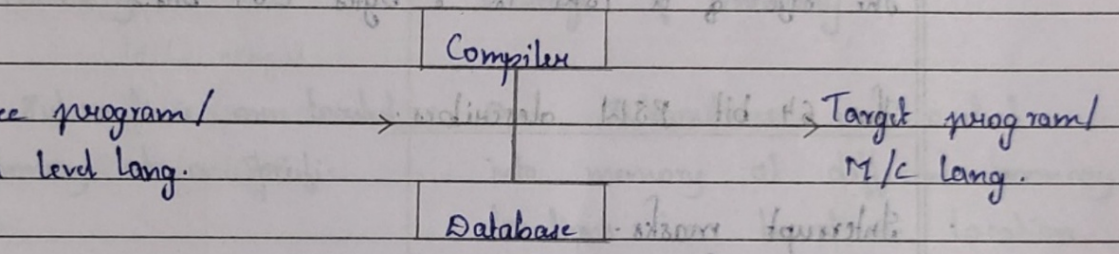
PAGE NO :
DATE : / /

SYSTEM PROGRAMMING
ASSIGNMENT.

1. Define compiler, Assembler and interpreter.

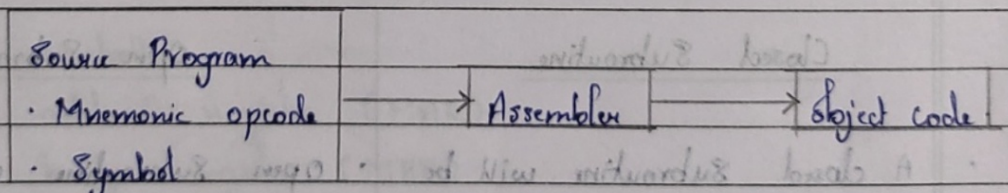
Compiler :-

It is a translator that converts High level language program to machine language. Input to compiler is source program and output is object program.



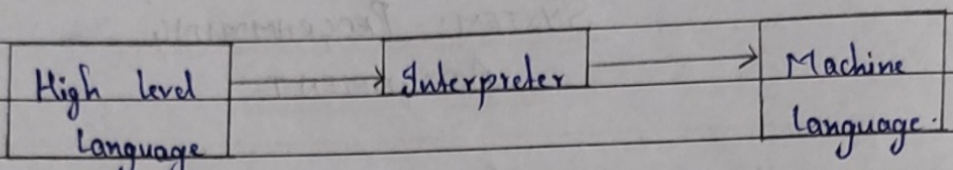
Assembler :-

Is a program that translates the assembly language program (Source code) into machine language program (Object code).



Interpreter :-

Is a computer program that directly executes instructions written in a programming or scripting language, without requiring them previously to have been compiled into a machine language program.



2. What is PSW?

Program status word contains the information required for proper execution of a given program. It contains the value of the location counter, protection information and interrupt status. The size of a PSW is 8 bytes (64 bits).

The 64 bit PSW describes.

- Interrupt masks.
- Privilege states.
- Condition code.
- Instruction address.

3. Explain Open Subroutine and closed Subroutine.

Closed Subroutine	Open Subroutine
• A closed subroutine will be stored outside the main routine and there is a transfer in control to the subroutine for processing it.	• Open subroutine or macro definition is one whose code will be inserted at the point of function call within the main definition.
• Large size macros can be	• If the macro definition is very

executed, any number of times.	Large and if you call these functions frequently shortage of memory may occur.
• There is an overhead of transferring the control to the function and returning back which takes time.	• Saves time because there is no overhead of program control transfer and return.
• Saves memory	• Wastage of memory.
• closed subroutines are loaded into memory at a specific address.	• Open subroutines are loaded into memory at different memory locations i.e based on location of calling macro.

