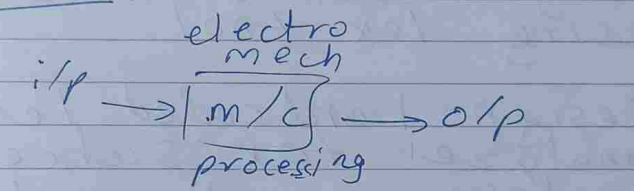
C C

Compiler :- software → App s/w
                    → Sys s/w

Translator
s/w used to drive h/w

Computer :-



Computer is an electro-mech device / machine used for computation.

Software :-

Practicals:-

1. File Handling :- I/p text file o/p with line numbers, total lines & words
2. Design lexi analyzer for subset of C Language using lex tool.
3. Design a scientific calc using lex tool.
4. Design hand coded lex analyzer for subset of C language. Draw the transition diag and then implement lex analyzer
5. Write a code for finding first and follow set for grammar (any language)
6. Write a code to design SQL / HTML parser.
7. Implement SLR parser for given grammar

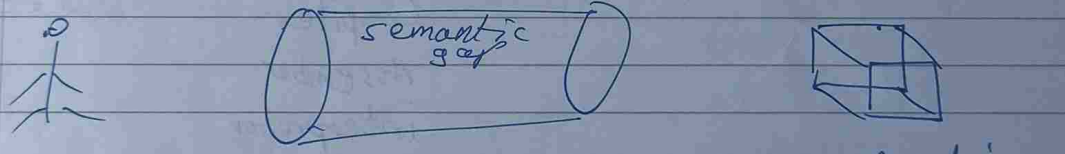
8. Implement static semantic analyzer.

