Q1 (i)

	System Program	Application Program
1	It manages and controls computer hardware	Enables end-user to perform specific, productive tasks.
2.	It is the O.S that handles devices, etc	It is third party software that needs help of system software
3.	Its function include managing resources of computer, automate its operation and facilitate program development.	They perform specific data processing or computational tasks for user.
4.	Eg. Microsoft Windows, Mac OSX	Eg. Word, Oracle, PPT, etc. Excel 8314

Order of program is as follows:

Editor → Compiler → Macro Processor → Assembler → Linker → Loader

Q1(b) $S \rightarrow Aa \mid b$
$A \rightarrow Ac \mid Sd \mid \varepsilon$

For $i=1$ $S \rightarrow Aa$ is thorough since there is no left recursion.

For $i=2$
$A \rightarrow Sb \mid c$ is modified as
$A \rightarrow Aab \mid c$ which has immediate left recursion. ∴ Modifying rule

$S \rightarrow Aa$
$A \rightarrow Sb \mid c$

Finally, we get

$S \rightarrow Aa$
$A \rightarrow cA'$
$A' \rightarrow abA' \mid \varepsilon$

Q1(c) A program needs memory resources to execute instructions. An activation is the execution of a sub program. In most languages, local variables are allocated when this execution begins.
The storage needed for an activation is organized as an activation record
- An activation record contains all the necessary information required to call a procedure.

It may contain:

- Local data: The data to be used inside a function is called local address temporary data.
- Links: Additional links reqd by the program
- Status - Stores machine status like registers, PC, etc
- Return Value: Stores return value.

Q1(d) Functions of loader include

(1) Allocation: Allocates space in memory where the object program would be loaded for execution.

(2) Relocation: It modifies object program by changing certain instructions so they can be loaded at different addresses from locations originally specified.

(3) Linking: Links two or more object codes and provides information needed to allow references b/w them.

(4) Loader: It brings the object program to memory for execution.

Q.2 For given grammar, construct operator precedence relation matrix, assuming *, + are binary operators and id is terminal symbol and E is non terminal

$E \rightarrow E + E$, $E \rightarrow E * E$, $E \rightarrow id$

Apply op. precedence for id + id * id

→ E → E+E | E*E | id

Precedence Table

	+	*	id	$
+	.>	<.	<.	.>
*	.>	.>	<.	.>
id	.>	.>	.>	.>
$	<.	<.	<.	<.

consider input id1 + id2 * id3

∴ string is $ <. id1 .> + <. id2 .> * <. id3 .> $

scan string from left until seeing .>

scan backwards from right to left till <.

everything between two relations <. a .> forms handle. It can be found as:

if '<.' Push in stack

if '.>' Pop till '<.' is found and reduce

$ <. id1 .> + <. id2 .> * <. id3 .> $

$ <. E + <. id2 .> * <. id3 .> $

$ <. E + <. id2 .> * <. id3 .> $

$ <. E + <. id2 .> * <. id3 .> $

$ <. E + <. E * <. id3 .> $

$ <· E + <· E * E ·> $

$ <· E + E ·> $

$ <· E ·> $

Q.2 (b) Explain the role of code optimization in compiler designing ? Explain peephole optimization along with an example.

→ Optimization is a program transformation technique, which tries to improve the code by making it consume less resources and deliver high speed. It follows 3 rules given below:

① The output code must not change meaning of the program

② Optimization should increase the speed of the program.