4. Explain General machine structure with neat block diagram.

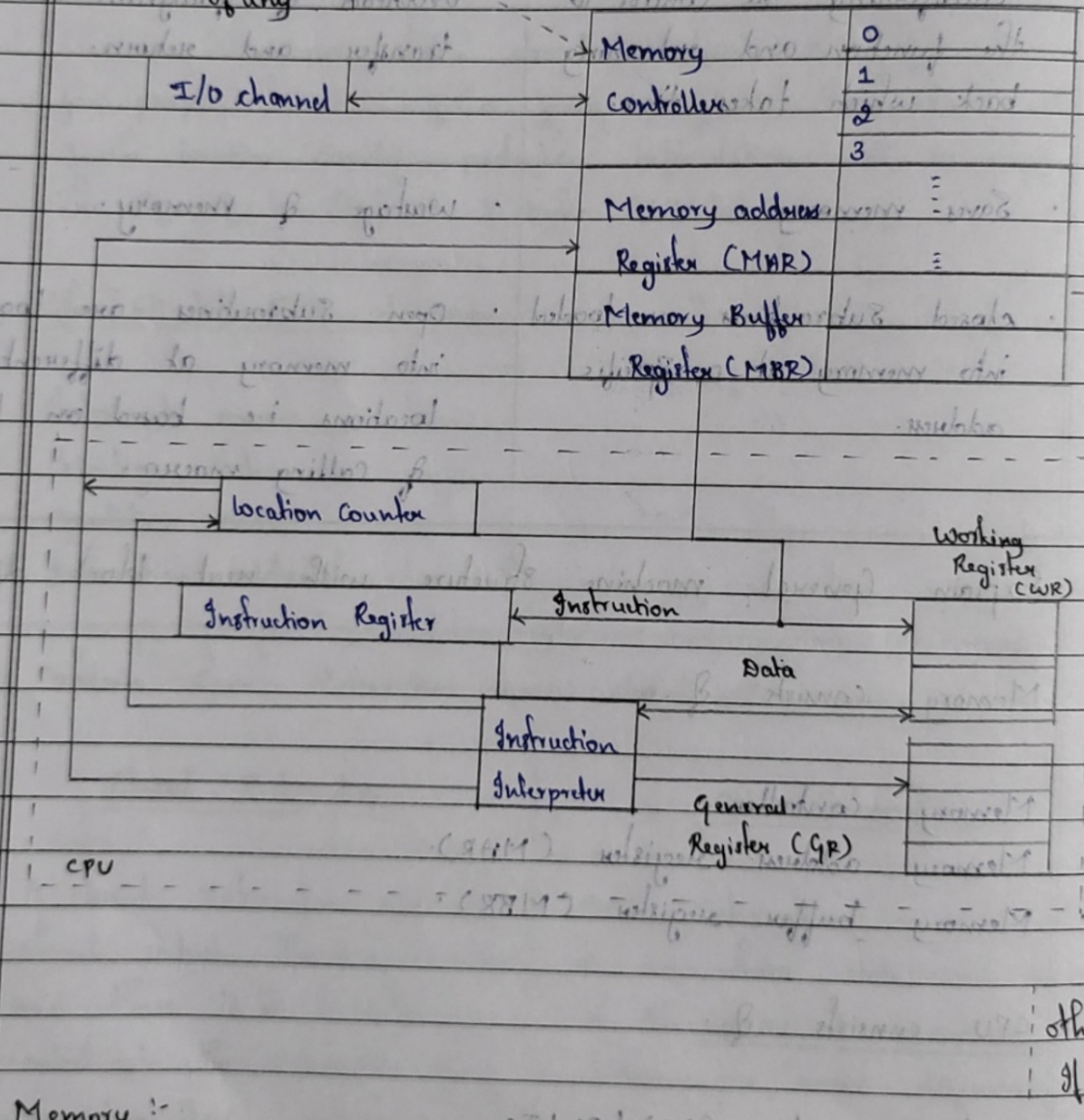
Memory consists of.

- Memory controller.
- Memory address register (MAR).
- Memory buffer register (MBR).

CPU consists of.

- location counter (LC) / PC / IC.
- Instruction register (IR)
- Instruction Interpreter (II).

Working register (WR)
General register (GR)



Memory :-

The primary interface between memory and the CPU is via the memory address register and memory buffer register.

PAGE NO :
DATE : / /

i) Memory address register (MAR)

It contains the address of memory location that is to be read from or stored into.

ii) Memory Buffer Register (MBR)

It contains a copy of the designed memory location specified by the MAR after a read operation or the new contents of the memory location prior to the write operation.

iii) Memory Controller (MC)

It is a hardware that transfers the data or the instruction between the MBR and the core memory location the address of which is present in MAR.

