VIT | Vidyalankar Institute of Technology
Accredited A+ by NAAC
(Autonomous College Affiliated to University of Mumbai)

**Weekly Test (R-2022) -(2023-2024)**

| Date: 11/03/ 2024 | Test No: 2 | Time: 1 Hr | Branch: CMPN |
|---|---|---|---|
| Semester: VI | | Subject: SPCC | Marks: 30 |

| Q. 1) | **Attempt any five (2 Marks Each)** | CO |
|---|---|---|
| a) | Define a Macro in Pseudo Assembly Language that can perform any logical operation (as specified in call) on 2 memory locations | CO3 |
| b) | Give functions of AIF and AGO statements | CO3 |
| c) | Write a Macro in Pseudo Assembly Language to increment all locations of an array by a given value. Consider array name, array size and increment value to be passed as parameters. | CO3 |
| d) | Compare Functions and Macro | CO3 |
| e) | Give functions of both passes of 2 Pass Macro Preprocessor | CO3 |
| f) | What is the role of KPT and EVS data structure? | CO3 |
| g) | Compare Compile and Go loading scheme and General-purpose loading scheme | CO4 |
| h) | What type of module loader is needed by Linker? Why? | CO4 |
| **Q. 2)** | **Attempt any two. (10 Marks Each)** | |
| a) | Explain with a neat flowchart/ algorithm, working of 2 Pass Simple Macro Pre-processor | CO3 |
| b) | Give databases required by a Single Pass Macro Pre-processor that handles Keyword parameters, Nested Macro Calls and Expansion time loops | CO3 |
| **Q 3)** | **Attempt any two. (5 Marks Each)** | |
| a) | Explain with example, four functions of General Purpose Loader | CO4 |
| b) | Compare Absolute loader, Relocatable loader, and Direct Linking loader | CO4 |
| c) | What is dynamic loading and dynamic linking? What are their advantages over static loading and static linking? | CO4 |

| | |
|---|---|
| **CO3** | **Identify the need for different features and designing of macros** |
| **CO4** | **Distinguish different loaders and linkers and their contribution in developing efficient user applications** |

MSE 2.

Weekly Exam 2 — Solution

VIT Vidyalankar Institute of Technology
(Accredited A+ by NAAC)
(Autonomous Institute Affiliated to University of Mumbai)

~~Mid Semester Examination~~

| Branch | Date | Sem. | Roll No. / Exam Seat No. | Subject | Student's Signature | Junior Supervisor's Name and Sign |
|---|---|---|---|---|---|---|
| CMPN | 11/3/24 | VI | — | SPCC | | |

| Question No. | A | B | C | D | E | F | G | H | Total | Total out of (20 /30 / 40) |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | | | | | | | | | | |
| 2 | | | | | | | | | | |
| 3 | | | | | | | | | | |
| 4 | | | | | | | | | | |

| Examiners Signature | Student's Sign (After receiving the assessed answer sheet) |
|---|---|
| | |

1 (a)

```
MACRO
LOP   &X, &Y, &OP
LOAD  &X
 &OP  &Y
STORE &X
MEND
```

1 (b) AIF : Conditional branch instruction which is processed by Macro processor. If the given condition is true Macro processor changes current Expansion pointer to the specified label.

Format:
AIF (condition) .label

AGO : It is ~~used~~ unconditional branch to specified label during macro expansion.

Format: AGO .label

1(c)

```
MACRO
INRARR  &X, &N, &V
LCL     &I
&I SET  0                 //optional..
.AGAIN ANOP
&I SET  &I + 1
       AIF (&I = &N) .EXIT
```

Array name, size, Increment value

1 (c)

```
MACRO
INRARR  &X, &N, &V
LCL     &I
&I SET  0                 //optional
.AGAIN AIF  (&I = &N) .EXIT
       LOAD  &X + &I
       ADD   &V
       STORE &X + &I
&I     SET   &I + 1
       AGO   .AGAIN
.EXIT  MEND
```

Array Name, Array Size, Increment value.

Sample Macro call

```
INRARR  G, 3, 5
```

```
LOAD  G+0
ADD   5
STORE G+0
LOAD  G+1
ADD   5
STORE G+1
LOAD  G+2
ADD   5
STORE G+2
```

1 (d) (i) Functions are linked post translation whereas Macros are expanded before translation

(ii) In Functions, control is transfered at run time, whereas, in macros, code substitution increases size of code but no control transfer occurs at run time.