9. Implement intermiediate code gen in 3 address code form, represented in quadruples

10. Implement the diff optimization techniques on intermiediate code (m/c indep → 10 techs
dep → few

OREILLY
Lex and Yack, John Levine
Tony Mason

Software

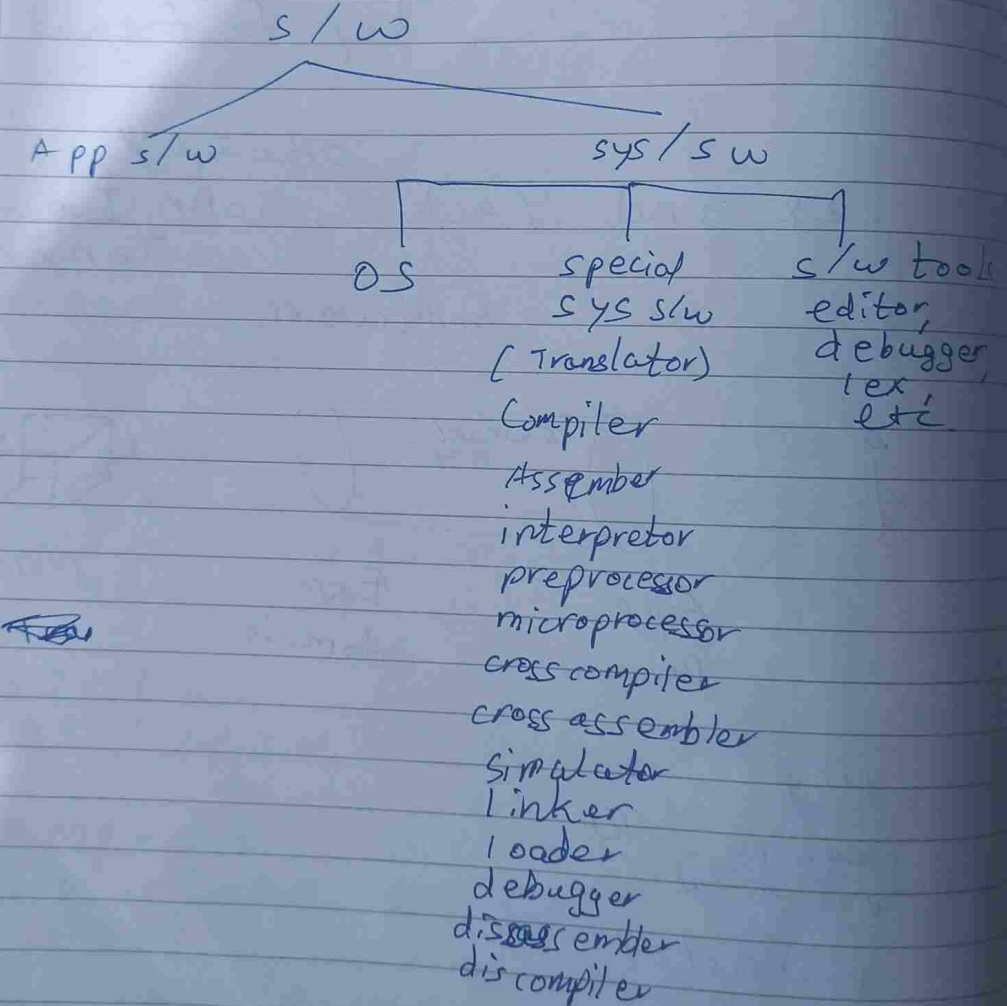


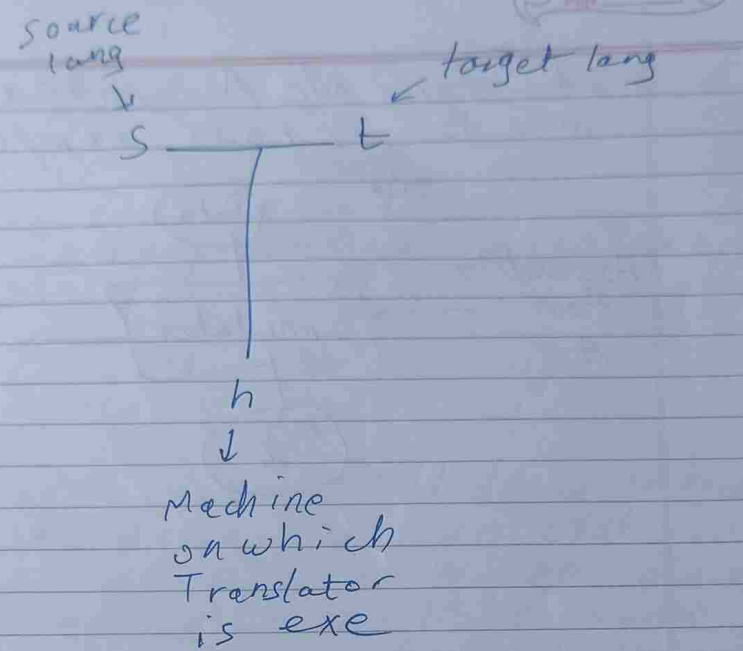
App Domain | Exe domain | Machine

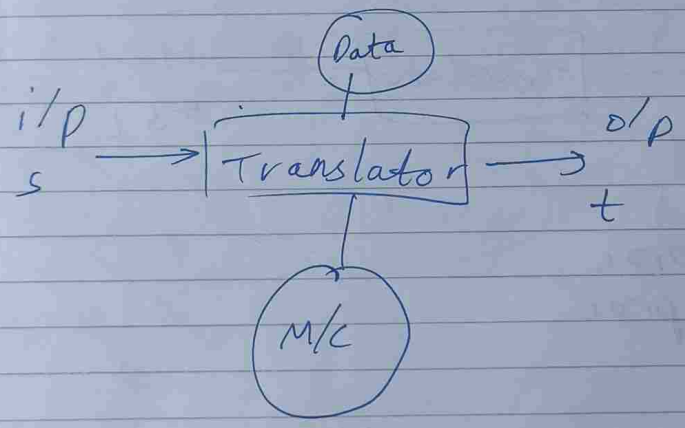
H/w
↓
toggling switch
vacuum tube
↓
transistor
↓
SSI
↓
MSI
↓
VLSI

