UNIT I -INTRODUCTION TO COMPILER
PART A

1. What is a Complier?

A Complier is a program that reads a program written in one language-the source language-and translates it in to an equivalent program in another language-the target language . As an important part of this translation process, the compiler reports to its user the presence of errors in the source program.

2. State some software tools that manipulate source program?
   - i) Structure editors
   - ii) Pretty printers
   - iii) Static
   - iv) checkers
   - v) Interpreters.

2. What are the cousins of compiler? April/May 2004, April/May 2005
   I. The following are the cousins of
   Preprocessors
   - i. Assemblers
   - ii. Loaders
   - iii. Link editors.

4. What are the main two parts of compilation? What are they performing?
   The two main parts are
   - **Analysis** part breaks up the source program into constituent pieces and creates an intermediate representation of the source program.
   - **Synthesis** part constructs the desired target program from the intermediate representation

5. What is a Structure editor?

A structure editor takes as input a sequence of commands to build a source program .The structure editor not only performs the text creation and modification functions of an ordinary text editor but it also analyzes the program text putting an appropriate hierarchical structure on the source program.

6. What are a Pretty Printer and Static Checker?
   - A Pretty printer analyses a program and prints it in such a way that the structure of the program becomes clearly visible.
   - A static checker reads a program, analyses it and attempts to discover potential bugs without running the program.

7. How many phases does analysis consists?
   Analysis consists of three phases i
   - .Linear analysis
   - ii. Hierarchical analysis
   - iii. Semantic analysis

8. What happens in linear analysis?

This is the phase in which the stream of characters making up the source program is read from left to right and grouped in to tokens that are sequences of characters having collective meaning.

9. What happens in Hierarchical analysis?

This is the phase in which characters or tokens are grouped hierarchically in to nested collections with collective meaning.

10. What happens in Semantic analysis?

This is the phase in which certain checks are performed to ensure that the components of a program fit together meaningfully.

11. State some compiler construction tools?Arpil /May 2008

      i. Parse generator
      ii. Scanner generators
      iii. Syntax-directed translation engines
      iv. Automatic code generator
      v. Data flow engines.

12. What is a Loader? What does the loading process do?

A Loader is a program that performs the two functions
    i. Loading
    ii .Link editing

The process of loading consists of taking relocatable machine code, altering the relocatable address and placing the altered instructions and data in memory at the proper locations.

13. What does the Link Editing does?

Link editing: This allows us to make a single program from several files of relocatable machine code. These files may have been the result of several compilations, and one or more may be library files of routines provided by the system and available to any program that needs them.

14. What is a preprocessor?

A preprocessor is one, which produces input to compilers. A source program may be divided into modules stored in separate files. The task of collecting the source program is sometimes entrusted to a distinct program called a preprocessor.

The preprocessor may also expand macros into source language statements.

Skeletal source program

↓

Preprocessor

↓

Source program
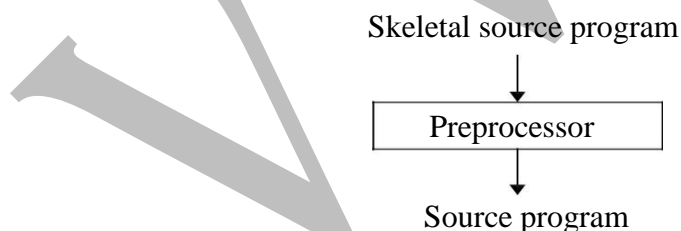
15. State some functions of Preprocessors

    i)   Macro processing
    ii)   File inclusion
    iii)   Relational Preprocessors
    iv)   Language extensions

16. What is a Symbol table?

A Symbol table is a data structure containing a record for each identifier, with fields for the attributes of the identifier. The data structure allows us to find the record for each identifier quickly and to store or retrieve data from that record quickly.

17. State the general phases of a compiler
   i.   Lexical analysis
   ii. Syntax analysis
   iii. Semantic analysis
   iv. Intermediate code generation
   v.   Code optimization
   vi. Code generation

18. What is an assembler?
   Assembler is a program, which converts the source language in to assembly language.

19. What is the need for separating the analysis phase into lexical analysis and parsing? (Or) What are the issues of lexical analyzer?
   • Simpler design is perhaps the most important consideration. The separation of lexical analysis from syntax analysis often allows us to simplify one or the other of these phases.
   • Compiler efficiency is improved.
   • Compiler portability is enhanced.