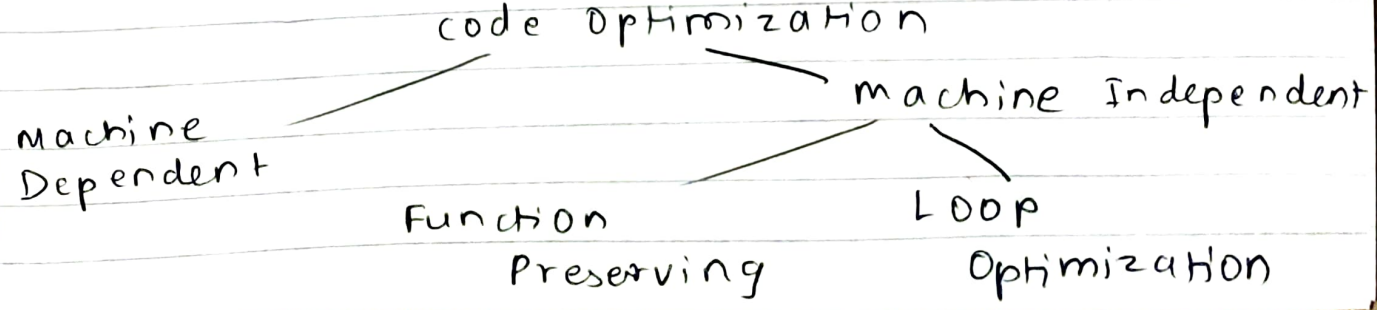
③ It should be fast and should not delay the overall compiling process.

Efforts for an optimized code can be made at various levels of compiling the process. High-level programming constructs are replaced by very efficient low-level codes.

At the begining users can change/rearange the code or use better algorithms.

After generating intermediate code compiler can modify it by address calculations and improving loops.

While producing target code, the compiler can make use of memory hierarchy and CPU registers.



Peephole optimizations examine at most few instructions into other less expensive ones such as turning multiplication of 2x into addition of 'x' twice. It works on a small set of instructions. This small set is called a 'peephole'. It works by recognizing sets of instructions that dont actually do anything, and can be replaced with a leaner set of instructions. Types of peephole optimization:

① constant folding: It finds out constant sub expression in advance

② Strength Reduction: Slow operation are replaced with faster equivalents.

③ Null sequences: Useless operations are removed.

④ combine Operations: Similar operations are replaced by 1 equivalent

⑤ Algebraic Laws: Are used to simplify or reorder instructions

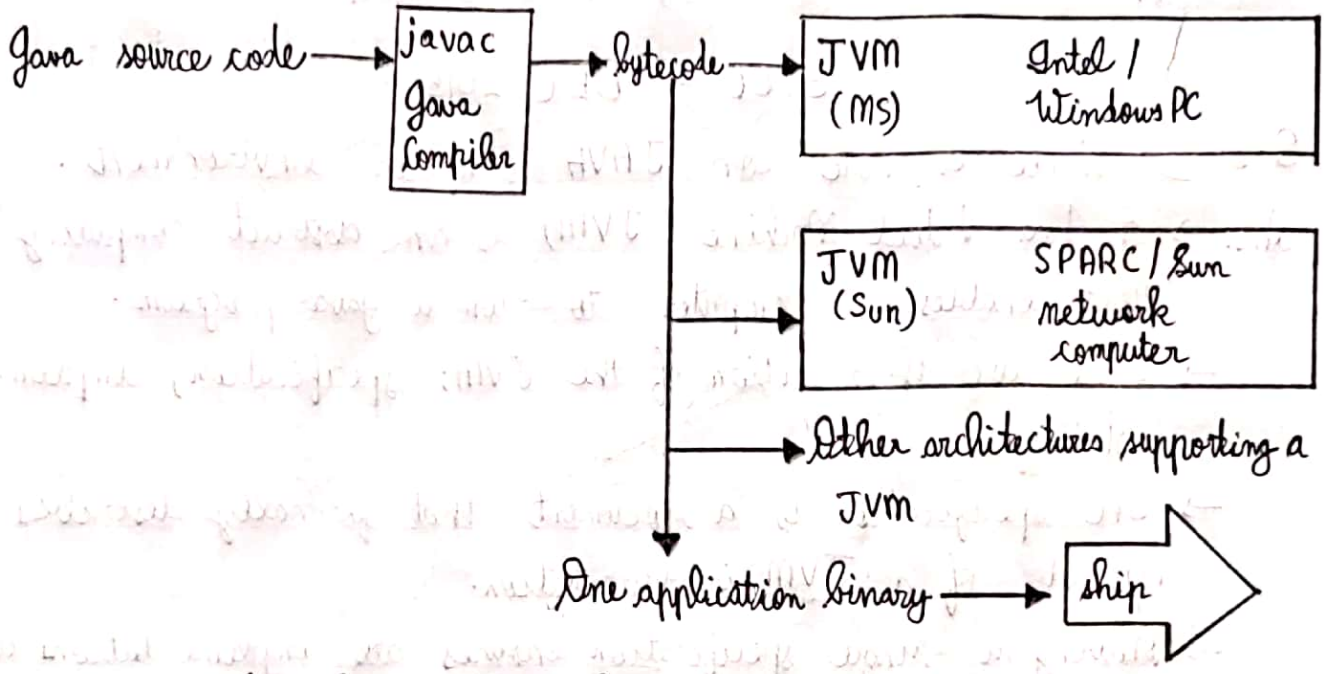
eg.① MOV x, R0
MOV R0, R1
we can delete the first instruction and re-write as MOV x, R1

