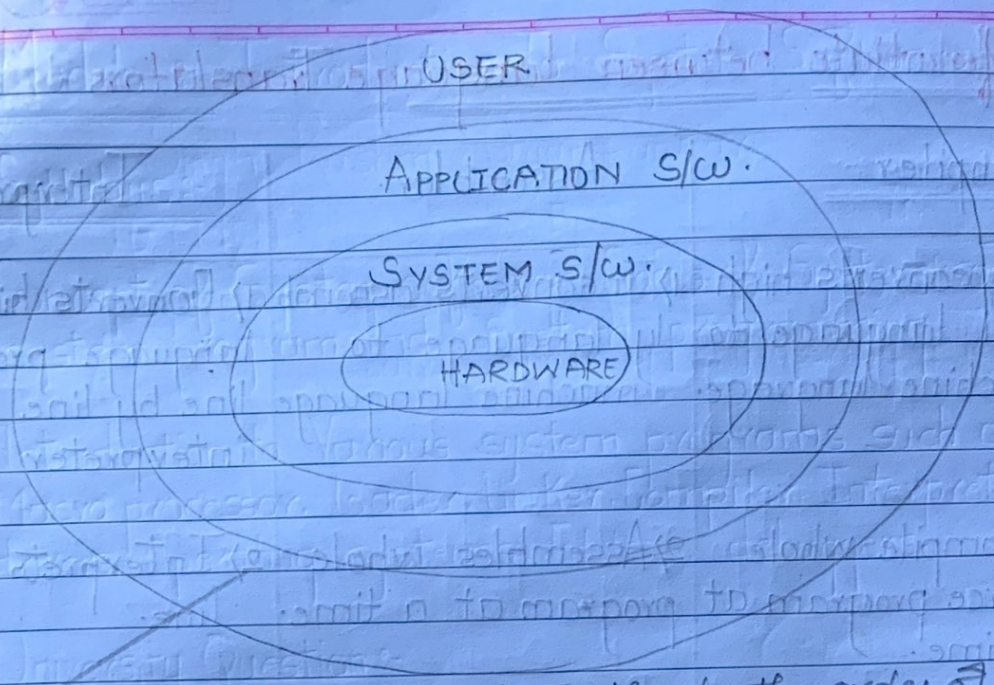
2 mark questions

Q. Differentiate between System and Application softwa

System software.	Application softwar
1) System s/w is a type of computer program that is designed to run a computers h/w and application program.	1) Application s/w are program which perform specific task for the use
2) System s/w is written in a low level language i.e assembly language. (8) because require less space.	2) It is written in high level language C, C++, Ja (8) more space.
3) System s/w are installed on the computer when OS is installed.	3) Application s/w are installed according to user requirement.
4) A computer system is unable to run without system software.	4) Application software are user specific.
5) In general, the user does not interact with, system s/w, because system s/w work in the background.	5) User's are supposed to interact with the applicati program.
6) System s/w are general purpose.	6) It is specific purpose.
7) Assembler, Macroprocessor, compiler, OS, DD, linker & loader, Editor, preprocessor.	7) MS office package, graphic package, hospital managemer



System Softwares :- Q. Indicate the order of system s/w which are used for program development to execution.

1) Compiler
2) Assembler
3) Interpreter
4) Loader.
5) Linker.
6) OS.
7) Device Driver.

8) There are 3 Language Translators :-

1) Compiler.
2) Assembler.
3) Interpreter.

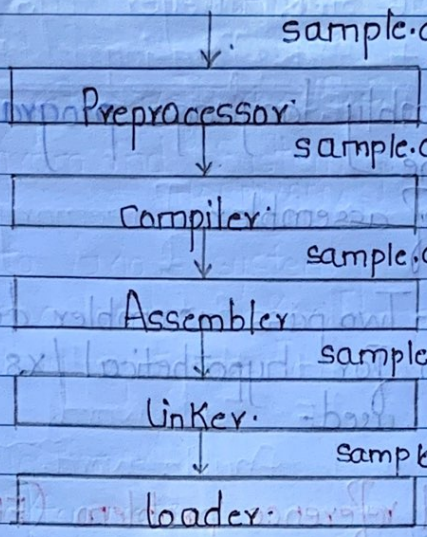
2 marks

Q. Differentiate between language Translators.

Compiler	Assembler	Interpreter
1) It converts high level language to machine language.	1) It converts assembly language to machine language	1) Converts high level language program line by line in interpreter form.
2) Compile whole source program at a time.	2) Assembles whole program at a time.	2) Interprets line by line.
3) Generate object file.	3) Generate object file.	3) Does not generate object file.
4) It requires large memory space to store the program	4) It requires large memory space to store program but less than compiler	4) It requires less memory space to store
5) Speed of execution is fast	5) Speed of execution is fast	5) Speed of execution is slow.
6) Debugging is difficult.	6) Debugging is difficult.	6) It is simple to debug.
7) Source program C, C++ etc.	7) Source program assembly language	7) Source program PHP, python, list
8) Eg: Turbo C.	8) Eg: Tasm, Masm.	8) Eg: Basic Interpreter
9) Program compiled once & executed many times.	9) Program compiled once & executed many times.	9) Every time, the program has to be interpreted

Q. Indicate the order

* Imp Indicate the order of system program from developing to execution stage.



Q. Macroprocessor, Editor, compiler, loader, linker, Assembler

- Editor
- Macroprocessor (preprocessor),
- compiler
- Assembler
- Linker
- Loader

1) Specify the problem (FPR)
2) write different data structures & their formats
3) specify algorithm.
4) Draw flowchart.
5) explain database with the help of example.

2) Symbolic operand specification

Label Mnemonic operand.
Imp.

3) Storage specification.

Five DC F '4'
(full)
(declarative constant)

Four DS H
(data storage)

* Types of statements:-

1) Imperative statements.
eg: all instruction executable.

2) Declarative statement.
eg: FIVE DC F '4' & executable.

3) Assembler directive statement.
START, END, USING, DROP

* There are 2 Types of program.

