 0-9)

The 1st 3 patterns simply returns a token type The token 'LET' for the key word LET, MUL for the operator '*' and assigns for the operator '='. The representation of LET, MUL and other tokens are integers.

The 4th pattern specifies the form of identifiers to be recognised. The first character must be followed by range ((a-z) & (A-z)). This may be followed by any number of characters.

for example, Let x = y * z
It would be scanned as sequence of tokens.
Let ID assign ID MULIP
Note that 1st pattern matches the input string that is selected. So the keyword LET is recognised as token not as an Id.

UNIT-III :-

Chapter-2: other system softwares

Text editor :-

The Text editor has became an important part of almost any computing environment. It is now considered as a primary interface to the computer for all types of knowledge workers as they compose, organise, study and manipulate computer based information. The text editor consist of 3 sections such as:

(1) overview of editing process
(2) user interface
(3) editor structure

1) Overview of editing process:-

It allows the user to create and revise a target document. The term document include objects such as computer programs, text, equations, tables, diagrams, line arts and photographs etc. The document editing process is an interactive user computer design in order to accomplish four tasks such as:

* Select the part of document to be viewed and manipulated
* Determine how to format and how to display it.
* Specify and execute the operations that modify the target document.
* Update the view approximately.

The selection of what is to be viewed & manipulated is controlled by filtering. The filtering extracts relevant subset of target document. In the actual editing phase, the target document is created/ altered with a set of operations such as insert, delete, replace move and copy.

2) User Interface:-

The user of an interactive editor is presented with a conceptual model of editing system. This model is an abstract framework on which the editor operates. This editor allows the operations on numbered sequence of 80 characters lene can be written within a single line/integral no. of lines. The user interface is concerned with input, output devices and the interaction language of the system.

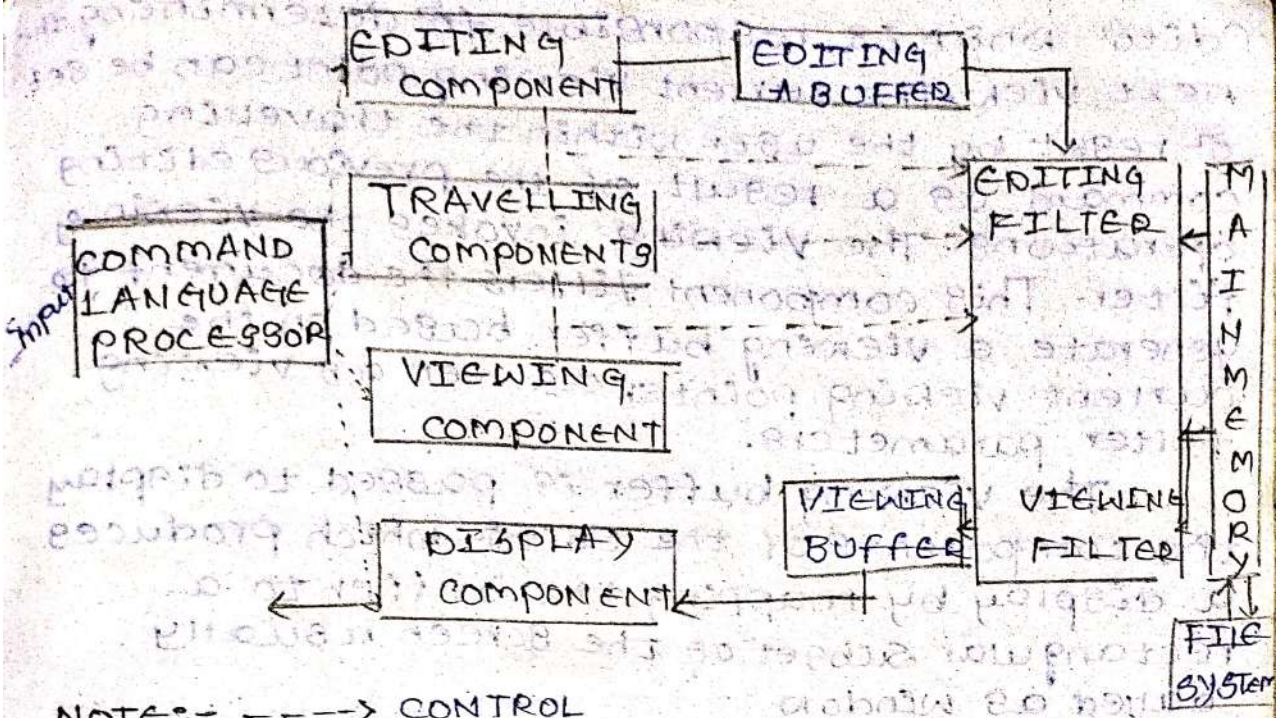
Input devices are used to enter elements the text being edited, to enter the commands and to derive editable elements. This devices are used with the editors that can be derived in to three categories, such as (i) Text/String device (keyboard)
(ii) Button/choice device (
(iii) Locator device (mouse)

(i) Text/string devices:-
These devices are typically typewriter like keyboards and which the user press and release the keys. All the computers are of "QWERTY" variety. These devices are used to enter the text/string by the user.