CPU COMPONENTS.

i) Instruction Interpreter:-

It is a group of electrical circuits (hardware) that performs the purpose of the instructions fetched from the memory. It is like a decoder that decodes the type of the instruction. i.e interprets the instructions fetched from memory.

ii) Location Counter (LC):-

It is also known as program counter (PC) or Instruction counter is a hardware memory device which holds the location of the current instructions being executed.

iii) Instruction Register (IR):-

It contains a copy of the current instruction is executed.

iv) Working Register (WR's) :-

They are memory devices that serve as 'SCRATCH PADS' for the instruction interpreter. They are basically general purpose registers.

v) General purpose registers (GPR's) :-

They are used by the programmers as storage locations and for special functions.

I/O channels :-

I/O channels can be thought of as separate computers which interprit special instructions for inputting and outputting information from memory.

5. Write Add micro flow chart.

ADD 2,176 where ADD is the operation code, 2 is the register number and 176 is the memory location. This instruction means, add the contents of general register 2 with the data stored in memory location 176 and store the result in register 2.

Step 1 :-

Move the address of the current instruction which is present in LC to MAR.

Step 2 :-

Read the instruction from memory whose address is specified in MAR and move it to MBR.

PAGE NO :
DATE : / /

Step 3 :-

put the instruction into IR.

Step 4 :-

Find the type of instruction specified in IR. If instruction type = "ADD" peform the following steps.

Step 5 :-

Move the address field in the instruction present in IR (i.e 176) to MAR.

Step 6 :-

Read the data present in IR (i.e 8) to WR.

Step 7 :-

Move the general register value present in IR (i.e 2) to WR.

Step 8 :-

Add the contents of WR and MBR, and store the result in WR.

Step 9 :-

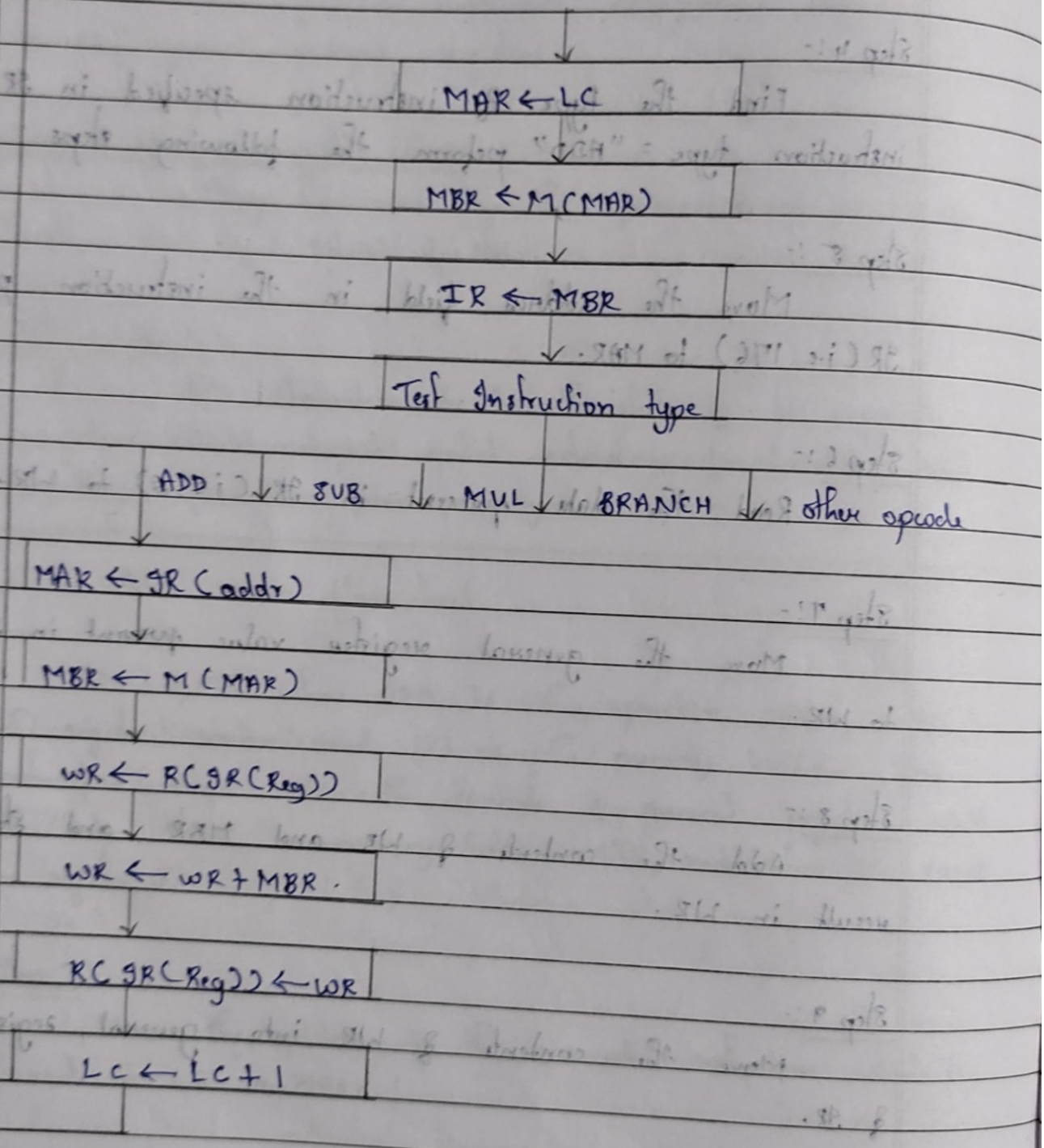
Move the contents of WR into general register field of IR.