② a=a+0 can be replaced by a itself
a=a+1 can be replaced by INCR a

SPCC – DEC 2015

Q.3 (a) Write a note on JAVA compiler environment.

Ans:
→ A Java Virtual Machine (JVM) is an abstract computing machine that enables a computer to run a Java program.
→ There are three notion of the JVM: specification, implementation & instance.
→ The specification is a document that formally describes what is required of a JVM implementation.
→ Having a single specification ensures all implementations are interoperable.
→ A JVM implementation is a computer program that meets the requirements of the JVM specification.
→ An instance of a JVM is an implementation running in a process that executes a computer program compiled into Java byte code.
→ The javac command compiles Java source code into Java bytecodes.
→ You then use the Java interpreter – the java command – to interpret the Java bytecodes.
→ Java source code must be contained in files whose filenames end with the .java extension.
→ The file name must be constructed from the classname, as classnam.java if its public or is referenced from another source file.
→ For every class defined in each source file compiled by javac, the compiler stores the resulting bytecodes in a class file with a name of the form classnam.class.
→ It's location is contained by the CLASSPATH environment variable.
→ The compiler looks in the class path for both a source file & a class file, recompiling the source if it is newer.



Java Compile-Time Environment

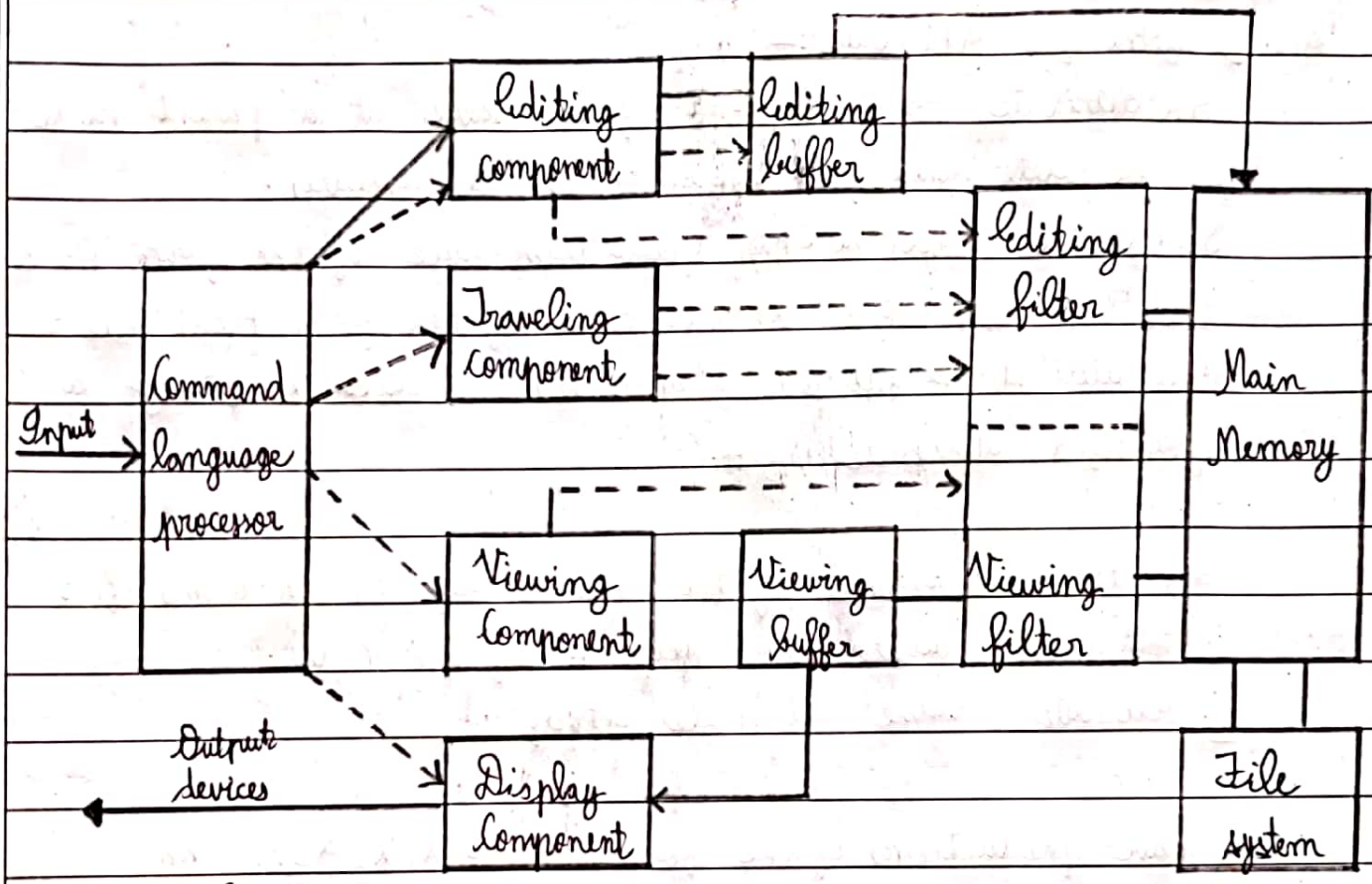
Q.3.(b) Write a brief note on Design of a editor.