1 (e) Pass 1 : Define Macro in MNT & MDT
Pass 2 : Expand Macro Calls.

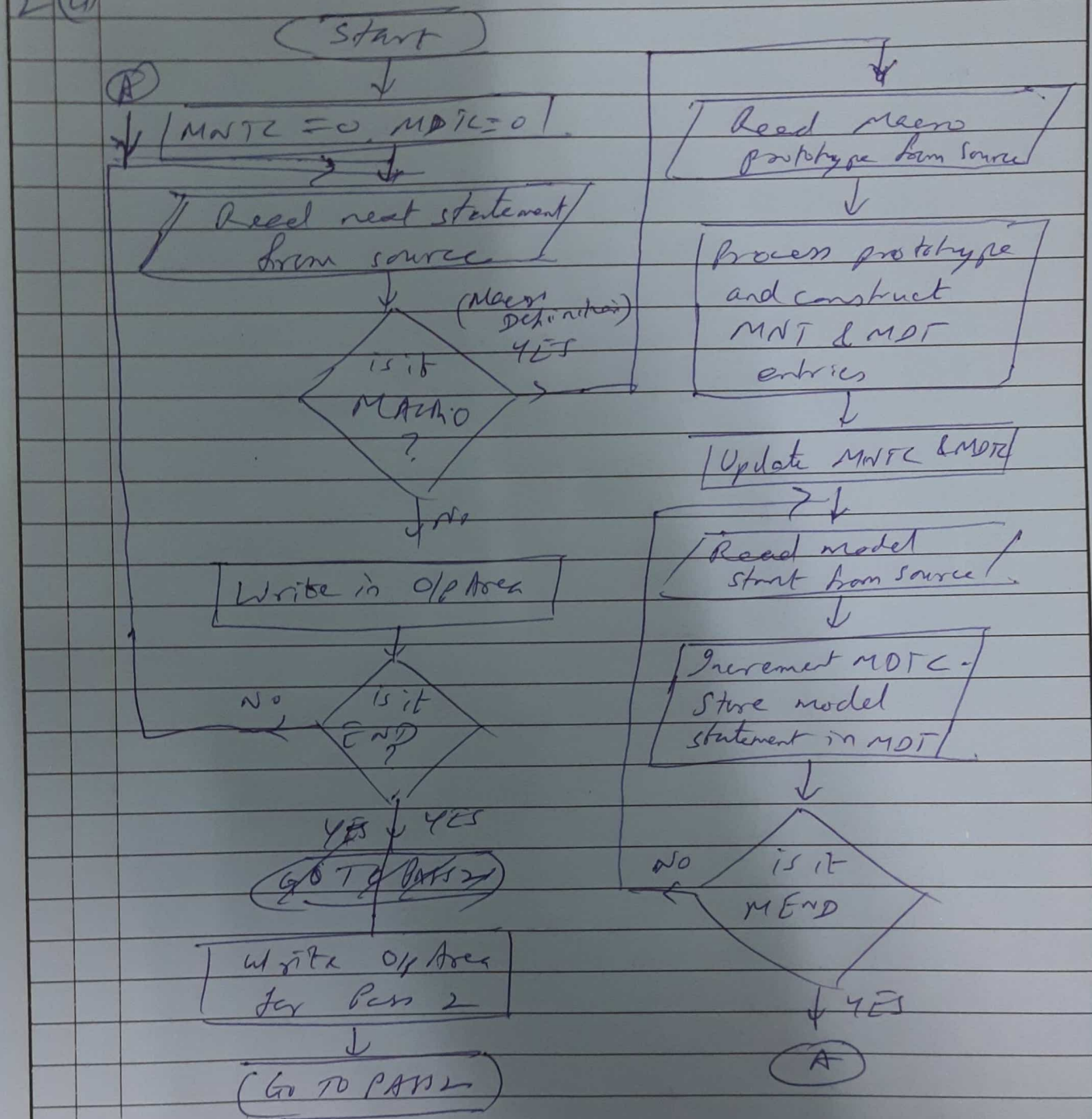
1 (f) KPT : Used to store names of keyword parameters & their default.
EVS : Used to store values of Expansion time variables during macro expansion.

1 (g) (i) Compile & Go loading scheme requires retranslation everytime the program is to be executed. This is not needed in general purpose loaders.