Step 10 :-

Increment IC to the address of next instruction.

Step 11:

Repeat step 1



PAGE NO:
DATE: / /

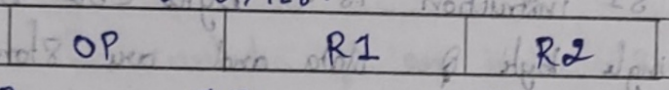
6. Explain instruction format with an example.

Different types of Instruction format are:

i) RR instruction:-

- RR instruction denotes register to register operation. That is both the operands are registers.
- The length of it is 2 bytes (16 bits)

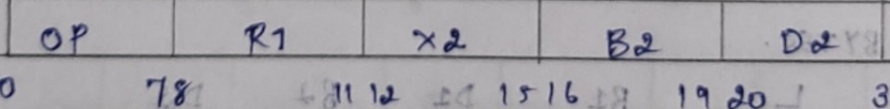
RR - 2 BYTES.



ii) Rx instruction:-

- Rx instruction denotes a register and indexed storage operation. That is one operand is a register and the another one storage operand.
- The length of Rx type instruction is 4 bytes (32 bits)
- Storage operand refers to the data stored in core memory. The address of the storage operand is calculated as follows Address = contents of base register + contents of index register + displacement = c(B2) + c(X2) + D2.

Rx - 4 BYTES.

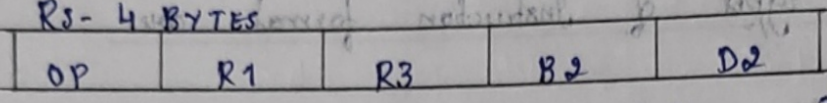


iii) Rs instruction:-

- Rs instruction denotes register and storage operation.

- The length of RS type instruction is 4 bytes (32 bits)

RS - 4 BYTES



0 7 8 11 12 15 16 19 20 31

iv) SI instruction :-

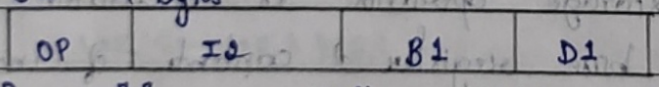
- SI instruction denotes a storage and immediate operand operation.
- The length of SI instruction is 4 bytes (32 bits). Immediate operands are single byte of data and are stored as part of the instruction.

v) SS instruction :-

- SS instruction denotes a storage to storage operation.
- The length of SS instruction is 6 bytes (48 bits)

SI instruction :

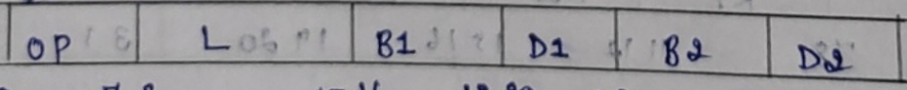
SI - 4 Bytes



0 7 8 15 16 19 20 31

SS Instruction :

SS - 6 BYTES.