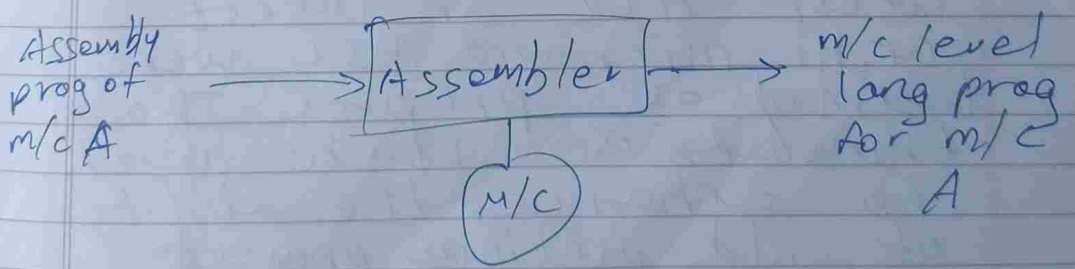
machine → stable, structured, predictable

s/w → The semantic gap in between app domain & exe domain is breached by a logical entity is called software
logical entity → group of progs

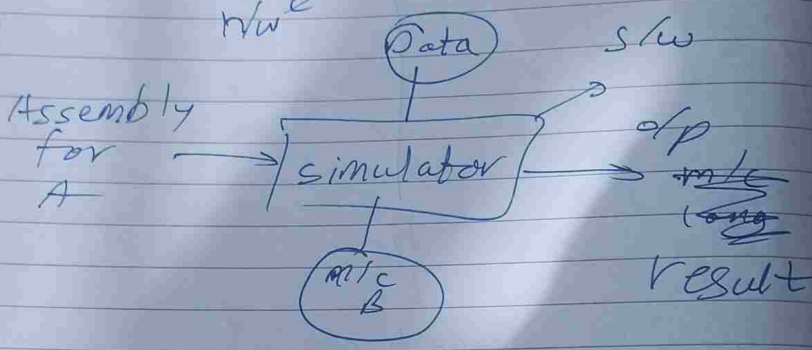


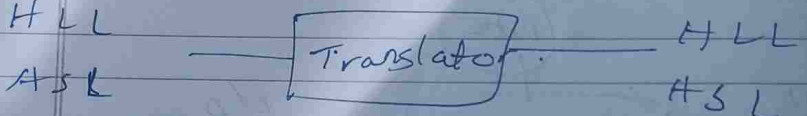

source lang
target lang
S
T
h
Machine on which Translator is exe


Data
i/p
s
Translator
o/p
t
M/c


Assembly prog of m/c A
Assembler
m/c level lang prog for m/c A
M/c

If for m/c A and exe on m/c B, cross compiler (lang)





Preproc
macroproc

A prog written in HLL or ASL into another latest HLL or latest archi micropro ASL were termed as preproc. These translators were used to enhance the ability of old precious progects