Ans: → A text editor is a tool that allows a user to create and revis document in a computer.

→ Though this task can be carried out in other modes, the word text editor commonly refers to the tool that does this interactively.

→ Most text editors have a structure similar to that shown in the figure.

→ Command language processor accepts command, uses semantic routines - performs functions such as editing and viewing.



Typical Editor Structure

→ Editing operations are specified explicitly by the user and display operations are specified implicitly by the editor.

→ In editing a document, the start of the area to be edited is determined by the current editing pointer maintained by the editing component.

→ Filtering and editing may be interleaved, with no explicit editor buffer being created.

→ Loading an entire document into main memory may be infeasible, only a part of the document is loaded.

Q.3.c) Explain synthesized and inherited attributes used in Syntax Directed Definition.

Ans: 1) Synthesized Attributes :-

An attribute is synthesized if its value at a parent node can be determined from attributes of its children.

In general, given a CFG production rule of the form 'A → a B g' then an associated semantic rule of the form "A.attribute = f(a.attribute, B.attribute, g.attribute)" is said to specify a synthesized attribute of A.

Synthesized attribute of the node is defined in terms of :

i) Attribute values at children of the node.

ii) Attribute value at node itself.

We have productions of the form : $A \rightarrow X_1 X_2 X_3 \ldots X_n$

→ with semantic rules of the form : $b = f(c_1, c_2, c_3, \ldots c_n)$

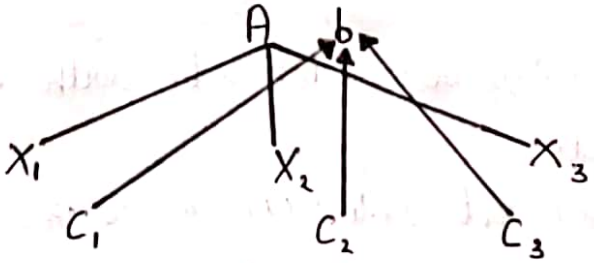
→ where b and the c's are attributes of the grammar symbols

→ b is called a synthesized attribute if :

a) b is an attribute of A (i.e. the LHS), and

b) The c's are all attributes of the X's (symbols on the RHS).

Synthesized attributes can be evaluated by a single bottom-up traversal of the parse tree.



Synthesized Attributes

2) Inherited Attributes :-

An attribute is inherited if the attribute value of a parse-tree node is determined from attribute values of its parents & siblings.