0 7 8 15 16 19 20 31 32 35 36 47.

PAGE NO :
DATE : / /

7. Write a program Address modifications using index register.

Use 3 main instruction.

- Load (L) the instruction.
- ADD (A) 49, and then.
- STORE (ST) instruction.

Loop these 3 instructions, (L, A, ST).

update the contents of an index Reg by 4 during each pass by adding 4 to the address specified in load and store instructions.

Absolute add	Relative addr	Instructions	Comments.
48	0	SR 4,4	Clear Register 4.
50	2	L 2,904(4,1)	Load data element of array.
54	6	A 2,900(0,1)	Add 49.
58	10	ST 2,904(4,1)	Replace data element.
62	14	A 4,896(0,1)	Add 4 to index register.
		Branch back to relative location 2, nine times	

8. Explain data format for MOT, POT, ST, MDT, MNT.

i) MOT: Format of Machine operation code table.

Is shown as follows.

← 6 bytes entry →

Mnemonic op-code (4 byte) character	Binary op-code (1 byte) Hex	Instruction length (2 bits) binary	Instruction format (3 bits) binary	Not used in this design (3 bits)
"Abbb"	5A	10	001	-
"LOAD"	4A	10	001	-
"Movb"	5E	01	000	-
"MVIb"	IE	11	100	-

b → represents blank.

Codes used -

Instruction length

01 = 1 half words

10 = 2 half words

11 = 3 half words.

Instruction

000 - RR

001 - RX

010 - RS

011 - SI

100 - SS.

Machine Operation Table (MOT), that indicates for each instruction, a) Symbolic mnemonic b) length, c) binary machine op code and d) format (eg R, S, RX, SI).

ii) POT: Format of pseudo operation code table.

Pseudo - Operation Table (POT), that indicates for each pseudo-op the symbolic mnemonic and the action to be taken in pass 2.

PAGE NO :
DATE : / /

← 8 - byte per entry →

Pseudo - op (5 + byte) (character)	Address of routine to process pseudo - op (3 - byte = 24 bit addr)
"DROPbb"	PIDROP
"ENDbb"	PIEND
"EQUbb"	PIEQU
"START"	PISTART
"USING"	PIUSING

These are presumably labels of routines in pass 1; the table will actually contain the physical addresses.

iii) ST - Format of Symbol table.

Symbol Table (ST), prepared by pass 1, containing each label and its corresponding value.

← 14 bytes per entry →

Symbol (8 - bytes) (characters)	value (4 - bytes) (hexadecimal)	length (1-byte) (hexadecimal)	Relocation (1-byte) (character)
"JOHNbbbb"	0000	01	"R"
"FOURbbbb"	000C	04	"R"
"FIVEbbbb"	0010	04	"R"
"TEMPbbbb"	0014	04	"R"

iv) MDT : Macro definition table.

It is table of text lines. It consists of two fields, index that keep track of line numbers of the macro definition and the card that is 80 bytes of size and is responsible for storing the macro definition. Everything except the pseudo code MACRO is inserted into the Macro definition.