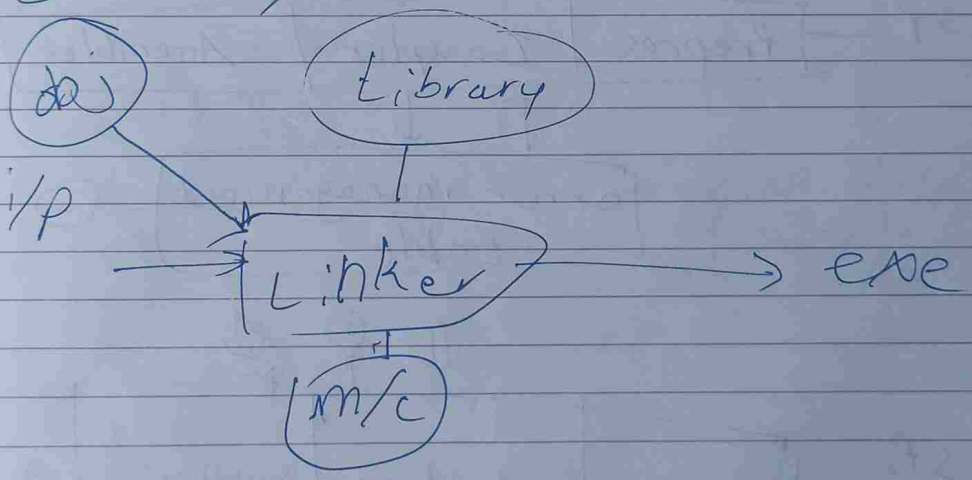
obj not exe

to accomodate new features of HCL

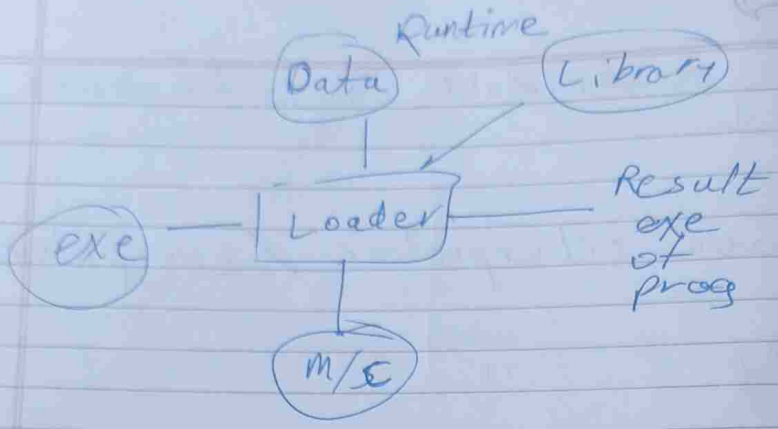
Java interpretor / compiler ?

Diff "

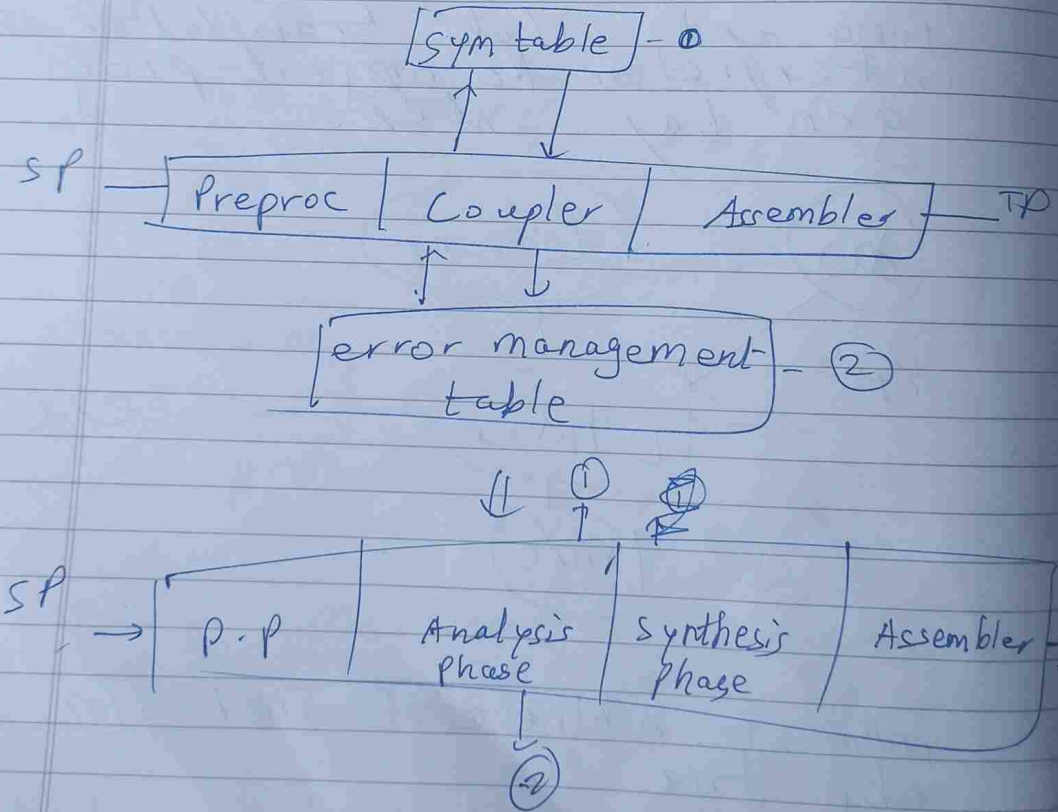
HLL to simplified lang which can be directly exe and will provide result on given i/p data is called intermediate code lang and that translator is interpretor. No object prog gen by inter.