20. What is Lexical Analysis?
   The first phase of compiler is Lexical Analysis. This is also known as linear analysis in which the stream of characters making up the source program is read from left-to-right and grouped into tokens that are sequences of characters having a collective meaning.

21. What is a lexeme? Define a regular set. Nov/Dec 2006
   • A Lexeme is a sequence of characters in the source program that is matched by the pattern for a token.
   • A language denoted by a regular expression is said to be a regular set

22. What is a sentinel? What is its usage? April/May 2004
   A Sentinel is a special character that cannot be part of the source program. Normally we use 'eof' as the sentinel. This is used for speeding-up the lexical analyzer.

23. What is a regular expression? State the rules, which define regular expression?
   Regular expression is a method to describe regular language
   Rules:
   1) ε-is a regular expression that denotes {ε} that is the set containing the empty string

   2) If a is a symbol in Σ,then a is a regular expression that denotes {a}
   3) Suppose r and s are regular expressions denoting the languages L(r) and L(s) Then,
       a) (r)/(s) is a regular expression denoting L(r) U L(s).
       b) (r)(s) is a regular expression denoting L(r)L(s)
       c) (r)* is a regular expression denoting L(r)*.
       d) (r) is a regular expression denoting L(r).

24. What are the Error-recovery actions in a lexical analyzer?
   1. Deleting an extraneous character
   2. Inserting a missing character
   3. Replacing an incorrect character by a correct character
   4. Transposing two adjacent characters

25. Construct Regular expression for the language
   L= {w ε{a,b}/w ends in abb}
   Ans: {a/b}*abb.

26. What is recognizer?

Recognizers are machines. These are the machines which accept the strings belonging to certain language. If the valid strings of such language are accepted by the machine then it is said that the corresponding language is accepted by that machine, otherwise it is rejected.

PART B

1. Explain the phases of the compiler and how the following relation will be translated in to every phases?

    Position := initial + rate * 60.

2. Discuss in detail about the role of lexical and syntax analyzer with the possible error recovery option.

3. Explain the various phases of the compiler and examine with the program segments.

4. Describe the following expressions after each passes.

    a:= b*c-d how various passes could be combined as pass in the compiler.

5. Explain in detail about compiler construction tool.

6. Describe the errors encountered in different phases of compiler.

7. Define compiler? State different phases of compiler in detail.

8. How to solve the source program to target machine code by using language processing system.

9. Generalize the important terminologies used in programming language basis?

10. Analyze the given expressions 4:*+=cba with different phases of compiler

11. Classify the concept of compiler and interpreter.

UNIT-II LEXICAL ANALYSIS
PART A

1. What is Lexical Analysis?

The first phase of compiler is Lexical Analysis. This is also known as linear analysis in which the stream of characters making up the source program is read from left-to-right and grouped into tokens that are sequences of characters having a collective meaning.

2. What is a lexeme? Define a regular set.

- A Lexeme is a sequence of characters in the source program that is matched by the pattern for a token.
- A language denoted by a regular expression is said to be a regular set

3. What is a sentinel? What is its usage?

A Sentinel is a special character that cannot be part of the source program. Normally we use 'eof' as the sentinel. This is used for speeding-up the lexical analyzer.

4. What is a regular expression? State the rules, which define regular expression?

Regular expression is a method to describe regular language

If a is a symbol in Σ,then a is a regular expression that denotes {a}
Suppose r and s are regular expressions denoting the languages L(r) and L(s) Then,
   a) (r)/(s) is a regular expression denoting L(r) U L(s).
   b) (r)(s) is a regular expression denoting L(r)L(s)
   c) (r)* is a regular expression denoting L(r)*.
   d) (r) is a regular expression denoting L(r).

5. What are the Error-recovery actions in a lexical analyzer?

   1. Deleting an extraneous character
   2. Inserting a missing character
   3. Replacing an incorrect character by a correct character
   4. Transposing two adjacent characters

6. Construct Regular expression for the language

L= {w ε{a,b}/w ends in abb}
Ans: {a/b}*abb.

7. What is recognizer?

Recognizers are machines. These are the machines which accept the strings belonging to certain language. If the valid strings of such language are accepted by the machine then it is said that the corresponding language is accepted by that machine, otherwise it is rejected.