In general, given a CFG production rule of the form 'A = a Bg' then an associated semantic rule of the form "B.attribute = f(a.attribute, A.attribute, g.attribute)" is said to specify an inherited attribute of B.

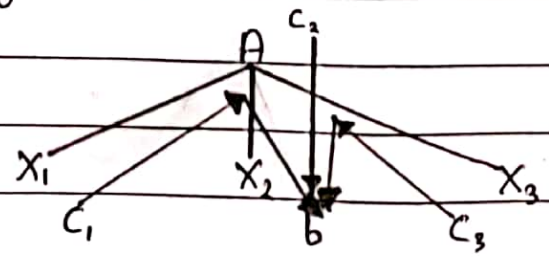
The inherited attributes are passed down from parent nodes to the children nodes of the abstract syntax tree during the semantic analysis of the parsing process.

Inherited attribute of the node is defined in terms of :

i) Attribute values at parent node.
ii) Attribute values at sibling.
iii) Attribute values of node itself.

→ We have the production of the form : $A \rightarrow X_1, X_2, X_3, \ldots X_n$.
→ Semantic rules of the form : $b := f(c_1, c_2, c_3, \ldots, c_n)$
→ b is an inherited attribute if :

a) b is an attribute of one of the X's (i.e. RHS).
b) The c's are the attributes of A and/or one or more of the other X's.
c) Which means they are beside or above where b is needed.



Inherited Attributes

Q.3.d) Find First & Follow Set for given grammar below :-

$E \rightarrow TE'$ $\quad E' \rightarrow +TE' \mid \varepsilon$

$T \rightarrow FT'$ $\quad T' \rightarrow *FT' \mid \varepsilon$

$F \rightarrow (E)$ $\quad F \rightarrow id$

Ans: First sets :-

First (F) = { (, id }

First (T') = { * , ε }

First (T) = { (, id }

First (E') = { +, ε }

First (E) = { (, id }

First (TE') = { (, id }

First (+TE') = { + }

First (ε) = { ε }

First (FT') = { (, id }

First (*FT') = { * }

Follow sets :-

Follow (E) = { $,) }

Follow (E') = { $,) }

Follow (T) = { +,), $ }

Follow (T') = { + ,), $ }

Follow (F) = { + , * ,), $ }

SPCC: December 2015

Q.4]

Ans 1] Dynamic linking loader:

① Dynamic linking loader is a general re-locatable loader that loads external shared libraries into a running process and binds the shared libraries dynamically to the running process.

② This permits a program to load and unload routines at runtime, saving time and memory.

③ Design: The assembler provides four types of records in object file:

a) External symbol Dictionary: The ESD record contains information about all symbols defined in a program. It contains three types of definitions: Segment definition, Local Definition and External Reference

b) Text cards: They contain the actual object code translated version of the source program.

c) Relocation and linkage Directory: The RLD contains information about locations in program whose contents depend on address at which program is placed.

d) END card: This specifies end of object file and specifies start address for execution.

④ Dynamic link library (DLL) is Microsoft's implementation of shared libraries in the Windows operating system. In unix operating system, dynamic linker is an external executable that the kernel loads and executes. When an executable file is loaded, the OS kernel reads path of dynamic linker from it and attempts to load and execute the other binary.

(P.T.O.)

Example:

```
                                    Relative location (RL)
PG1   START                         0
ENTRY A,B                           4
EXTRN PG2,C                         8
      :
A     DC                            20
B     DC                            24
      :
      A+20                          40
      B-25                          44
      C-5                           48
      :
      END                           52
```

ESD:

Symbol	Type	Id	RL	Length
PG1	SD	1	0	52
A	LD	1	20	
B	LD	1	24	
PG2	ER	2	-	
C	ER	3	-	

RLD:

ESD Id	Symbol	Flag	RL
1	A	+	40
1	B	-	44
3	C	-	48

Ans 2] $S \rightarrow F$; $S \rightarrow (S-F)$; $F \rightarrow a$

First (S) = { a, (} Follow (S) = { $ }

First (F) = { a } Follow (F) = { - }

LL(1) Table:

Non-Terminals	Input Symbols			
	a	-	$	(
S	$S \rightarrow F$	$S \rightarrow (S-F)$		$S \rightarrow (S-F)$
F	$F \rightarrow a$			

On parsing string (a-a)

Stack	Input	Output
$ S	(a-a)	S → (S-F)
$) S-F (	(a-a)	S → (S-F)
$) S-F	a-a)	F → a
$) S-a	a-a)	F → a
$) S-	-a)	
$ S	a)	S → F
$ F	a)	F → a
$ a	a)	F → a
$)	)	
$	Accept	

Question 5

(4) Explain different pseudo ops used for conditional macro expansion along with an example.