Relocatable program.	Absolute progam
PROG START ≡ END.	PROG START 100 ≡ END.
Addresses are not fixed & programs are stored anywhere in the memory.	Addresses are fixed & progra are stored in fixed memor location.

11/01/19. Friday

Q1: what is forward reference problem & how it is solved OR whether it is multi-pass assembler in IBM

Q2: how FRP solved in Intel 86.

Q3: database format of 2-pass assembler.

Module 2:- Assemblers. 9hours.

Syllabus:-

Elements of assembly language programming.
Assembly scheme
pass structure of assembler.

Assembler design:- Two pass assembler design & single pass assembler design for hypothetical /X86 family processor, data structures used.

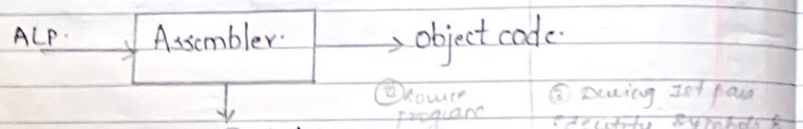
Q1. What is forward reference problem (FRP) & how it is solved in IBM 360 processor.

Q2. Design 2-pass assembler with flowchart & database.

Q3. Design single pass assembler for Intel X86 processor.

Q: difference between single pass & 2 pass.

* Assembler.



1) Features of ALP

A → Add.
S → subtract
ST → start.

Design of 2-pass assembler for IBM 360/370 processor:-
1) specify the problem (FPR)
2) write different data structures & their formats
3) specify algorithm.
4) Draw flowchart.
5) explain database with the help of example.

2) Symbolic operand specification

label Mnemonic operand.
Imp.

3) Storage specification.

Five DC F '4'
(declarative constant)

Four DS H
(data storage)

* Types of statements:-

1) Imperative statements.
eg: all instruction executable.

2) Declarative statement.
eg: FIVE DC F '4' & executable.

3) Assembler directive statement.
START, END, USING, DROP

* There are 2 Types of program.

Relocatable program	Absolute program
PROG START ≡ END.	PROG START 100 ≡ END.
Addresses are not fixed & programs are stored anywhere in the memory.	Addresses are fixed & program are stored in fixed memory location.

14/01/19
Monday

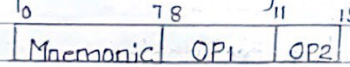
Design of 2 pass Assembler for IBM 360/370.

Registers → 16 GPR/integer regs (R0-R15) - 32 bits
4 FPR/Floating (F0-F3) - 64 bits.
1 PSW 64 bits.

Memory/DataType →	units of memory	Bytes	Bits
	Byte.	1	8
H	Half word.	2	16
F	Full word	4	32
	Double word.	8	64

RR instruction format:-

Both operands are register.

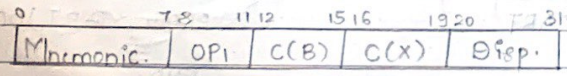


$2^4 = 16$.

RX Instruction format:-

one operator → Register
one operator → memory location

C(B) + C(x) + D
content of base reg. + content of index register + displacement



Imp. Q) What is the reason for the assembler to be multi-pass assembler for IBM 360?

OR

What is FRP in an assembler? How to solve this problem.

→ Forward Reference.

(i) The rule of assembly language program states that symbol should be defined anywhere in program.

(ii) It might be possible that a symbol is referred before its def^{n} called forward reference.

```
10:  A   1, a   <-- symbol   Forward reference → symbol Table.
                                                    5.
.
.
.
20:  a   DC F '5'  → symbol define.
```

FRPO : FR problem.

(i) Assembler replace symbol by its value but due to forward reference assembler does not know the address of the symbol called FRP.

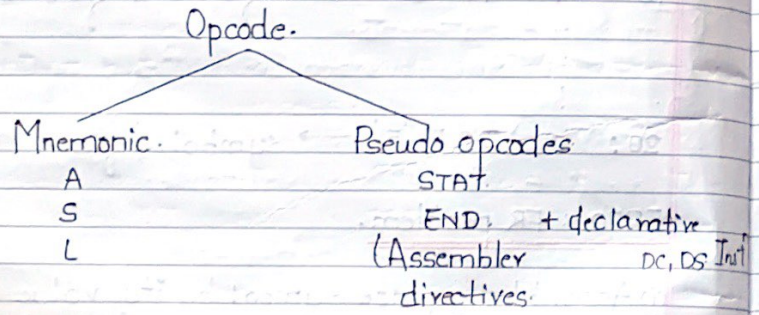
(ii) This problem is solved by making 2 passes over the i/p program.

(iii) The purpose of pass 1 is to define the symbols & literals which occur in the program & purpose of pass 2 is to assemble instruction & data.

to generate obj code.

Databases of 2 pass Assembler.

1) MOT
2) POT
3) Symbol Table
4) Literal Table
5) Base Table.
6) LC.



1) MOT (Mnemonic Opcode Table)

Structure / format of MOT:-

Mnemonic opcode (4 bytes)	Binary opcode (8 bits) (Hexadecimal)	Inst^{n} length (2 bits).	Inst^{n} format (3 bits)	Not used (3 bit)
01-half word: 'Abbb'	5A	10	001	—
10-full word: 'ALRb'	1E	01	000	—

Date 15/01/19 Tuesday

(i) MOT is predefined table, i.e the contents of MOT are not changed/filled/altered during assembly process.

(ii) In pass1, MOT is consulted to obtain instruction length & to update location counter.

(iii) In pass 2, MOT is referred to

- To replace Mnemonic by binary opcode.
- Update location counter.
- Assemble the instruction.

2) POT (Psedo Opcode Table)

- predefined table.
- contents - not filled or altered during assembly process.
- All assembler directives & declarative statements are stored in POT. eg:- START, END, DC, DS.

8 bytes/entry.

Psedo-opcode (5 Bytes)	Physical address of Psedo-opcode. (3 bytes)
START.	P1 routine address ← 24bit addressing
DC bbb	P2 routine address.

start
end
using
Drop
DC
DS
EQU

Relocatable ... If value present then Absolute]

while giving name to a program, the no of characters should be less than 11.