8. Differentiate compiler and interpreter.

Compiler produces a target program whereas an interpreter performs the operations implied by the source program.

9. Write short notes on buffer pair.

Concerns with efficiency issues used with a look ahead on the input
It is a specialized buffering technique used to reduce the overhead required to process an input character. Buffer is divided into two N-character halves. Use two pointers. Used at times when the lexical analyzer needs to look ahead several characters beyond the lexeme for a pattern before a match is announced.

10. Differentiate tokens, patterns, lexeme.

- Tokens- Sequence of characters that have a collective meaning.
- Patterns- There is a set of strings in the input for which the same token is produced as output. This set of strings is described by a rule called a pattern associated with the token
- Lexeme- A sequence of characters in the source program that is matched by the pattern for a token.

11. List the operations on languages.

- **Union -** L U M ={s | s is in L or s is in M}
- **Concatenation** – LM ={st | s is in L and t is in M}
- **Kleene Closure –** L* (zero or more concatenations of L)
- **Positive Closure –** L+ (one or more concatenations of L)

12. Write a regular expression for an identifier.

An identifier is defined as a letter followed by zero or more letters or digits.The regular expression for an identifier is given as **letter (letter | digit)***

13. Mention the various notational shorthands for representing regular expressions.

- One or more instances (+)
- Zero or one instance (?)
- Character classes ([abc] where a,b,c are alphabet symbols denotes the regular expressions a | b | c.)
- Non regular sets

14. What is the function of a hierarchical analysis?

Hierarchical analysis is one in which the tokens are grouped hierarchically into nested collections with collective meaning. Also termed as Parsing.

15. What does a semantic analysis do?

Semantic analysis is one in which certain checks are performed to ensure that components of a program fit together meaningfully. Mainly performs type checking.

PART B

1. What are roles and tasks of a lexical analyzer?
2. Converting a Regular Expression into Deterministic Finite Automata.
3. Discuss in detail about the role of Lexical analyzer with the possible error recovery actions
4. Describe the Input buffering techniques in detail.
5. (i). Solve the following regular expression into minimized DFA.
       (a/b)*baa
   (ii). Comparison between NFA and DFA.
6. Define Lex and Lex specifications? How lexical analyzer is constructed using lex? Write a lex program that recognizer the tokens.
7. Solve the given regular expression (a/b)* abb (a/b)* into NFA using Thompson construction and then to minimized DFA.
8. Explain an algorithm for Lex that recognizes the tokens.
9. Suppose we were to revise the definition of a DFA to allow zero or one transition out of each state on each input symbol. Some regular expressions would then have smaller DFA's than they do under the standard definition of a DFA. Give and generalize an example of one such regular expression.
10. How to converting a Regular Expression directly to a DFA. Explain with one example.

UNIT III - SYNTAX ANALYSIS
PART A

1. Define parser.

Hierarchical analysis is one in which the tokens are grouped hierarchically into nested collections with collective meaning.
Also termed as Parsing.

2. Mention the basic issues in parsing.

There are two important issues in parsing.

* Specification of syntax

* Representation of input after parsing.

3. Why lexical and syntax analyzers are separated out?

Reasons for separating the analysis phase into lexical and syntax analyzers:

* Simpler design.
* Compiler efficiency is improved.
* Compiler portability is enhanced.

4. Define a context free grammar.

A context free grammar G is a collection of the following

* V is a set of non terminals
* T is a set of terminals
* S is a start symbol
* P is a set of production rules

G can be represented as G = (V,T,S,P) Production rules are given in

the following form

Non terminal → (V U T)*

5. Briefly explain the concept of derivation.

Derivation from S means generation of string w from S. For constructing derivation two things are important.

i) Choice of non terminal from several others.

ii) Choice of rule from production rules for corresponding non terminal.

Instead of choosing the arbitrary non terminal one can choose

i) either leftmost derivation – leftmost non terminal in a sentinel form

ii) or rightmost derivation – rightmost non terminal in a sentinel form

6. Define ambiguous grammar.

A grammar G is said to be ambiguous if it generates more than one parse tree for some sentence of language L(G).

i.e. both leftmost and rightmost derivations are same for the given sentence.