(ii) Button/choice device:-
These devices generate an interrupt that usually cause invokation of an associated application program. These devices includes a special functional keys on a alphanumeric keyboard. The user choose a string/symbol instead of pressing a button.

(iii) Locator device:-
These are 2D analog to digital converters that place a cursor symbol on the screen. The most cursors devices for editing applications are mouse and datatablets. The datatablet is a flat rectangular electro magnetively sensitive panel.

3. Editor's structures:-
The following diagram shows the editor's structure.



NOTE:- ----> CONTROL
——> DATA

The command language processor accepts input from the user input device and analyse the tokens and syntatic structure of the commands. These command processor may produce an intermediate representation of design editing operations. The syntatic routine involves editing, travelling, viewing and displaying functions.

The relationship between this classes may be considerably complicated by the simple moral by describing the overview of editing process.

In editing a document the area to be edited is determined by the current editing pointer can be set or reset explicity by the user with the travelling commands. The travelling commands of the edito actually perform's the setting of current editing and viewing pointers that determines at which the viewing or editing filter begins

Similarly, in viewing a document, the area to be viewed is determined by the current viewing pointer. This pointer is maintained by viewing the component of

editor which is responsible for determining the next view. The current viewing point can be set & reset by the user within the travelling command as a result of the previous editing operations. The viewing invokes the viewing filter. This component filters the document to generate a viewing buffer based on the current viewing pointer as well as viewing filter parameters.

The viewing buffer is passed to display the components of the editor, which produces a display by mapping the buffer to a rectangular subset of the screen usually called as window.

*Interactive Debugging System [IDS]:-

An interactive debugging system provides programs with facilities that aid in testing & debugging of programs. This section gives us some of the most important requirements for an Interactive Debugging System and provides some basic system considerations that consists of three process, such as:

a) ① Debugging functions & capabilities
② Relationship with other parts of the system
③ user Interface criteria

a) Debugging functions & capabilities:-

It describes some of the most important capabilities of an interactive debugging system. Some of these functions are more difficult to implement. The most obvious requirement for a set of unit test functions, that can be specified by the programmer. One important group of such functions deals with execution sequencing which controls the flow of program.

for example, the program may be after a fixed no. of instructions are executed. Similarly the programmer may define

breakpoints which causes execution to be suspended when a specified point in the program is reached.

A debugging system also provides functions such as tracing and trace back. Tracing can be used to track the flow of execution & trace back shows the path by which the current statement was reached. It also shows the statement that have modified a given variable or parameter.

(2) Relationship with other parts of the system:-

An interactive debugger must be related to the other parts of the system in different ways. One important requirement for an interactive debugger is always available and an integral part of the system. When an error is discovered, immediate debugging is possible because it may be difficult to reproduce the program failure in some other environment. for example, user need to be able to debug in a production environment. Debugging is more important at production time than at development time. When an application fails during a production time, the work will be stopped. It is quite differ from the test environment many program failures cannot be repeated outside the production environment.

The debugger also exist in a way that is consisting with the security and integrety components of the system. It must not be possible for someone to use the debugger to access any data that would not be accessable to that individual.

(3) User interface criteria:-

The behaviour and presentation of an interactive system is crucial to accept

the users. The best way to overcome this difficulty is to have a system that is simple in its organisation and familiar in its language. The facilities of debugging system should be organised into a basic categories of functions which reflects common user tasks.

The user interaction should make the use of full screen display and windowing system as much as possible. The primary advantage is that a great deal of information can be displayed and changed easily & quickly, with menus and the full screen editor, the user has less information to enter & remember.

The use of full screen display & techniques such as menus is highly desirable. For example, there should be complete functional equivalence between the command & menus. The command language should have a clear logical simple syntax. commands should be simple rather than compound and should require few parameters as possible. There should be a consistant use of parameters names across the commands. parameters should automatically checked for errors in search attributes has a type and range of values.