Ans In assembly language programs, we can combine a group of repititive statements into a single group that can be invoked several times. This is called a Macro. A conditional macro is a mechanism that allows a single version of source code of a program to be used to generate multiple versions of the executable. Conditional assembly can be achieved using:-

1) AIF statement:

- Used to specify branching condition
- provides conditional branching facility
- syntax: AIF (condition) Label
- if condition is satisfied, label is executed, else it continues with next execution.
- performs arithmetic test & branches only if tested condition is true

2) AGO statement:

- provides unconditional branching facility
- no condition specified
- syntax: AGO Label
- macroprocessor continues sequential processing of instructions with the indicated statement.
- these statements are directives to macroprocessor and do not appear in macro expansion.

Example:

```
MACRO
 EVAL 1   &A, &B
 AIF ( &A EQ &B ).NEXT
 LOAD  &A
 SUB   &B
 ADD   &C
 AGO   .FIN
.NEXT   LOAD &C
.FIN    MEND
```

```
        EVAL 1      X, Y, Z
```

LOAD X
SUB Y
ADD Z

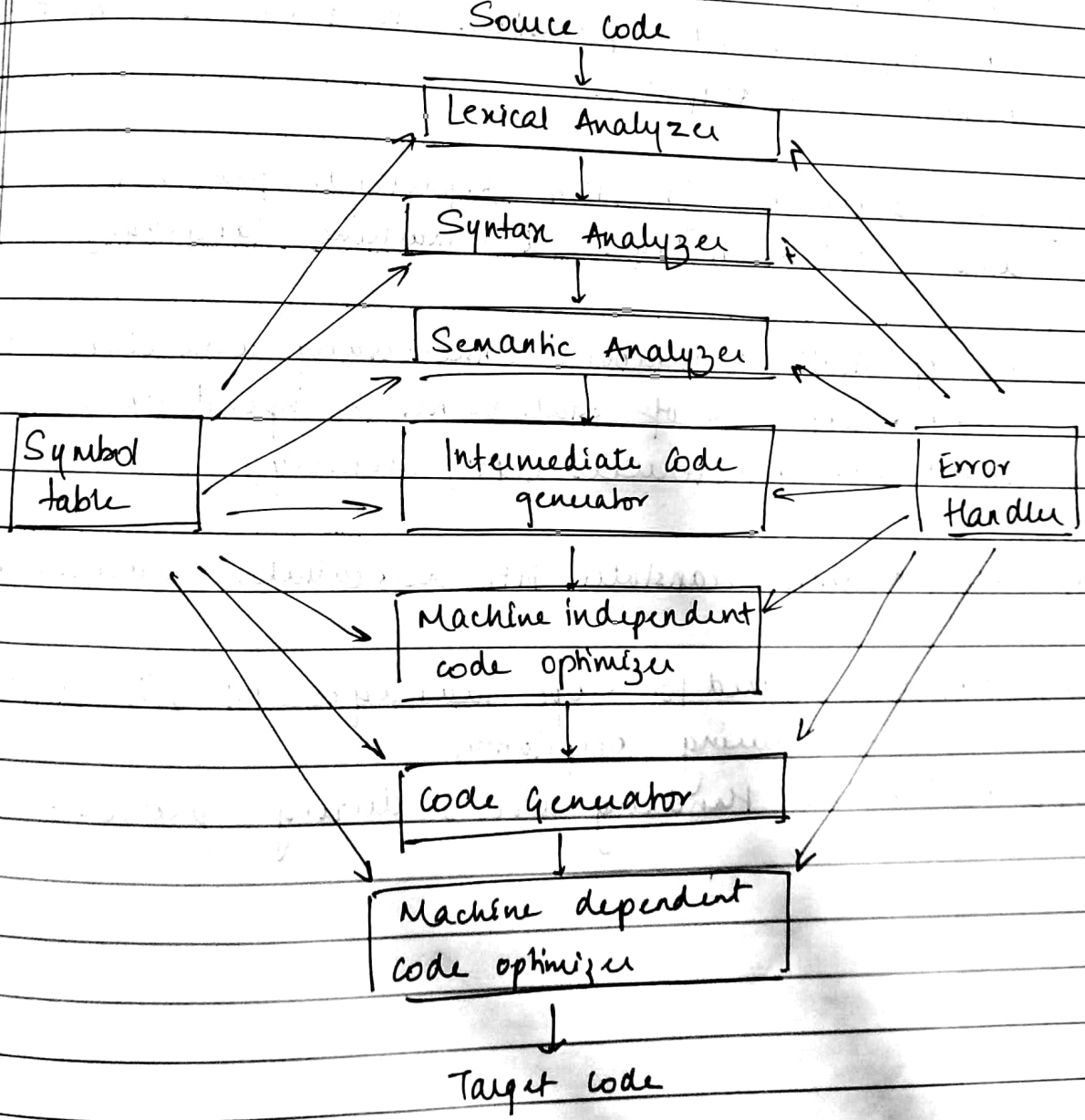
```
        EVAL 1      X, Y, Z  ---  LOAD Z

        END
```

