# Vivekanand Education Society's Institute of Technology
## Department of Computer Engineering

**Subject: -SPCC**

**Class:- T.E. (D12)** **Semester:- VI** **Div:- A**

| | | | |
|---|---|---|---|
| Roll No: 21 | Name: Amit V. Joshi | | |
| Exp. No: | Title: Assignment 3 | | |
| DOP: | | DOS: | 20/04/2021 |
| GRADE: | | LAB OUTCOMES : | SIGNATURE: |

classmate
Date :
Page :

# SPCC Assignment-3

Q1) i) In frwd referencing, var/label is referenced before it is declared.

ii) Diff problems can be resolved using one pass/Two pass frwd refrencing.

iii) In one pass fwd refrencing, source prog is translated instruction by instruction. Assembler leave address space for label when it is refrenced & when assembler found the declaration of label, it uses back patching.

iv) Two pass frwd ref consist of two passes. During 1st pass symbol table opcode table & label table are maintained.

v) In opcode table, inst size & addr is stored. when label declaration is found. then its location is also stored in the label table

vi) During 2nd pass, translation from source lang to machine lang takes place. Instruction addr & label addr are used from symbol table instead of their names.

vii) Compiler doesn't know where prog will be executed in memory so compiler generated logical addr instead of absolute addr.

viii) Loader also uses Relocn const to solve problem of relocn

ix) External ref prob is resolved by linker during compilation

x) linker connects obj prog to the code for std. lib functions.

Q2)

| | | | |
|---|---|---|---|
| 200) | +04 | 1 | 211 |
| 201) | +05 | 1 | 217 |
| 202) | +04 | 1 | 217 |
| 203) | +05 | 3 | 218 |
| 204) | +01 | 3 | 212 |
| 210) | +07 | 6 | 214 |
| 211) | +00 | 0 | 005 |
| 212) | +00 | 0 | 001 |
| 214) | +02 | 1 | 219 |

| menomonic opcode | class | mnemonic info |
|---|---|---|
| MOVE M | IS | (04,1) |
| OS | DL | R#7 |
| START. | AD. | R#11 |

| | | | |
|---|---|---|---|
| 215) | +07 | 1 | 202 |
| 216) | +00 | 0 | 000 |

OPTAB.

| symbol | addr | length |
|---|---|---|
| LOOP | 202 | 1 |
| NEXT | 214 | 1 |
| LAST | 216 | 1 |
| A | 217 | 1 |
| BACK | 202 | 1 |
| B. | 218 | 1 |

| | literal | address |
|---|---|---|
| 1 | ='5' | 200 |
| 2 | ='1' | 214 |
| 3 | ='1' | 219. |

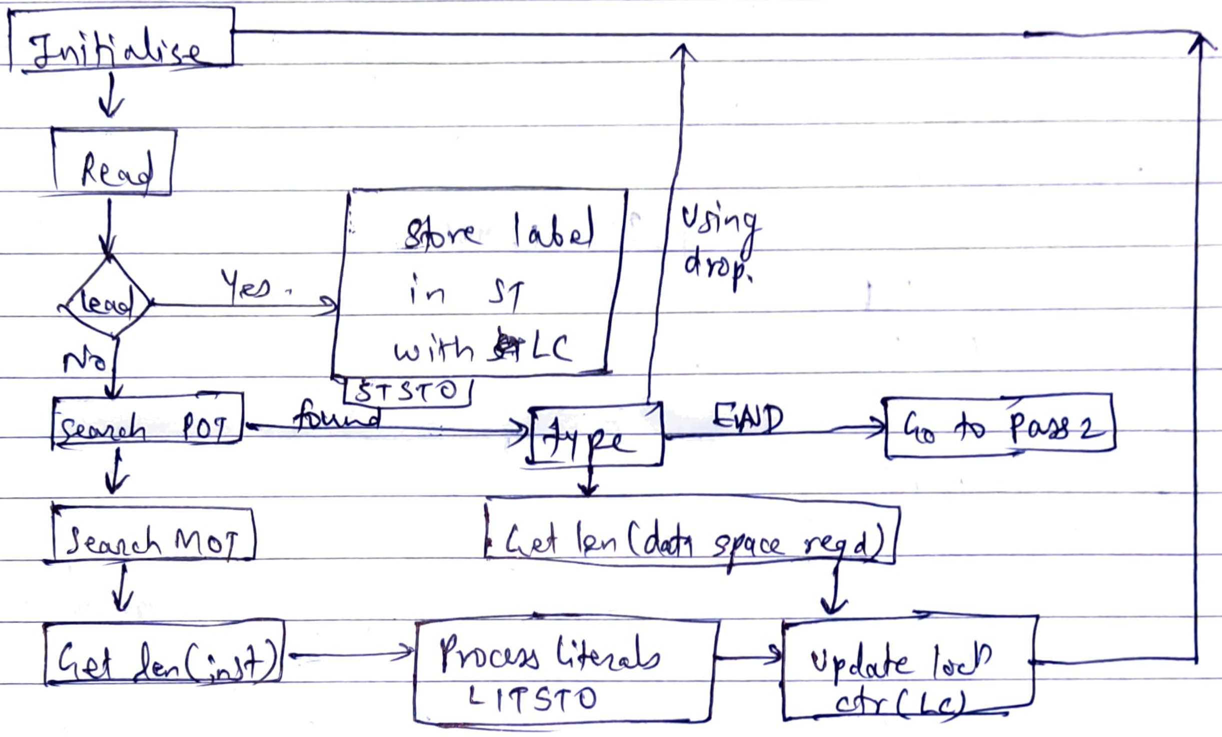
LITTAB

| literal no |
|---|
| # 1 |
| # 3 |
| - |

POOLTAB.