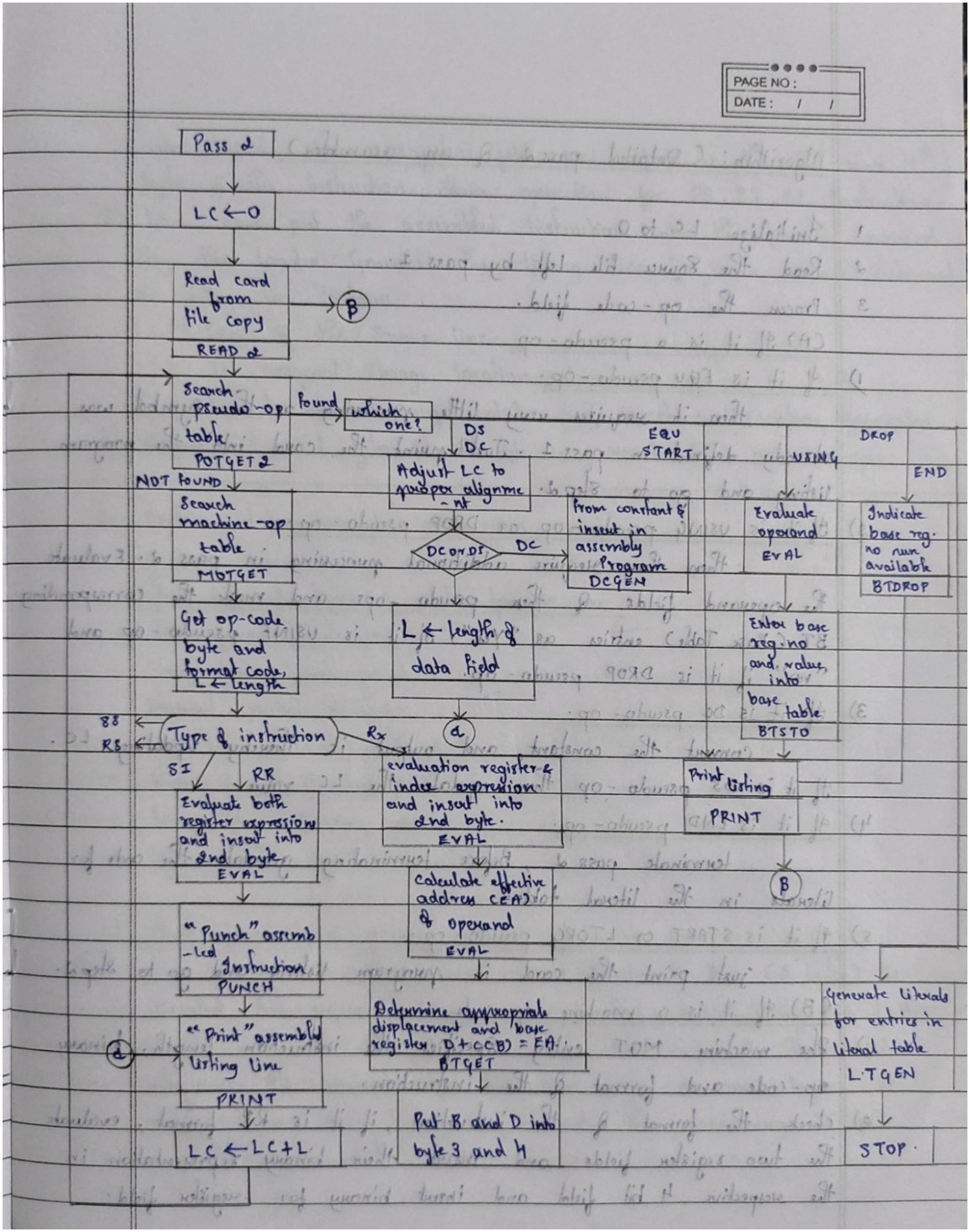
INDEX		CARD	
		MACRO	
15	&LAB	INCR	&ARG1, &ARG2, &ARG3
16	&LAB	A	1, &ARG1
17		A	2, &ARG2
18		A	3, &ARG3
19		MEND	

v) MNT : Macro Name Table
- Stores macro names.
- Serves as an index to MDT
- Pointers to the beginning and the end of the macro definition.

INDEX	NAME	MDT index
0		
1		
2		
3	"INCRbbbb"	15.

9. Write a neat chart for pass-2 of an assembler.

Pass 2 of an Assembler.