7. What is a operator precedence parser?

A grammar is said to be operator precedence if it possess the following properties:

1. No production on the right side is ε.

2. There should not be any production rule possessing two adjacent non terminals at the right hand side.

8. List the properties of LR parser.

1. LR parsers can be constructed to recognize most of the programming languages for which the context free grammar can be written.

2. The class of grammar that can be parsed by LR parser is a superset of class of grammars that can be parsed using predictive parsers.

3. LR parsers work using non backtracking shift reduce technique yet it is efficient one.

9. Mention the types of LR parser.

- SLR parser- simple LR parser
- LALR parser- lookahead LR parser
- Canonical LR parser

10. What are the problems with top down parsing?

The following are the problems associated with top down parsing:

- Backtracking
- Left recursion
- Left factoring
- Ambiguity

11. Write the algorithm for FIRST and FOLLOW.

FIRST

1. If X is terminal, then FIRST(X) IS {X}.

2. If X → ε is a production, then add ε to FIRST(X).

3. If X is non terminal and X → Y1,Y2..Yk is a production, then place a in FIRST(X) if for some
i , a is in FIRST(Yi) , and ε is in all of FIRST(Y1),…FIRST(Yi-1);

FOLLOW

1. Place $ in FOLLOW(S),where S is the start symbol and $ is the input right endmarker.

2. If there is a production A → αBβ, then everything in FIRST(β) except for ε is placed in
FOLLOW(B).

3. If there is a production A → αB, or a production A→ αBβ where FIRST(β) contains ε ,
then everything in FOLLOW(A) is in FOLLOW(B).

12. List the advantages and disadvantages of operator precedence parsing.

Advantages

This type of parsing is simple to implement.

Disadvantages

1. The operator like minus has two different precedence (unary and binary).Hence it is hard to
handle tokens like minus sign.

2. This kind of parsing is applicable to only small class of grammars.

13. What is dangling else problem?

Ambiguity can be eliminated by means of dangling-else grammar which is show below: stmt → if
expr then stmt

| if expr then stmt else stmt | other

14. Write short notes on YACC.

YACC is an automatic tool for generating the parser program.

YACC stands for Yet Another Compiler Compiler which is basically the utility available from
UNIX.

Basically YACC is LALR parser generator.

It can report conflict or ambiguities in the form of error messages.

15. What is meant by handle pruning?

A rightmost derivation in reverse can be obtained by handle pruning.

If w is a sentence of the grammar at hand, then w = γn, where γn is the nth right-sentential form
of some as yet unknown rightmost derivation

$S = \gamma0 => \gamma1\ldots => \gamma n{-}1 => \gamma n = w$

16. Define LR(0) items.

An LR(0) item of a grammar G is a production of G with a dot at some position of the right side. Thus, production A → XYZ yields the four items

A→**.**XYZ

A→X**.**YZ

A→XY**.**Z

A→XYZ**.**

17. What is meant by viable prefixes?

The set of prefixes of right sentential forms that can appear on the stack of a shift-reduce parser are called viable prefixes. An equivalent definition of a viable prefix is that it is a prefix of a right sentential form that does not continue past the right end of the rightmost handle of that sentential form.

18. Define handle.

A handle of a string is a substring that matches the right side of a production, and whose reduction to the nonterminal on the left side of the production represents one step along the reverse of a rightmost derivation.

A handle of a right – sentential form γ is a production A→β and a position of γ where the string β may be found and replaced by A to produce the previous right-sentential form in a rightmost derivation of γ. That is , if S =>αAw =>αβw,then A→β in the position following α is a handle of αβw.

19. What are kernel & non-kernel items?

Kernel items, whish include the initial item, S'→ **.**S, and all items whose dots are not at the left end.

Non-kernel items, which have their dots at the left end.

20. What is phrase level error recovery?

Phrase level error recovery is implemented by filling in the blank entries in the predictive parsing table with pointers to error routines. These routines may change, insert, or delete symbols on the input and issue appropriate error messages. They may also pop from the stack.

PART B

1) Construct a predictive parsing table for the grammar
   E -> E + T / F

   T -> T * F / F F ->

   (E) / id

2) Give the LALR parsing table for the grammar.

S -> L = R / R   L -> *

R / id   R -> L

3) Consider the grammar E ->

TE'

E' -> + TE' / E

T -> FT'

T' -> *FT' / E

F -> (E) / id

Construct a predictive parsing table for the grammar shown above. Verify whether the input string

id + id * id is accepted by the grammar or not.