(i) During pass1, POT is used to process psedo opcodes such as START (start of program), END (end of program), DC (declare constants), DS (declare storage) etc.

(ii) During pass2, POT is used to process assembler directives such as using & DROP.

3) Symbol Table.

(i) Symbol Table is used to keep track on the symbols which are defined in the program. (Generally symbols are present in the label field).

(ii) During pass1, symbol table is used to make entry of a particular symbol in the symbol table.

(iii) In pass 2, symbol table is used to generate the address of the symbol.

Format of ST:-

Symbol 8 bytes	value (4 bytes)	length (1 Byte)	R/A (1 Byte)
PRG1	0	1	R
a	4	4	R

LC	label :	Mnemonic	op1	op2
0	PRG1	START		
0	—	A	1	4
1				
4	a :	ST	1	b

4) Literal Table.

(i) All the literals (i.e constants) are stored in literal table.

(ii) Generally literals are part of operand.

(iii) Format is same as symbol table.

5) Base Table.

(i) Base table is used to check which register available as base register.

(ii) Base table is not used during pass 1.

(iii) It is used only during pass 2.

Base Table

Reg availability (Y/N) (1 Byte)	Contents of Base reg (3 bytes)
0 - N	—
: :	—
: :	
15 - Y	*

(0 to 4 allow N, only 15 is Y so * is written)

(iv) Assembler directives is used to check, which register is used as a base register in the program. eg: using *, 15.

(v) DROP assembler directive to tell the assembler that particular reg is not used as a base register.

(vi) During pass 1 & pass 2, MOT is referred to find out the length of the instruction & to update the location of pointer.

Q. Write Sample ALP & also explain the o/p of each pass of 2 pass assembler.

label:	Mnemonic	OP1	OP2
PRG1	START		
	USING	*	15 → R15 available base register
	Load L	1	FIVE
	Add A	1	FOUR
	Store ST	1	TEMP
FOUR	DC	F	'4'
FIVE	DC	F	'5'
TEMP	DS	F	'1'
	END		

Location counter: 0, 4, 8, 12, 16, 20 (S-substract, Declare constant, Declare storage)

* default - increment by 1 :
  - DB B = 1 byte 8 bit
  - DW H = 2 byte 16
  - F = 4 byte 32
  - D = 8 byte 64

Date: 16/01/19 Wednesday

RX - one is in register and operand is in memory location / variable

RR - same register.

Databases :-

1) MOT :-

Mnemonic opcode (4 bytes)	Binary opcode (1 Byte) Hex.	Instruction length (2 bits)	Instruction format (3 bits)	Not used (3 bits)
Lbbb	58	10	001	
Abbb	5A	10	001	
STbb	50	10	001	

2) POT :-

Psedo-opcode (5 bytes)	Address (3 bytes)
START	P1 routine Address
USING	P1 routine
DC	
DS	
END	

3) Symbol :-

Symbol	Value	length	R/A (Relational/Absolute)
PRG	0	01 (for start only 1 byte is needed)	R
FOUR	12	04	R
FIVE	16	04 (because F is present)	R
TEMP	20	04	R

4) Literal Table.

Literal	Value	length	R/A
F '4'	12	04	R
F '5'	16	04	R
F '1'	20	04	R

5) Base Table.

Reg available (Y/N)	value.
0-N	
:	-
:	-
15-Y	*

Program.

				o/p pass1 LC	Mnemonic	op1	op2	o/p pass2
PRG1	START	0		0				0
	USING	*, 15		0				0
	L	1	, FNE	0	L	1	—	0 L 1,16 (0,15)
	A	1	, FOUR	4	A	1	—	4 A 1,16(0,15)
	ST	1	, TEMP	8	ST	1	—	ST 1,20(0,15)
FOUR	DC	F	'4'	12				
FIVE	DC	F	'5'	16				
TEMP	DS	F	'1'	20				
	END.							

EQU - equate (It won't increment)

Q.

Program:-

			Pass 1 LC	Mnemonic	Pass 2 LC	Mnemonic
PRG	START		0		0	
	USING	*, Base	0		0	
	L	Acc, Four	0	L-,-	0	L 1,12(0,15)
	A	Acc, Five	4	A-,-	4	A 1,16(0,15)
	ST	Acc, Temp	8	ST-,-	8	ST 1,20(0,15)
Four	DC	F '4'	12		12	
Five	DC	F '5'	16		16	
Base	EQU	15	20		20	
ACC	EQU	1	20		20	
Temp	DS	1 F	20		20	
END			24		24	

Dec 17/10/11 Thursday

Explain both pass-1 & pass-2.

Q. Explain 2-pass assembler with the help of flow-chart & databases

PASS I ASSEMBLER.