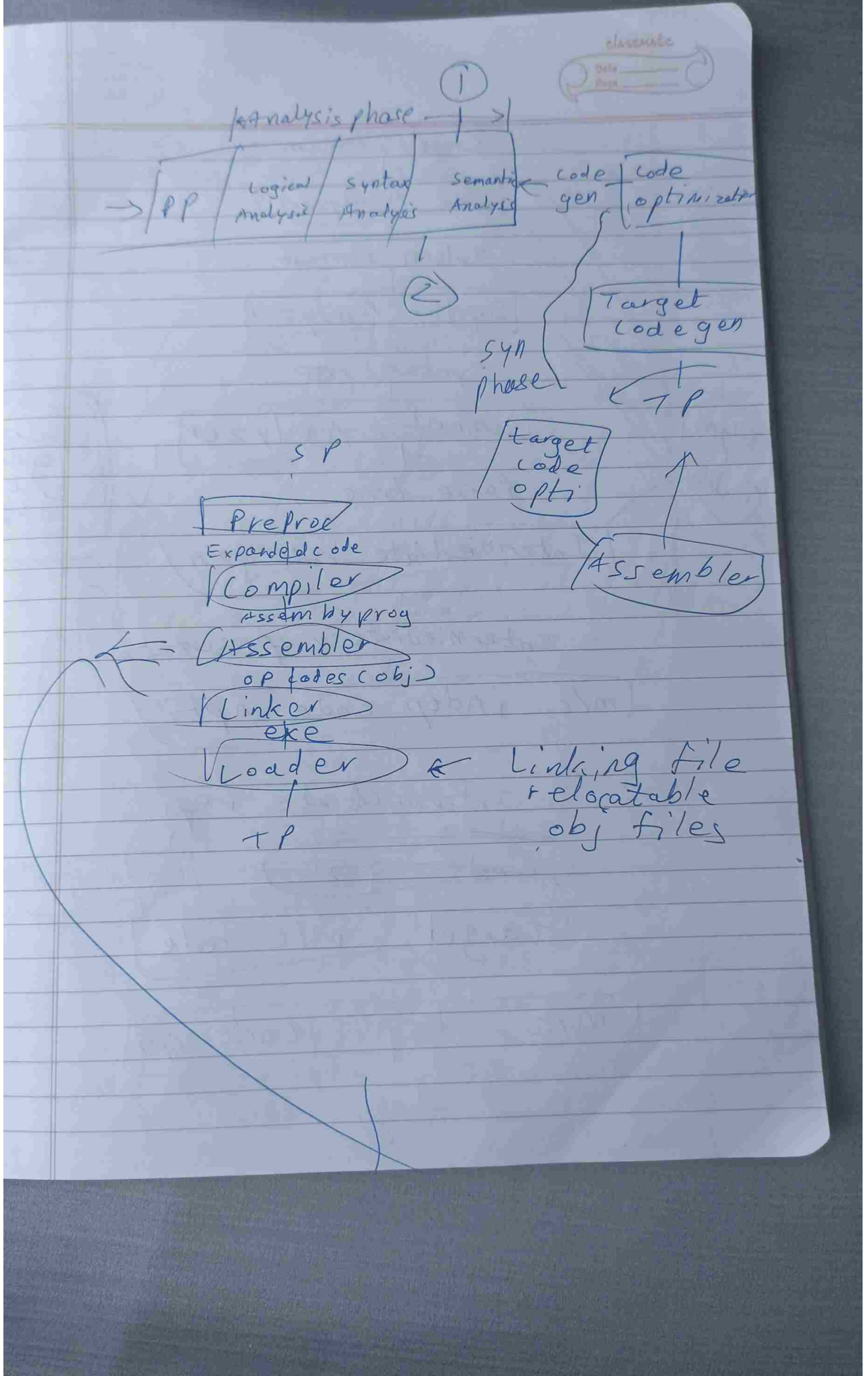


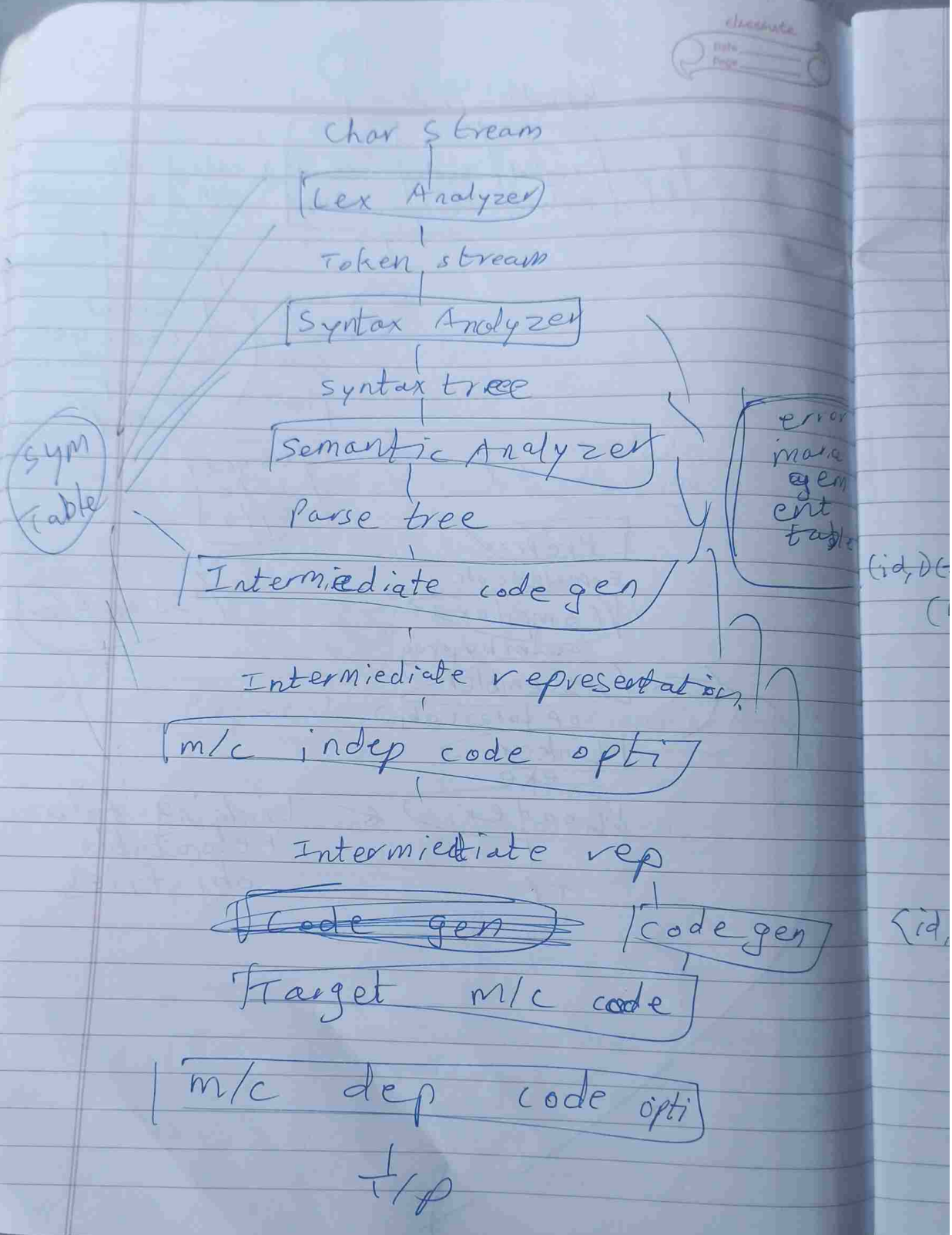