4) Consider the grammar.

E -> E + T   E -> T

T -> T * F   T -> F

F -> (E) / id

Construct an LR parsing table for the above grammar. Give the moves of the LR parser on id * id + id

5) For the grammar given below, calculate the operator precedence relation and the precedence functions
E -> E + E | E – E |
| E * E | E | E |

| E ^ E | (E) | | -E |

id

6) Compare top down parsing and bottom up parsing methods.

7) What are LR parsers? Explain with a diagram the LR parsing algorithm.

8) What are parser generators?

9) Explain recursive descent parser with appropriate examples.

10) Compare SLR, LALR and LR parses.

UNIT IV - SYNTAX DIRECTED TRANSLATION & RUN TIME ENVIRONMENT
PART A

1. List the different storage allocation strategies.

 The strategies are:
- Static allocation
- Stack allocation
-

Heap allocation

2. What are the contents of activation record?

The activation record is a block of memory used for managing the information needed by a single execution of a procedure. Various fields f activation record are:

Temporary variables Local variables

Saved machine registers Control

link

Access link

Actual parameters

Return values

3. What is dynamic scoping?

In dynamic scoping a use of non-local variable refers to the non-local data declared in most recently called and still active procedure. Therefore each time new findings are set up for local names called procedure. In dynamic scoping symbol tables can be required at run time.

4. Define symbol table.

Symbol table is a data structure used by the compiler to keep track of semantics of the variables. It stores information about scope and binding information about names.

5. What is code motion?

Code motion is an optimization technique in which amount of code in a loop is decreased. This transformation is applicable to the expression that yields the same result independent of the number of times the loop is executed. Such an expression is placed before the loop.

6. What are the properties of optimizing compiler?

The source code should be such that it should produce minimum amount of target code.

There should not be any unreachable code.

Dead code should be completely removed from source language.

The optimizing compilers should apply following code improving transformations on source language.

i) common sub expression elimination

ii) dead code elimination

iii) code movement

iv) strength reduction

7. What are the various ways to pass a parameter in a function?

- Call by value
- Call by reference
- Copy-restore
- Call by name

8. Suggest a suitable approach for computing hash function.

Using hash function we should obtain exact locations of name in symbol table. The hash function should result in uniform distribution of names in symbol table.

The hash function should be such that there will be minimum number of collisions. Collision is such a situation where hash function results in same location for storing the names.

9. Define bottom up parsing?

It attempts to construct a parse tree for an input string is beginning at leaves and working up towards the root (i.e.) reducing a string „w" to the start symbol of a grammar. At each reduction step, a particular substring matching the right side of a production is replaced by the symbol on the left of that production. It is a rightmost derivation and it" s also known as shifts reduce parsing.

10. What are the functions used to create the nodes of syntax trees?

- Mknode (op, left, right)
- Mkleaf (id,entry)
- Mkleaf (num, val)

11. What are the functions for constructing syntax trees for expressions?

i) The construction of a syntax tree for an expression is similar to the translation of the expression into postfix form.

ii) Each node in a syntax tree can be implemented as a record with several fields.

12. Give short note about call-by-name?

Call by name, at every reference to a formal parameter in a procedure body the name of the corresponding actual parameter is evaluated. Access is then made to the effective parameter.

13. How parameters are passed to procedures in call-by-value method?

This mechanism transmits values of the parameters of call to the called program. The transfer is one way only and therefore the only way to returned can be the value of a function.

```
Main ( )
{
print (5);
}
int Void print(int n)
  {
  printf ("%d", n);
  }
```

14. Define static allocations and stack allocations

Static allocation is defined as lays out for all data objects at compile time. Names are bound to storage as a program is compiled, so there is no need for a run time support package.

Stack allocation is defined as process in which manages the run time as a Stack. It is based on the idea of a control stack; storage is organized as a stack, and activation records are pushed and popped as activations begin and end.

15. What are the difficulties with top down parsing?

a) Left recursion

b) Backtracking

c) The order in which alternates are tried can affect the language accepted

d) When failure is reported. We have very little idea where the error actually occurred.

16. Define top down parsing?

It can be viewed as an attempt to find the left most derivation for an input string. It can be viewed as attempting to construct a parse tree for the input starting from the root and creating the nodes of the parse tree in preorder.