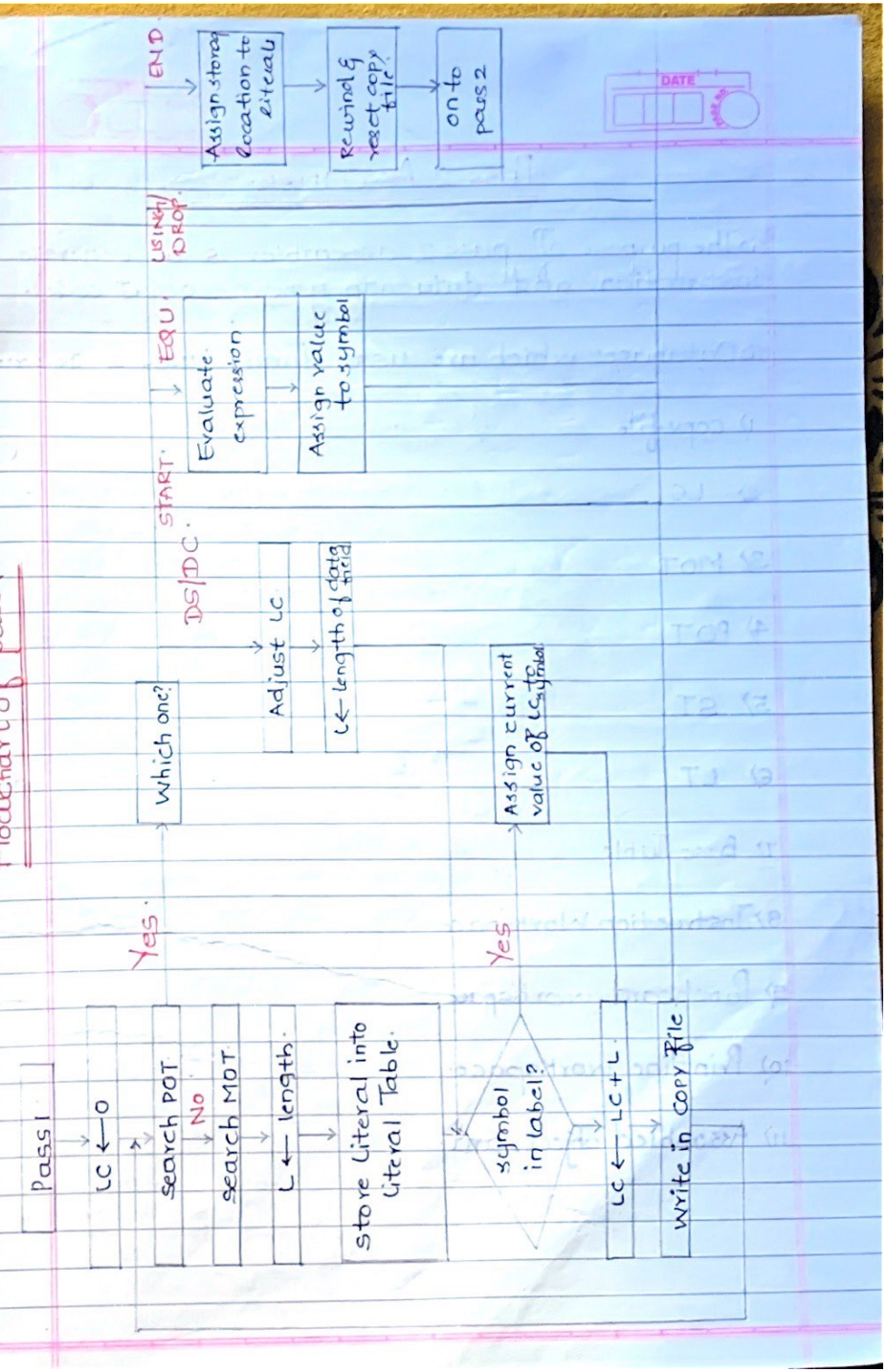
Database of pass I assembler.

1) Original source program. assembly code is the I/P of the program
2) LC To keep the Track of instⁿ & data.
3) MOT It is consulted to find length of the instruction & update location counter.
4) POT To process pseudo opcodes such as DC, DS, EQU, start and end.
5) ST To store symbol & its values.
6) LT To store literals & its values.
7) copy file. Input to pass 2.

(i) The purpose of pass I assembler is to define symbols and literals.

(ii) Various databases used during pass I assembler.

Flowchart of pass I.



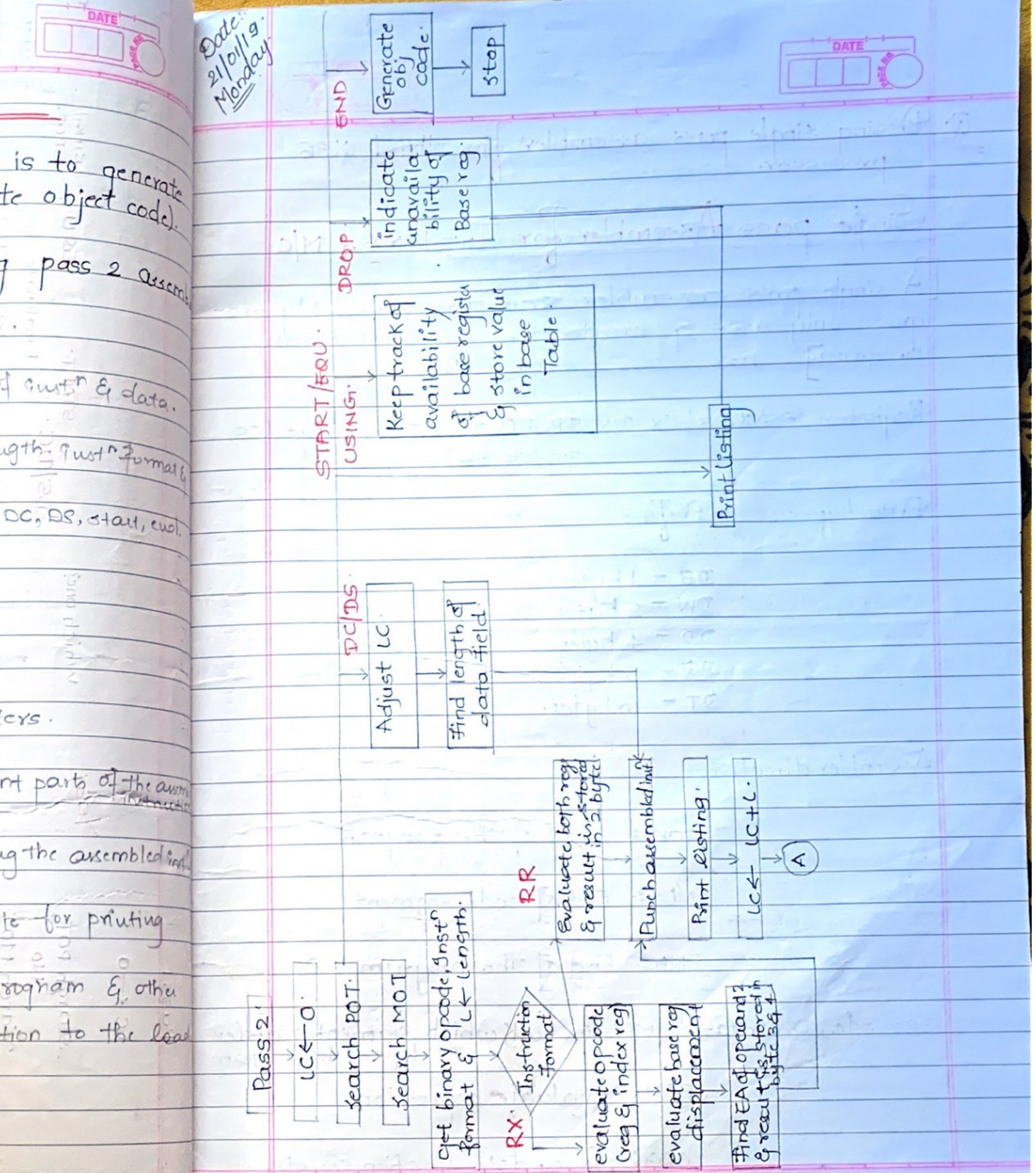
Pass 2 Assembler.