Pass 2
LC ← 0
Read card from file copy
READ 2
Search pseudo-op table
POTGET 2
FOUND
NOT FOUND
which one?
DS
DC
EQU
START
USING
DROP
END
Adjust LC to proper alignment
DC or DS
DC
from constant & insert in assembly Program
DCGEN
Evaluate operand
EVAL
Indicate base reg. no now available
BTDROP
Search machine-op table
MOTGET
L ← length of data field
Enter base reg. no and value into base table
BTSTO
Get op-code byte and format code, L ← length
Type of instruction
SS
RS
RX
SI
RR
evaluation register & index expression and insert into 2nd byte.
EVAL
Print listing
PRINT
Evaluate both register expressions and insert into 2nd byte
EVAL
calculate effective address (EA) & operand
EVAL
"Punch" assembled Instruction
PUNCH
Determine appropriate displacement and base register D+(C(B)) = EA
BTGET
Generate literals for entries in literal table
LTGEN
"Print" assembled listing line
PRINT
Put B and D into byte 3 and 4
STOP.
LC ← LC+L

Algorithm: (Detailed pass 2 of an assembler):

1 Initialize LC to 0.

2 Read the source file left by pass 1.

3 Process the op-code field.

(A) If it is a pseudo-op

1) If it is EQU pseudo-op.

then it requires very little processing as the symbol was already defined in pass 1. Just print the card into the program listing and go to step 2.

2) If it is USING pseudo-op or DROP pseudo-op.

then they require additional processing in pass 2. Evaluate the operand fields of these pseudo-ops and mark the corresponding BT (Base Table) entries as "yes", if it is USING pseudo-op and "no" if it is DROP pseudo-op.

3) If it is DC pseudo-op.

Convert the constant and output it thereby updating LC. If it is DS pseudo-op then update the LC value.

4) If it is END pseudo-op.

terminate pass 2, Before terminating generate the code for literals in the literal table.

5) If it is START or LTORG pseudo-op.

just print the card in program listing and go to step 2.

(B) If it is a machine op-code.

1) The machine MOT entry specifies the instruction length, binary op-code and format of the instruction.

2) check the format of the instruction, if it is RR format, evaluate the two register fields and insert their binary representation in the respective 4 bit field and insert binary for register field.

index field, base register field and Offset in the 3rd and 4th bytes of the instruction. Same operation for RS, SI, SS instructions.

3) Save and put the assembled instruction into the format required by the loader. Generate the listing line for each source statement. A listing line contains.
   (a) Copy of the source line.
   (b) Its assigned storage location.
   (c) The hexadecimal representation of machine instruction generated.

4) Update LC and goto step 2 till we reach END pseudo-op.

10. Explain data format with an example.

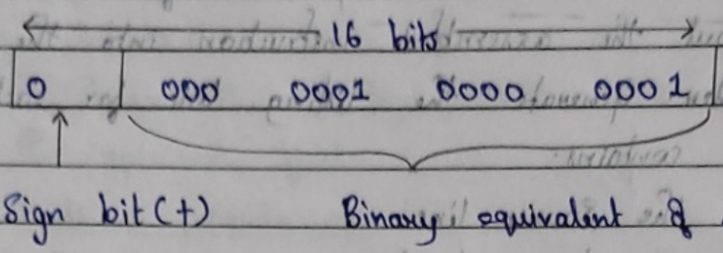
The different types of data formats are:

- Short form fixed point.
- Long form fixed point.
- Decimal packed numbers.
- Unpacked decimal numbers.
- Short form floating point.
- long form floating point.
- Logical (characters).

i) Short form fixed point numbers:-

In short form fixed point 16 bits (2 bytes) is allocated to represent an integer number, out of this the first 1 bit is used the sign (0 for + and 1 for -). The machine interprets the contents of this two bytes as an integer. It interprets the first bit as sign and the remaining 15 bits as a binary number.

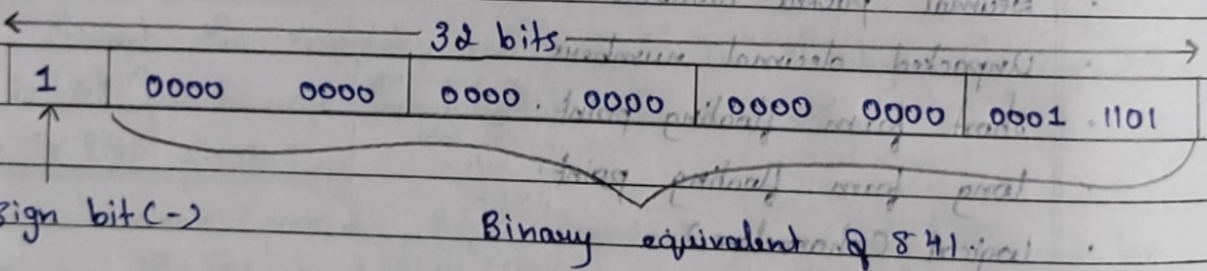
Example : Decimal number + 257 is represented as.



ii) Long form fixed point numbers:

In long form fixed point 32 bits (4 bytes) are allocated, out of this first one bit is reserved for sign (+ or -). The machine interprets the contents of this four bytes as an integer. It interprets the first bit as sign and the remaining 31 bits as a binary number.