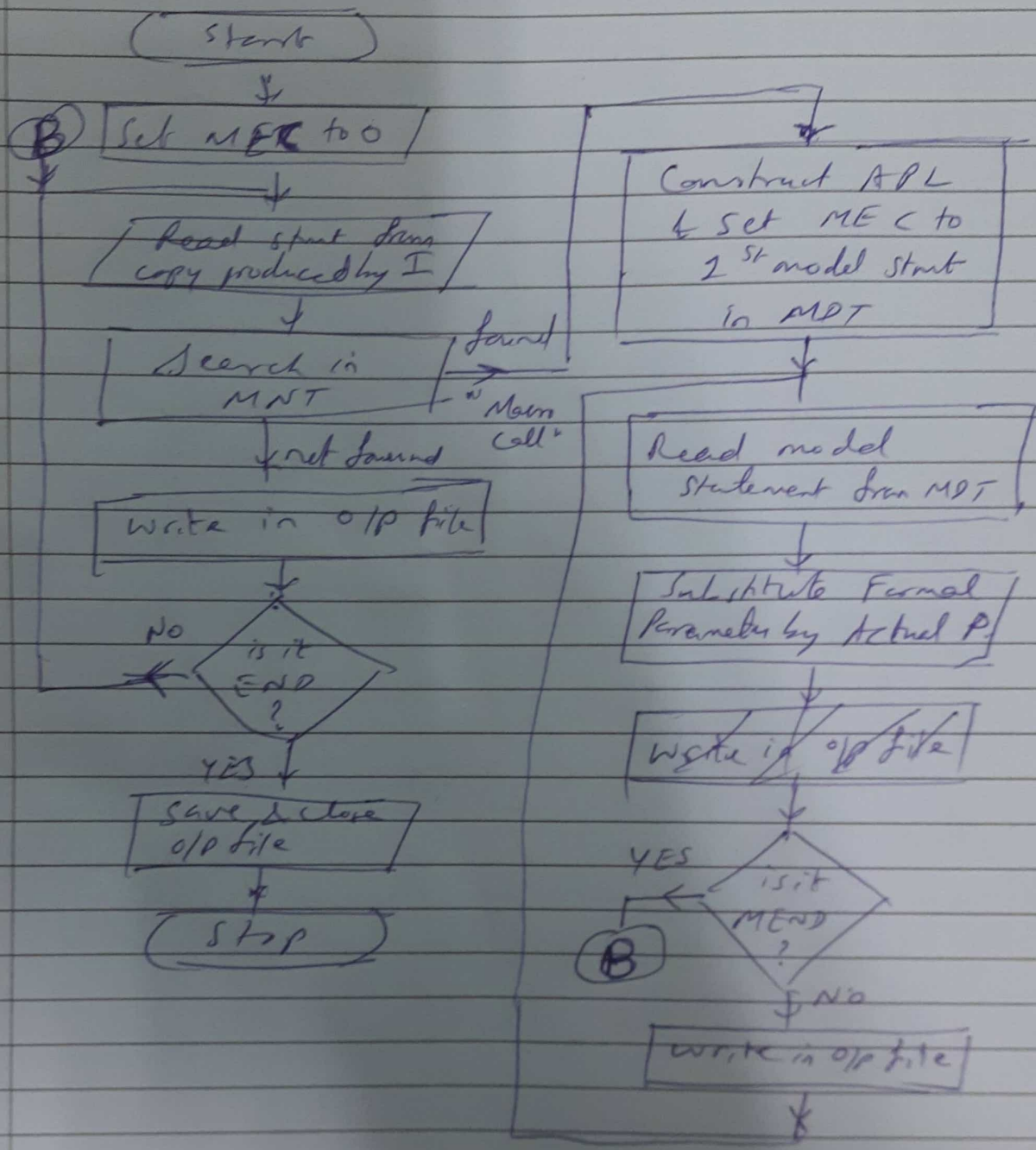
(ii) Compile & Go loading scheme requires compiler (translator) to reside in memory when user program is being loaded. This causes less memory available for user program. This problem is overcome by general purpose loader.

1 (h) Linker produces an executable file which can be loaded & executed at different memory locations everytime. Hence, Linker needs Relocatable loader which Allocates, Relocates & Loads

2 (a)




PASS 1 of 2-Pass Macro Processor.




Pass 2 of 2-Pass Macro Processor.

Explanation:

Pass 1 of 2-pass Macro processor performs Macro Definition phase. It reads line from I/P source file that contains Macro definitions & Macro calls and it looks for keyword "MACRO"

On encountering keyword "MACRO", Pass 1 constructs MNT & MDT entries wherein information of the Macro is stored.