(i) The purpose of pass 2 assembler is to generate instruction and data (to generate object code).

(ii) Databases which are used during pass 2 Assembler

1) copy file prepared by pass 1 assembler.
2) LC To update the LC. To Keep Track of instⁿ & data.
3) MOT is consulted to obtain inst length, inst format & binary opcode.
4) POT To process assembler directives such as DC, DS, start, end, using and drop.
5) ST Replace symbol by its address.
6) LT Replace literals by its address.
7) Base Table To keep track of base registers.
8) Instruction Workspace: To store different parts of the assembled instruction
9) Punchcard workspace: used for punching the assembled inst
10) Printline workspace. To prepare obj file for printing
11) Assembled object card. It store obj program & other required information to the loader

Date: 21/10/19 Monday



Q. Desing single pass assembler for intel X86 processor.

Single pass Assembler For Intel X86 M/c

[A single pass assembler scan the assembly program only once & create the equivalent binary program].

Registers :- AX, BX, CX, DX, SP, BP, SI, DI, seg regs (CS, DS, SS, ES)

Data types :- Define

DB - 1 byte
DW - 2 bytes
DD - 4 bytes
DQ - 8 bytes
DT - 10 bytes.

Assembler directive:

Start :- start of the program

ENDS :- End of code segment

END :- End of the program

CS: code, DS: data ASSUME :- To check which segment register are defined in program

EQU :- Evaluate expression

ORG :- To update location counter.

Database Formats

1) MOT - mnemonic Opcode table.
2) SRTAB - segment register table.
3) STSRT - store segment register table.
4) Symbol table (SYMTB)
5) FRT - Forward reference table.
6) CRT - Cross reference table.

Date: 23/10/19 Wednesday

Explain & design with the help of databases (data structure) & flowchart working of single pass assembler.

Datastructure (databases) of single pass Assembler

1) MOT :- - predefined table
- Hash organized.
- Store instructed / declaratives / assembler directives.

ADD DB START

Formats:

Mnemonic	Binary opcode	Instruction Format Info	Routine Id.
ADD	3EH	00H	R2
START			
DB			

00H → only one information format/register
FFH → All instruction format support/memory

2) SRTAB (segment register Table).

seg register	segment name.
CS	code
DS	data.

Assume : CS : code. DS : Data.

Whenever assembler encounter ASSUME statement it makes an empty in segment register table & copy the previous entry into store segment register table.

3) STSRT (store segment register Table).

seg register.	Segment name.
CS.	
DS	
SS	
ES.	

4) Symbol table :-

Symbol table is hash organized data structure which organizes all to the data. It contains all relevant information about symbol.

symbol	EQU	seg name	Define	Type	offsets in seg.

Owner seg	length size (L)	stmt #	ptr to first FRT	ptr to last

ptr to first FRT :- Whenever symbol is foward reference & entry is made in FRT, this field points to the first FRT.

Whenever a symbol reference is mode, entries mode in CRT. This field points to last CRT.

CRT :- Cross reference table.