into of addresses of inst in obj file



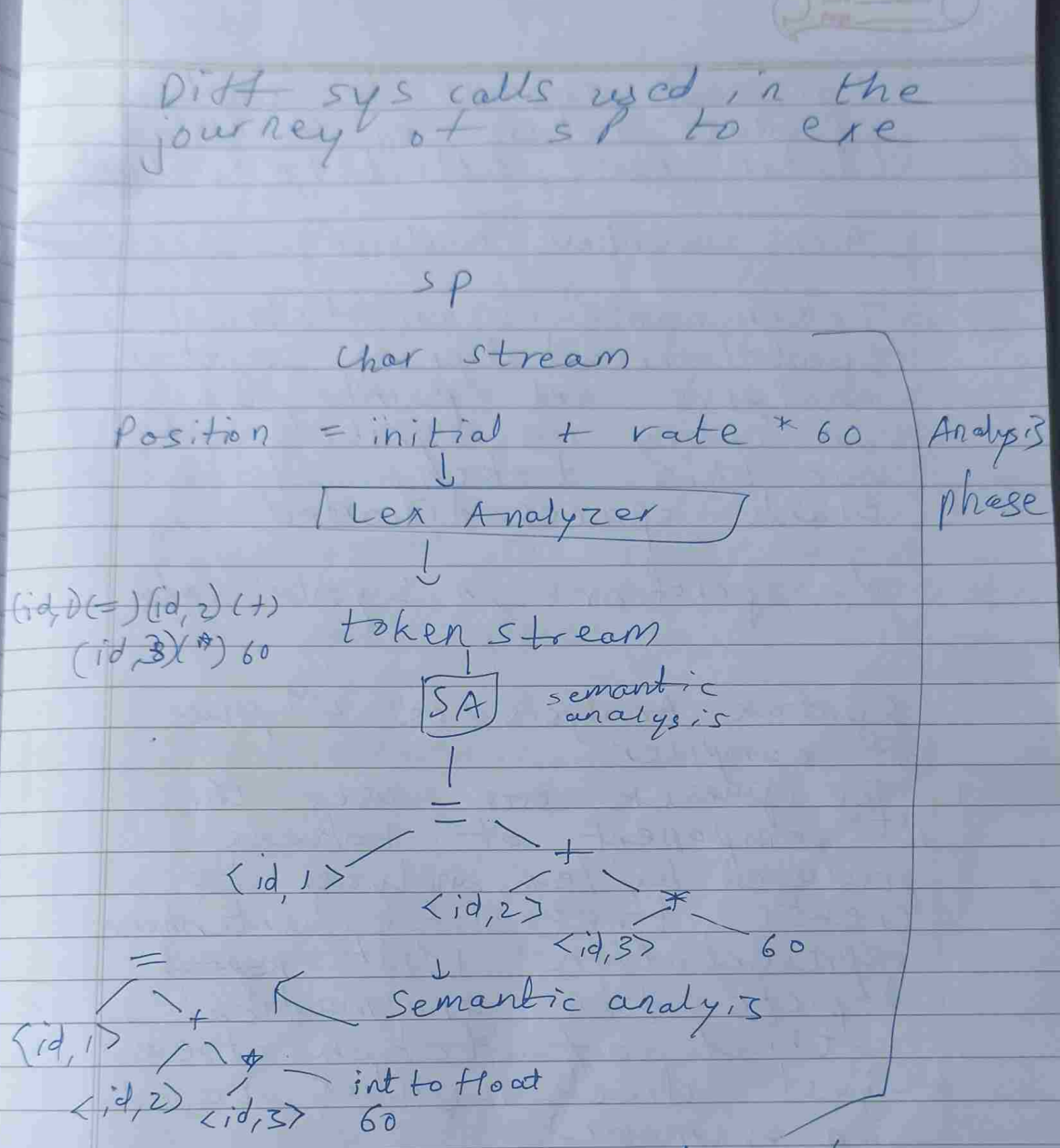
All the obj code module assumes their same starting address