(B) What are the different phases of Compiler? Illustrate internal representation of source program for the following statement after each phase:

position = initial - rate × 60

Ans. A Compiler is a language translator that takes as input source program & generates object program. Compilation of a program executes through a fixed series of phases. Each uses form of program produced by earlier phase. Subsequent phases operate on lower level code representation



Phases of Compiler

1) Lexical Analysis - works as text scanner, scans source code as a stream of characters & converts into lexemes (units) in the form of tokens. (eg: identifiers, constants, operators etc)

2) Syntax Analysis - (Parsing). Takes tokens as input and generates parse tree. Token arrangement is checked against source code grammar, ie. syntax is checked.

3) Semantic Analysis - checks whether parse tree is constructed according to rules of the language, and keeps track of identifiers, types and expressions. It produces annotated syntax tree as output.

4) Intermediate code generation - Represents program for abstract machine (in between high level & machine language)

5) Code optimization - Removes unnecessary code lines and arranges sequence of statements to speed up execution without wasting resources (CPU/ memory)

6) code generation - Translates into relocatable machine code.

7) Symbol table - Used for scope management, Data structure maintained during compilation

8) Error handler - Handling errors during process.

Lexical Analysis

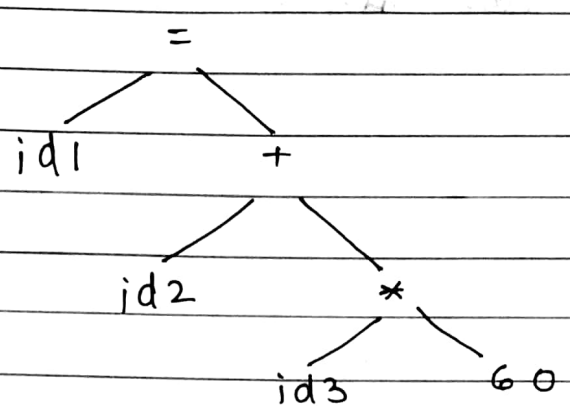
Position = initial + rate * 60

↓

id1 = id2 + id3 * 60

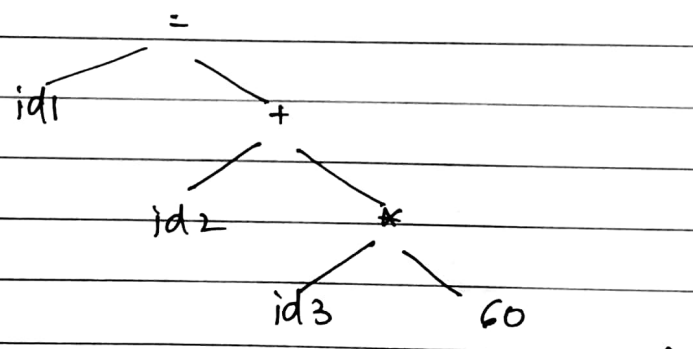
↓

Syntax Analysis



↓

Semantic Analysis



(int to real)