Q3) i) It reads entire source program & constructs symbol table of names and labels used in the program, that is, name of data fields & programs labels & their relative location (offset) within the seg.
ii) Pass 1 also determines amt of code to be generated for each inst.
iii) Pass 1 databases include:-
- Input src prg
. Locn counter
- Machine opern table (MOT)
- ~~Ps~~ Pseudo Opern table (POT)
- Symbol Table (ST)
- Literal table (LT)
- Copy of i/p to be used in pass 2

FLOWCHART:



Q4) i) Simple Macro Processor:-

In this, each macro is expanded with its code.

ii) Parameter used / Positional Macro:

It is able to insert given objects into its expansion but it is not able to insert given objects into its expansions but it is not able to modify the inst that replaces the call.

iii) Macro calls within MACROs / Nested MACROs:

When one macro expansion is taking place, we can encounter another macro call / during one macro call, other is called.

iv) Conditional Macro Expansion:

AIF & AGO are 2 conditional macro expansⁿ pseudo opcodes which permit conditional selection of sequence of the machine instr. that appears in expression of macro calls. In this, only part of macro is copied into code & will be under parameter control.

v) Recursive Macro calls:-

It allows macro Invocⁿ statement appearing within body of macro

To write recursive macro we need recursive function & term^n cond^n

vi) Macro instr defining MACROS:

Single macro is used to define grp of similar macros. To call inner macro, it is necessary to define outer macro first.

e.g.

```
MACRO
TEST            ST  .1,SWAP
L   1,F1        MEND.
A   1,F1
```

| MACRO | SUB-ROUTINE. |
|---|---|
| i) Can only be used in prog they are defined in & only after def^n. | i) Can be called from both prog & progs where they are not defined. |
| ii) Can take man 9 parameters. | ii) Subroutines can take any no of parameters. |
| iii) Macros are expanded in Compil^n / gen^n | iii) Subroutines are generated / expanded at routine. |

Q5) 3 main DAS used by Macro Preprocessor

i) DEF TAB / MDT ( macro Def^n Table) - stores macro def^n, including macro prototype & macro body.

- Comment lines are omitted
- References to the macro instructions parameters are converted to positional not^n for efficiency in substituting arguments.

ii) NAM TAB / MNT (Macro Name Table)

Stores macro names, which saves index to DEFTAB contain pointers to beg & end of def^n. It used KPT. (keyword parameter table)

iii) ARGTAB.

Used during expans^n of macro invo^n arguements are stored in this table acc to their pos^n. arg. list.

Q6) Loader is a prog that loads machine code of prog into system memory for execution. Function of loaders:-

i) Allocation:-

Used to allocate space in memory for object program. Translator can't allocate space since overlap may occour or large wastage of memory takes place.

ii) Linking:-

It combines 2/more seperate object progs & resolve symbolic ref. between object decks,

iii) Relocation:-

Modifies object prog, so that it can be loaded ~~at~~ an addr diff from locn originaly specified & adjust all dependent location addresses.

iv) Loading:

Physically it places machine instructions & data into the memory for execution,

Schemes of the loader:-

1) Absolute loader:-

Task is to avoid reasembling of all subroutines when subrouteine is changed & to perform tasks of allocation & linking for programmer

2) Relocation loader:-

Task is to avoid reasembling of all subroutines when a subroutein is changed & to perform tasks of allocation & linking of programmer.

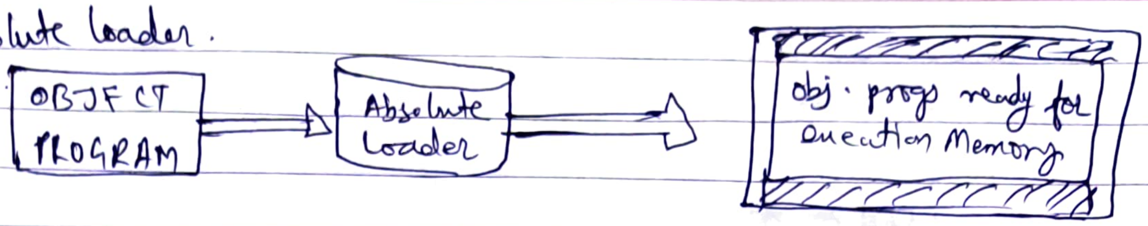
3) Dynamic loading:-

There are mainly binders capable of processing & allocating overlay str. It is also called load-an call scheme.

4) Dynamic linking:-

Loading & linking of external ref are postponed until execution time. This was made to sort disadv of prev. loading schemes like subroutines is referenced / over-executed.

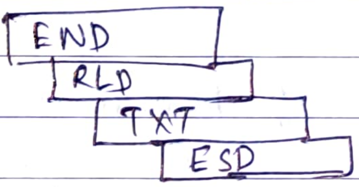
Q7). Absolute loader.



Algo :-

```
START
  read Header Record
  verify prog len & name
  read first text record
  while record type != E do
   begin
     if object code is in character form convert it into internal represe.
     move obj code to specified locn in memory.
     read next obj prog rec
     end
     Jump to addr specified in END record.
     STOP.
```

DLL Loader :- It uses 4 types of records in object file as



i) External Symbol Directory (ESD) : It contains info about all symbols that are mentioned in prog but that maybe referenced elsewhere

ii) Text Record (TXT) : It contains info about actual obj code translated version of source program.

iii) Relocation & Linkage Directory (RLD) : They are used to store these locations & address on which prog. content is dependent.

iv) END Record : Specifies end of obj file & starting addr for execution if assembled routine is in main program.