Analysis phase
1
PP
Logical Analysis
Syntax Analysis
Semantic Analysis
code gen
code optimization
2
Target code gen
Syn phase
TP
target code opti
SP
Preproc
Expanded src code
Compiler
assembly prog
Assembler
op codes (obj)
Linker
exe
Loader
TP
Linking file
relocatable obj files


Char stream
Lex Analyzer
Token stream
Syntax Analyzer
syntax tree
Semantic Analyzer
Parse tree
Intermiediate code gen
Intermiediate representation
m/c indep code opti
Intermiediate rep
code gen
Target m/c code
m/c dep code opti
I/p
sym table
error
{id,

Diff sys calls used in the journey of sp to exe



Lex Analyzer reads stream of char from source prog and groups chars into a meaningful seq called lexeme

For each lexeme, lex analyzer produce token in form of (token name, attribute value)

Then syntax analysis

Token name is an abstract symbol used during syntax analysis and points to an entry in symbol table for this token.
Blank spaces discarded

No operators in symbol table

Syntax Analysis is 2nd phase of compiler.

1. The parser uses the 1st component of tokens produced by lex analyzer to create tree-like intermediate representation that depicts the grammatical struct of token stream (just check syntax based on grammar)

A typical rep of syntax tree in which each interior node represents operation and children node

represents the arguments of the operation.

Semantic 3rd Phase

1. Uses the syntax tree and info in symbol table to check the meaning of the program (checks source prog for semantic consistency with language definition)
Important is type checking
Output is parse tree needed for next phase.

Intermediate code generator

↓

$t_1 = \text{inttofloat}(60)$
$t_2 = id3 * t_1$
$t_3 = id_2 + t_2$
$id_1 = t_3$

code optimization

ICG (3 Address code)

After syntax and semantic many compiler gen an explicit low level or machine like inter rep (aprog for abstract m/c)

This intermiediate rep should have 2 imp properties.

1. It should be easy to prodace.
2. It should be easy to translate into target m/c

Gen intermidiate code is rep into popular format known as 3 Address code in which max 3 vars used to rep statement

M/c indep optimization

$t_1 = id_3 * 60.0$

$id_1 = id_2 + t_2$

The m/c indep code opti phase attempts to improve the intermediate code so that better target code will result.

Code → Faster, shorter, consumes less memory.

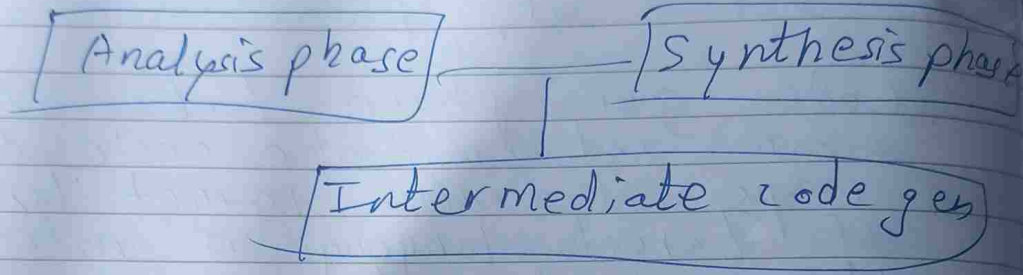
Code Generation :-

```
LD   R2, id3
MULI R2, 60.0
LD   R1, id2
ADD  R1, R2
ST   id1, R1
```

Code gen phase takes i/p from intermediate code and maps it into target language
If target language is m/c code the registers/mem loc are selected for each vars used by prog
Intermediate inst are trans into same seq of m/c inst to perform same task
The main task of code gen is assignment of regs to hold var values

Symbol Table Management
is a DS containing record for each var with their attributes. Should be designed to allow compiler to find record quickly as well to store and retrieve the data quickly.
The attributes may provide info abt storage allocated for var for types & scopes and in case of procedure names, no.

arguments and its types is also stored.



Why 2 phases?