17. Define a syntax-directed translation?

Syntax-directed translation specifies the translation of a construct in terms of Attributes associated with its syntactic components. Syntax-directed translation uses a context free grammar to specify the syntactic structure of the input. It is an input- output mapping.

18. Define an attribute. Give the types of an attribute?

An attribute may represent any quantity, with each grammar symbol, it associates a set of attributes and with each production, a set of semantic rules for computing values of the attributes associated with the symbols appearing in that production.

Example: a type, a value, a memory location etc.,

i) Synthesized attributes.

ii) Inherited attributes.

19. Give the 2 attributes of syntax directed translation into 3-addr code?

i) E.place, the name that will hold the value of   E and

ii) E.code , the sequence of 3-addr statements evaluating E.

20. Write the grammar for flow-of-control statements?

The following grammar generates the flow-of-control statements, if-then, if-then-else, and while-do statements. S -> if E

then S1

| If E then S1 else S2 | While E do S1.

PART B

1) Discuss in detail about the run time storage arrangement.
2) What are different storage allocation strategies? Explain.
3) Write in detail about the issues in the design of code generator.
4) Explain how declarations are done in a procedure using syntax directed translations.
5) What is a three address code? Mention its types. How would you implement these address statements? Explain with suitable examples.
6) Write syntax directed translation for arrays.

UNIT V- CODE OPTIMIZATION AND CODE GENERATION

PART A

1. Define code generations.

It is the final phase in compiler model and it takes as an input an intermediate representation of the source program and output produces as equivalent target programs. Then intermediate instructions are each translated into a sequence of machine instructions that perform the same task.

2. What are the issues in the design of code generator?

- Input to the generator
- Target programs
- Memory management
- Instruction selection
- Register allocation
- Choice of evaluation order
- Approaches to code generation.

3. Give the variety of forms in target program.

- Absolute machine language.
- Relocatable machine language.
- Assembly language.

4. Give the factors of instruction selections.

- Uniformity and completeness of the instruction sets
- Instruction speed and machine idioms
- Size of the instruction sets.

5. What are the sub problems in register allocation strategies?

- During register allocation, we select the set of variables that will reside in register at a point in the program.
- During a subsequent register assignment phase, we pick the specific register that a variable reside in.

6. Give the standard storage allocation strategies.

- Static allocation
- Stack allocation.

7. Define static allocations and stack allocations

   Static allocation is defined as lays out for all data objects at compile time. Names are bound to storage as a program is compiled, so there is no need for a Run time support package.
   Stack allocation is defined as process in which manages the run time as a Stack. It is based on the idea of a control stack; storage is organized as a stack, And activation records are pushed and popped as activations begin and end.

8. Write the addressing mode and associated costs in the target machine.

MODE	FORM	ADDRESS	ADDED COST
Absolute	M	M	1
Register	R	R	0
Indexed	c(R)	c+contents(R)	1
Indirect register	*R	contents(R)	0
Indirect indexed	*c(R)	contents(c+contents(R))	1

9. Define basic block and flow graph.

   A basic block is a sequence of consecutive statements in which flow of Control enters at the beginning and leaves at the end without halt or possibility Of branching except at the end.
A flow graph is defined as the adding of flow of control information to the Set of basic blocks making up a program by constructing a directed graph.

10. Write the step to partition a sequence of 3 address statements into basic blocks.

   1. First determine the set of leaders, the first statement of basic blocks.

- The rules we can use are the following. The first statement is a leader.

-
- Any statement that is the target of a conditional or unconditional goto is a leader.
- Any statement that immediately follows a goto or conditional goto statement is a leader. 2.

For each leader, its basic blocks consists of the leader and all statements

Up to but not including the next leader or the end of the program.

11. Give the important classes of local transformations on basic blocks

- Structure preservation transformations
- Algebraic transformations.

12. Describe algebraic transformations.

It can be used to change the set of expressions computed by a basic blocks into A algebraically equivalent sets. The useful ones are those that simplify the

Expressions place expensive operations by cheaper ones. X =

$$X + 0$$
$$X = X * 1$$

13. What is meant by register descriptors and address descriptors?

A register descriptor keeps track of what is currently in each register. It is consulted whenever a new register is needed.