If Macro keyword is not found in the source then it is written for Pass 2.

Hence, Pass 1 constructs MNT & MDT with all Macros defined in I/p file and also generates a copy of source for Pass 2 which contains assembly program with Macro calls but no Macro definitions.

Pass 2 reads statements from copy produced by Pass 1 and checks for Macro calls. If found, it calls for macro expansion wherein it reads statements from MDT replaces formal parameters by actual parameters and writes them to O/p file. If Macro call is not found then statements are written in O/p file till END of ALP is encountered.
This way Pass 2 will produce an O/p file which has no Macro definitions and Macro calls are expanded, in the place of call, by their corresponding generated statements.

This O/p file of Pass 2 is send to Translator (Compiler/Assembler) for translation to object file.

2 b) Macro Name Table (MNT)

| Macro Name | MDTP | No of PP | No of KP | KPTP | No of EVs |
|---|---|---|---|---|---|
| | | | | | |
| | | | | | |
| | | | | | |

Keyword Parameter Table (KPT)

| Keyword | Default |
|---|---|
| | |
| | |

MDT

| |
|---|
| (Macro Prototype) |
| : |
| MEND |

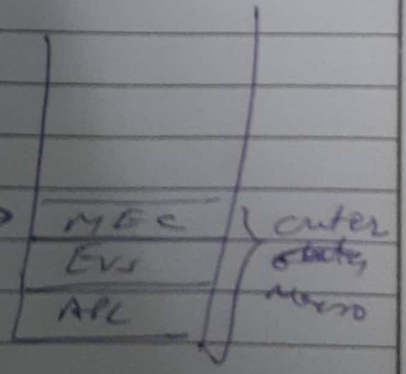
APL

| Formal Parameter | Actual Parameter |
|---|---|
| | |
| | |

EVS

| Value of EV |
|---|
| |
| |

Stack



3 (a) Functions of General Purpose Loader.

(i) Allocation: To allocate space for every module of the user program to be executed.

Eg: MAIN ⇒ 2000
SQRT ⇒ 2500

(ii) Relocation: To change address sensitive statements from their assemble/translate time address to their load time address.

Eg:
MOV AX, 3000 // in MAIN
to be relocated as
MOV AX, 2300 // with MAIN allocated from 2000.

(iii) Linking: To provide address location of all extend references (reference to symbols defined in other module)

Eg:
CALL SQRT
to be linked as
CALL 2500

(iv) Loading: To move bytes from secondary storage to the memory space allocated for the modules of the user program and make it start execution.

3 (b)

| | Absolute loader | Relocatable Loader | Direct Linking Loader |
|---|---|---|---|
| (i) | Loader only performs Loading | Loader performs Allocation Relocation Loading (May perform partial Linking) | Loader performs Allocation Relocation Linking Loading |
| (ii) | User performs Allocation & hence no Relocation problem | Relocation bits are used to tag address sensitive statements | Relocation entries state which statements to be relocated & w.r.t which module. |
| (iii) | Module are linked by User | Calls can be linked by loader using Transfer Vector No data linking | Calls & Data are linked by loader |
| (iv) | Occupies least space | occupies more space than Absolute loader | Occupies large memory space |
| (v) | Fastest: Only transfer of bytes required. | Moderate: Needs to relocate the program. | Slowest: Since it requires 2 scans of program |

3(c) Dynamic loading involves loading of modules as per requirement. This is supported by O.S & underlying hardware as Virtual memory. In this, when any virtual address (page or segment) referred is found to be not already loaded in memory then a fault occurs (page fault or segment fault). This fault causes call to Memory management service of O.S. and hence the called module is loaded in memory (i.e allocated space).

Adv:
(1) Program can be large and still execute on system with less physical memory.
(2) More degree of multiprocessing & concurrency.
(3) Only called modules are allocated space & hence no wastage of memory space.

Dynamic Linking involves linking calls and data of one modules with address of other module which is not known at compile time. Hence, translator considers these external references as call to linker. When these statements are to be executed the linker is called to identify the module to which it has to be linked. Such features are available from PL1 as function overloading or Polymorphism.

Dynamic linking requires dynamic loading but vice versa is not true.

Adv:

1. There could be multiple modules with different body but same name. The correct module to be linked can be determined at run time.

2. When project is very large and involves many modules and reusable codes, it could be an overhead to link all modules as they may never be called. Hence, dynamic linking saves time by linking only those modules which are called.