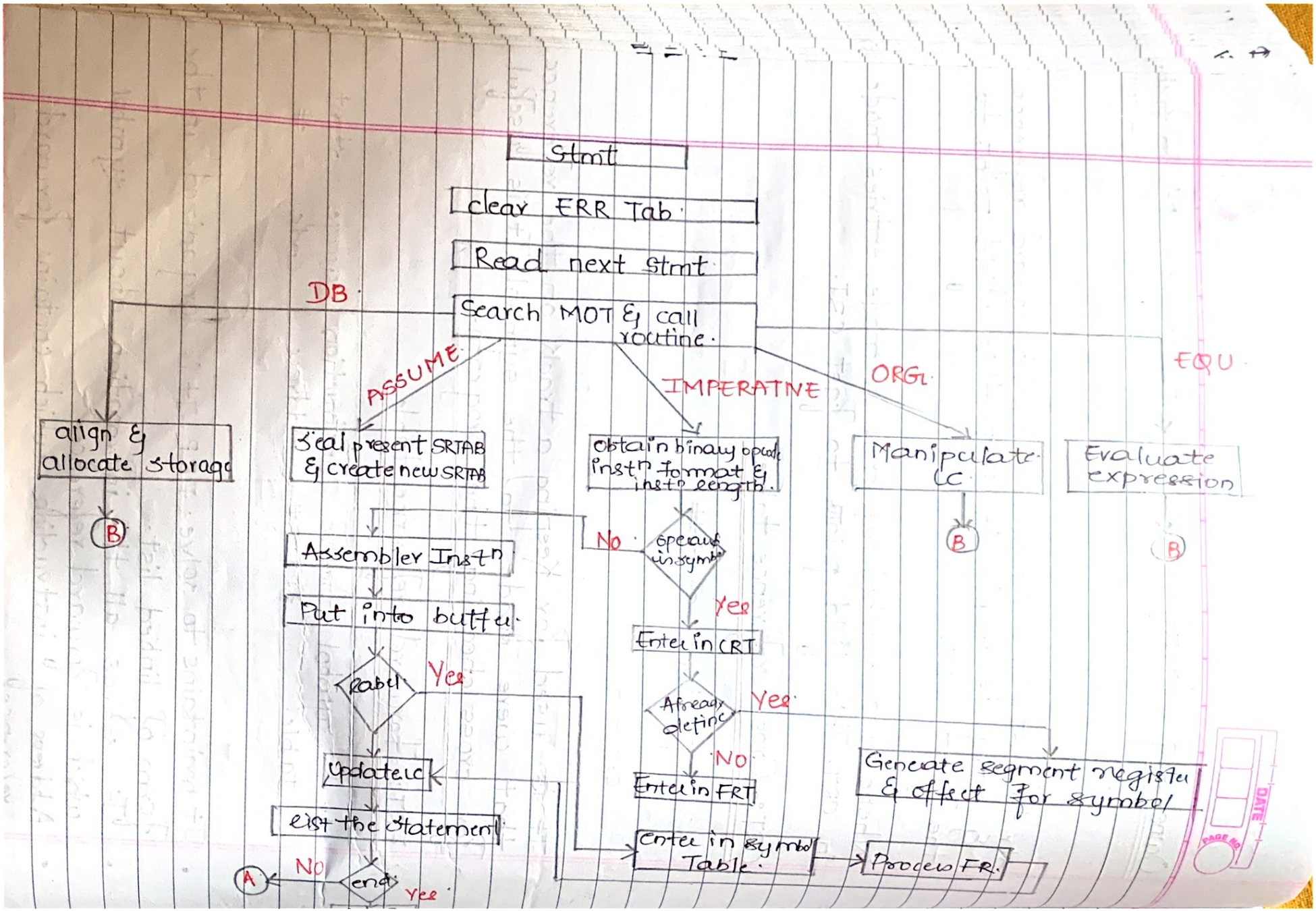
pointer	stmt ##

It is used for keeping a track on the reference that were made to the symbol. It is useful for cross checking the program.

FRT :- Forward reference table

ptr to symbol Table.	Active SRTAB#	Instruction Address	usage code.	stmt #.

- It maintains to solve : FRP. It is organized in the form of linked list.
- It stores all the information about symbol which is forward reference.
- Address of instruction which contains forward reference.


Stmt
clear ERR Tab
Read next stmt
Search MOT & call routine
DB
ASSUME
IMPERATIVE
ORG
EQU
align & allocate storage
B
Seal present SRTAB & create new SRTAB
Assembler Instn
Put into buffer
Label
Yes
Update LC
List the statement
end
No
Yes
A
obtain binary opcode instn format & instn length
No
opcode in symb
Yes
Enter in CRT
Already defined
Yes
No
Enter in FRT
Enter in symb Table
Manipulate LC
B
Evaluate expression
B
Generate segment register & offset for symbol
Process FR
DATE
PAGE NO

Dai 25/01/4 friday: Q. Difference between macros & procedure
Q. Explain macrofacility.
Q. Database formats of macroprocessor.

Module 3. Macros & Macro Processor.

(4 hrs) (10-20 Ma

Syllabus:-

Macros - definition.
Macroprocessor - defn, functions.
Difference between macros & procedure/subroutine/fu
Macro facility / features
- simple.
- parametized macro / Macro argum passing.
- Nested macro / Macro with mac
- Conditional macro

Design of single pass Macroprocessor & data structures used.

MACRO.

(i) Single line abbreviation for group of instructions.

MACRO DEFINITION :-

MACRO	—	start of macro defn. Macro Psedo-opcode.
	—	Macro name.
— — —	} —	Macro body.
MEND.	—	End of macro.

MACRO INSTRUCTION: Sequence of instructions

```
A 1, DATA 1
A 2, DATA 2
A 3, DATA 3
```

body → { — — — }

```
A 1, DATA 1
A 2, DATA 2
A 3, DATA 3
```

Define:

```
MACRO INCR
A 1, Data 1
A 2, Data 2
A 3, Data 1
MEND
```

(A 1, Data 1 / A 2, Data 2 / A 3, Data 1 : definition)

```
—
—
—
INCR
—
—
—
INCR
```

macro call

```
A 1, data 1
A 2, data 1
A 3, data 1
A 1, data 1
A 2, data
A 3, data
```

macro expansion

MACROPROCESSOR

(i) It is a system s/w which is responsible to process the macros.

(ii) There are 4 functions can be performed by macro processor

1) Recognize the macro — This can be identified through macro & MEND.
2) Save the macro definition. — Macro defn is stored in MDT.
3) Macro call. — Macroprocessor identify macrocall with the help of macro name.
4) Macro expansion. — whenever macro is call, macro replace by its definition.

Q. Differentiate between macros & procedure.

Macros.	Procedure.
1) Single line abbreviation for group of instructions.	1) When group of instructions are repeated we can use procedure.
2) Macros are faster than procedure.	2) Comparitively slower than macros.
3) More memory space is requried. Does not reduce the size of the program.	3) Require less memory space. Reduces the size of program.
4) When macro call, it replace by its definition.	4) define once call many times

5) No transfer of program counter.	5) Transferring of program counter is required.
6) It requires an overhead of macroprocessor.	6) There is no overhead of macroprocessor i.e it handle by processor itself.
7) Macro does not return a value.	7) Procedure return a value.
8) If group of instruction 3-7 i.e small code, we can use macro.	8) More than 7 line we can use procedure.
9) MACRO NAME — — — } Macro body MEND.	9) PROC NAME. — — — } group of instruction RET ENDP.
10) Stack datastructure is not used. Because there is MDT.	10) Stack datastructure is used to store the return address.

MACRO FACILITY / FEATURES.

① Simple.

Parameterized Macros / Argument Passing

Also Known as dummy parameters.