↓

Intermediate code generation

t1 = 60.0

t2 = id3 * t1

t3 = id2 + t2

id1 = t3

↓

Code optimization

t2 = id3 * 60.0

id1 = id2 + t2

↓

Date .....
Page .....

code generation

```
MOV R1, id3
MUL R1, 60.0
MOV R2, id2
ADD R1, R2
MOV id1, R1
```

8312 ★SPCC★

Q.6 (a)

Pseudo opcode table (POT):

- POT is fixed in length.
- In pass 1, using of pseudo opcode, POT is consulted for processing some pseudo opcodes like DS, DC, START, END, etc.
- In pass 2 using pseudo opcode POT is consulted for processing some pseudo opcode like DS, DC, USING DROP.

Pseudo opcode	Address of routine to process pseudo opcode
START	
USING	
DROP	
EQU	
DC	
DS	

START:
It is used to specify the starting execution of a program

USING:
It specifies the base table register that is been used

DROP:
used to remove the register in Base table.

EQU:
used to make program more readable, whenever EQU is added memory is allocated only a entry would make in symbol table.

Declare constant (DC):
It is used to declare or define the values.

Declare storage (DS):
It is used to store the value at specified address.

Machine opcode Table (MOT):

- MOT is a fixed length i.e we make no entry in either of the passes.
- It is used to accept the instructions & corresponding its binary opcode.
- In pass 1, using mnemonic opcode, MOT is consulted to obtain/update location counter (LC).
- In pass2, using mnemonic opcode, MOT is consulted to obtain.

1.) Binary opcode (to generate the instruction)
2) instruction length (to update the instruction)
3.) Instruction format. (to assemble the instruction)

M/C Instruction	Binary opcode	Length	Instruction format
L		4	
A		4	
ST		4	

- LI, data Load the data into register.
- AI, data Add the data into register
- STI, temp store the content of register to temp (const)

Instruction:
- RR (Register, Register)
- RX (Register, Address)

  Address = Base + Index + displacement.

Symbol Table (ST)

- Symbol table is used for keeping the track of symbol that are defined in the program.
- It is used to give a location for a symbol specified
- Symbol is said to be defined it appears in a label field.

- In pass 1, whenever a symbol is defined and for entry is made in symbol table.
- In pass 2, symbol table is used for generating address of a symbol.

Symbol	value	length	Relative/Absolute
PGR	00	01	R
FIVE	16	04	R
FOUR	20	04	R
BASE	10	01	A
TEMP	24	04	R

Literal Table

- Literal table is used to keep track of literals that are encountered in the programs
- We directly specify the name, literal is used to give a location for the name.
- Literals are always encountered in operand field of an instruction.

- On pass I, whenever a literal is ~~not~~ defined a for entry is made in literal table.

- On pass 2, literal table is used to generate address of a literal.

Literal	value	Length	Relative / Absolute
=F'5'	28	04	R

8312 *SPCC*

Q.6 (b)

In the process of code generation (mainly, 3 address code), all the labels may not be known in a single pass hence, we use a technique called backpatching.

Backpatching is the activity of filling up unspecified information of labels using appropriate semantic actions during the process of code generation

Backpatching Algorithms perform three types of operations

1) Makelist (i) - creates a new list containing only i, an index into the array of quadruples and returns a pointer to the list it has made.

2) Merge (i, j) - concatenates the lists pointed to by i and j and returns a pointer to the concatenated list.

3) Backpatch (p, i) - inserts i as the target label for each of the statements on the list pointed to by p

here 'p' is the pointer to any statement say

p => "if a<b goto ------".

initially, we don't know the label to fill just after goto

After backpatch (p, i)

p => "if a<b goto i"

Example 9 Let us consider again the expression

a < b

We only need to apply the above production 5 leading to

100 if a < b goto ____

101 goto ____

Moreover this expression, say E_1, is associated with the lists

E_1.truelist = [100]

E_1.falselist = [101]

and nextaddr has value 102.