An address descriptor keeps track of the location where ever the current Value of the name can be found at run time. The location might be a register, a Stack location, a memory address.

14. What are the actions to perform the code generation algorithms?

- Invoke a function get reg to determine the location L.
- Consult the address descriptor for y to determine y", the current location of y.
- If the current values of y and/or z have no next uses, are not live on exit from the block, and are in register, alter the register descriptor.

15. Write the code sequence for the d:=(a-b)+(a-c)+(a-c).

Statement	Code generation	Register descriptor	Address descriptor
t:=a-b	MOV a,R0 SUB b,R0	R0 contains t	t in R0
u:=a-c	MOV a,R1 SUB c,R1	R0 contains t R1 contains u	t in R0 u in R1
v:=t+u	ADD R1,R0	R0 contains v R1 contains u	u in R1 v in R0
d:=v+u	ADD R1,R0 MOV R0,d	R0 contains d	d in R0 d in R0 and memory

16. Write the labels on nodes in DAG.

A DAG for a basic block is a directed acyclic graph with the following Labels on nodes:

- Leaves are labeled by unique identifiers, either variable names or constants.
- Interior nodes are labeled by an operator symbol.
- Nodes are also optionally given a sequence of identifiers for labels.

17. Give the applications of DAG.

-
-

Automatically detect the common sub expressions
Determine which identifiers have their values used in the block.
- Determine which statements compute values that could be used outside the blocks.

18. Define Peephole optimization.

A Statement by statement code generation strategy often produces target code that contains redundant instructions and suboptimal constructs. "Optimizing" is misleading because there is no guarantee that the resulting code is optimal. It is a method for trying to improve the performance of the target program by examining the short sequence of target instructions and replacing this instructions by shorter or faster sequence.

19. Write the characteristics of peephole optimization?

- Redundant-instruction elimination
- Flow-of-control optimizations.
- Algebraic simplifications
- Use of machine idioms

20. What are the structure preserving transformations on basic blocks?
- Common sub-expression elimination
- Dead-code elimination
- Renaming of temporary variables
- Interchange of two independent adjacent statement

21. Define Common sub-expression elimination with ex.

It is defined as the process in which eliminate the statements which has the Same expressions. Hence this basic block may be transformed into the equivalent Block.

Ex:
a : =b + c b :=a
- d
c :=b + c

After elimination:
a : =b + c b :=a
- d
c :=a

22. Define Dead-code elimination with ex.

It is defined as the process in which the statement x=y+z appear in a basic block, where x is a dead that is never subsequently used. Then this statement maybe safely removed without changing the value of basic blocks.

23. Define Renaming of temporary variables with ex.

We have the statement u:=b + c ,where u is a new temporary variable, and change all uses of this instance of t to u, then the value of the basic block is not changed.

24. Define reduction in strength with ex.

Reduction in strength replaces expensive operations by equivalent cheaper ones on the target machines. Certain machine instructions are cheaper than others and can often be used as special cases of more expensive operators. Ex:

X^2 is invariably cheaper to implement as $x*x$ than as a call to an exponentiation routine.

25. Define use of machine idioms.

The target machine may have harder instructions to implement certain specific operations efficiently. Detecting situations that permit the use of these instructions can reduce execution time significantly.

26. Define code optimization and optimizing compiler

The **term code-optimization** refers to techniques a compiler can employ in an attempt to produce a better object language program than the most obvious for a given source program. Compilers that apply code-improving transformations are called **Optimizing-compilers.**

PART B

1. What are the issues in the design of code generator? Explain in detail.

2. Discuss about the run time storage management.

3. Explain basic blocks and flow graphs.

4. Explain about transformation on a basic block.

5. Write a code generation algorithm. Explain about the descriptor and function getreg().Give an example.

6. Explain peephole optimization

7. Explain DAG representation of basic blocks.

8. Explain principle sources of code optimization in details.

9. Explain the Source language issues with details.

10. Explain the Storage organization strategies with examples.

11. Explain storage allocation strategy.

12. Explain about Parameter passing.

13. Explain the non local names in runtime storage managements.

14. Explain about activation records and its purpose.

15. Explain about Optimization of basic blocks.

16. Explain the various approaches to compiler development.

17. Explain simple code generator with suitable example.

18. Discuss about the following:

   a) Copy Propagation b) Dead-code Elimination and c) Code motion