Source. ⟶ process macros.

```
—
—
—
A 1, data1
A 2, data1
A 3, data1
—
—
—
A 1, data2
A 2, data2
A 3, data2
```

```
MACRO   INCR   ARG.
        A   1,  &ARG.
        A   2,  &ARG.
        A   3,  &ARG.
MEND.
```

macro def

macro call:
INCR DATA1
INCR DATA2

macro expansion:
A 1, data1
A 2, data1
A 3, data1
A 1, data2
A 2, data2
A 3, data2

Date: 28/01/19. Monday

Nested Macro / Macro within Macro

```
L 1, Data1
A 1, 'F'.1
ST 1, Data1.
=
L 1, data2.
A 1, F 1
ST 1, data2.
```

Expanded code level 1

```
MACRO ADD1 &ARG
      L  1, &ARG
      A  1, F '1'
      ST 1, &Arg
      MEND

MACRO ADDS &Arg1, Arg2
      ADD 1 & arg1
      ADD 1 & Arg2
      MEND
```

ADDS Data1

ADDS Data2

Conditional Macro Expansion.

(i) Conditional pseudo opcode (branch) (AIF).

(ii) AGO - Unconditional pseudo opcode (Goto stmt)

```
loop1  A 1, Data 1
       A 2, Data 2
       A 3, Data 3.

Loop2. A 2, data 2.
       A 3, data 3.
       ....

Loop3  A 1, data 1
```

```
MACRO
&Ang VARY &count &arg1, &arg2, &arg3
      A 1, &arg1.
      A 2, &arg2.
      A 3, &arg3.
MEND.
```

```
.     A 1, data1.
AIF (&count EQ 1) .FINISH.
      A 2, Data 2
AIF (&count EQ 2) .FINISH.
      A 3, data 3.
AIF (&count EQ 3) .FINISH.
MEND.
```

Macro call

```
loop1 vary 3 Data1, data2, data3
loop2 vary 2 Data2, data3
loop3 vary 1, data1 — A 1, data1
```

macro expansion

A 1, data1
A 2, data2
A 3, data3

A 2, data2
A 3, data3.

Format of Databases:

1) ALA → argument list array.
2) MNT → Macro name table.
3) MDT: Macro definition table.

ALA Format (8 byte per entry)

Index	Argument
0	bbbbbbbb
1	Data1bbb
2	Data2bbb
3	Data3bbb

(i) Arguments or parameters are stored in ALA

MDT Format

Index	Defination
:	
15	vary &arg1, &arg2, &arg3
16	A 1, #1
17	A 2, #2
18	A 3, #3
19	MEND

(i) Macro definition is stored in the Macro definition table.

MNT Format

(i) Name of macro are stored in macro definition table.

Index	Macro Name	MDT Index
:		
5	Varybbbb	15

8 byte 4 Byte

Date 30/10/19 Wednesday

Q. Explain single pass Macroprocessor for nested call with the help of flowchart & databases.

→ In two pass macroprocessor there are two seprate passes. During first pass macroprocessor identify macrocalls add definition and second pass is used to expand the macro.

Databases used for single pass Macroprocessor

1) Input Macro source program

2) MDT [macro definition table] used to store the body of macro defination.

3) MNT. [Macro name Table] used to store name of the macros.

4) MDTC. (Macros defination Table counter) used to indicate next available entry in MDT.

5) MNTC (Macro Name Table counter) is used to point next available entry in MNT.

6) ALA (Argument list Array) :: It stores arguments or paramcters.

7) MDI (Macro definition input indicator).

It is a counter or switch which keeps the track of macro expansion.

if MDI = ON → macro expansion.
else MDI = OFF → otherwise.

8) MDLC (Macro definition level counter)

It keeps the track of macro call & macro definition.
MDLC is incremented by 1 when pseudo opcode occurs.

9) Copy file :- expanded macro source code is i/p to the assembler.

Index	MDT
1	Disp.
2	MOV AH,09.
3	lea DX,#0.
4.	INT 21H.
5.	MEND.
6	ACC.
7	MOV AH,01.
8	INT 21H.
9.	MEND.

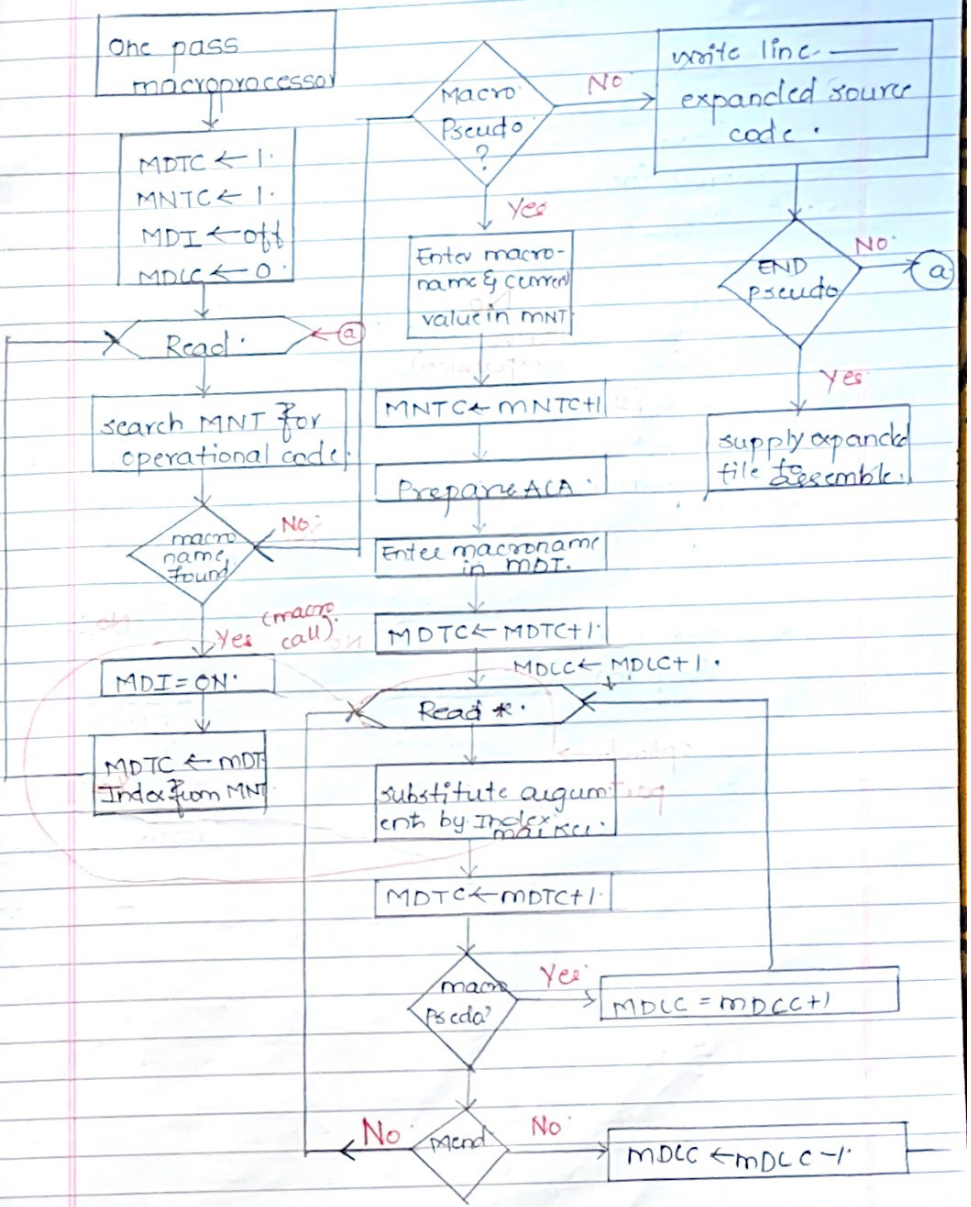
Index.	MNT.	Index MDT
MNTC → 1. → 2.	Disp	1
	Acc.	6

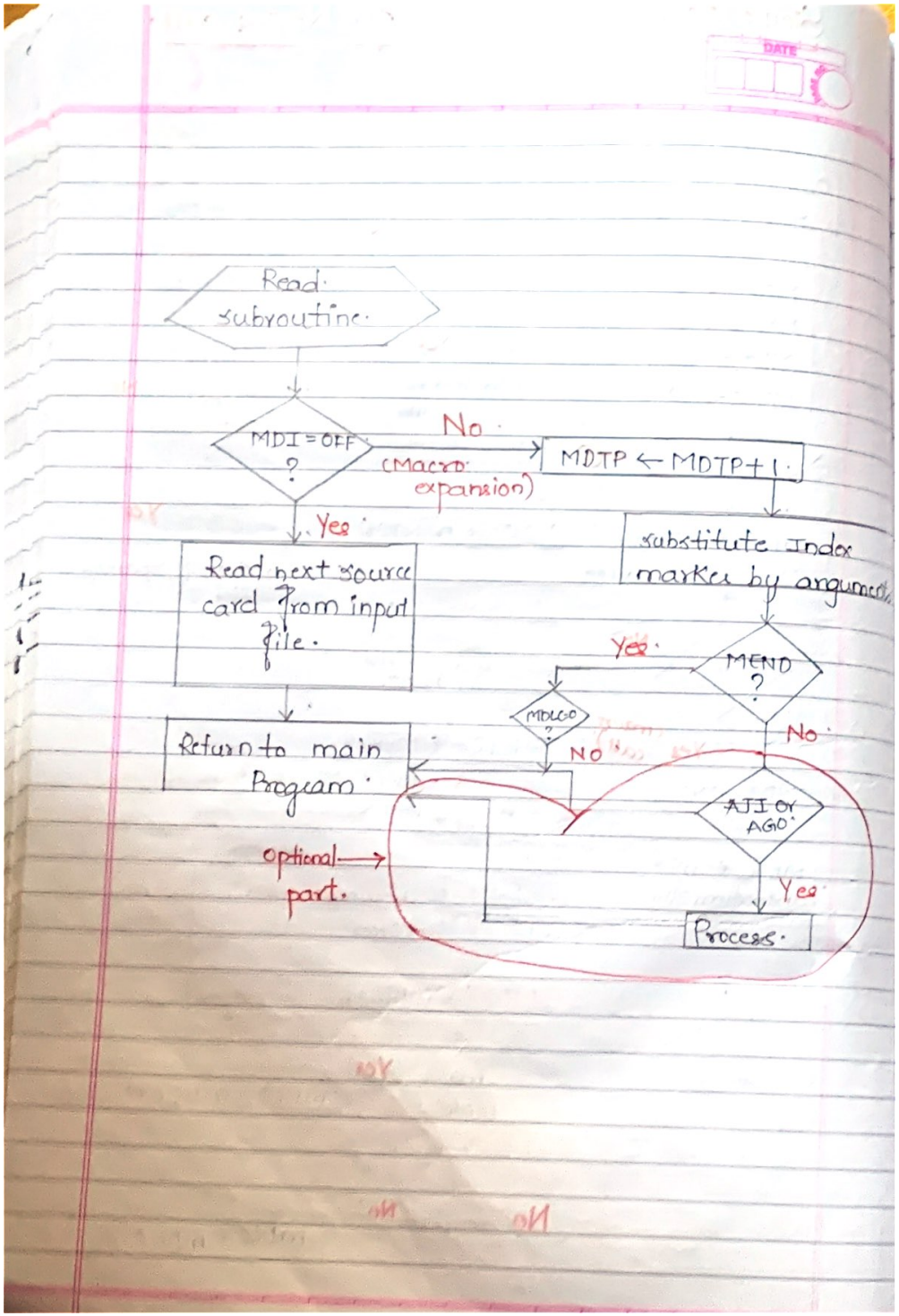
Index.	ACA.
0	XX.

model small
data
code
startup.
Disp Macro XX
Acc.
macro body: { mov AH,09. lea DX,XX. INT 21H. MEND.
ACC macro
MOV AH,01
INT 21H
endm.
Disp

* output is supplied to assembler.

Single pass Macroprocessor for Nested call.




Read subroutine.
MDI = OFF ?
No
(Macro expansion)
MDTP ← MDTP + 1.
Yes
Read next source card from input file.
substitute Index marks by arguments.
MEND ?
Yes
MDUSO ?
No
Return to main Program.
No
AJI or AGO.
optional part.
Yes
Process.