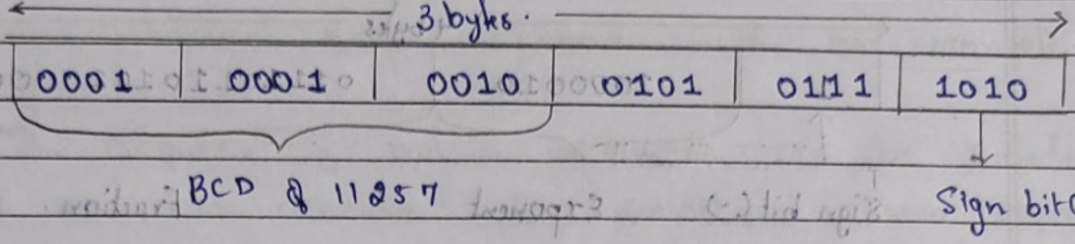
Example: Decimal number -541 is represented as:



iii) Packed decimal numbers:

Decimal digits, packed two to a byte, appear in fields of variable length (from 1 to 16 bytes) and are accompanied by a sign in the right most four bits. Instead of representing the binary numbers, numbers are represented in binary coded decimal format.

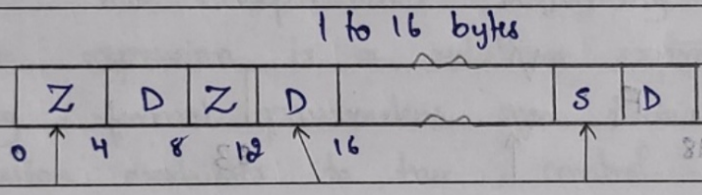
Example: Decimal number + 11257 is represented as.



iv) Unpacked Decimal numbers:

The size of unpacked decimal number varies from 1 to 16 byte out of which the last 1 byte (4+4 bits) is reserved for sign and data. Instead of representing the binary number, numbers are represented in binary coded decimal.

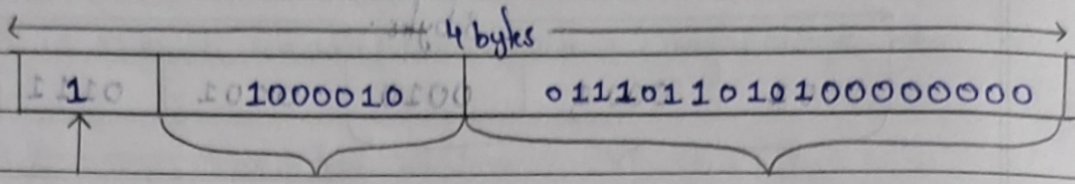
Example:



v) Short form floating point numbers:

In short form floating point 32 bits (4 bytes) are allocated. Out of this 1 bit is reserved for sign (+ or -). The floating point number is divided into exponential and fractional part. This representation gives a precision of seven decimal places.

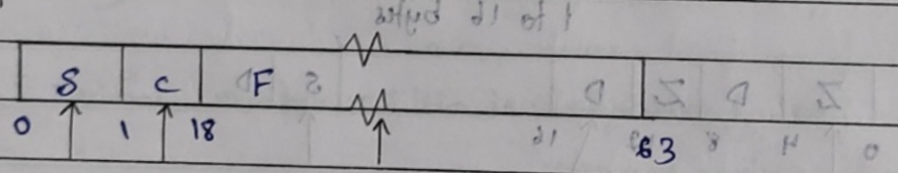
Example: Decimal number = 118.625 is represented as.



vi) Long form floating point number:-

This refers to a representation of a floating-point number that provides detailed information about its components and structure. In this format, a floating-point number is typically represented as a combination of three parts: the sign, the significand and the exponent.

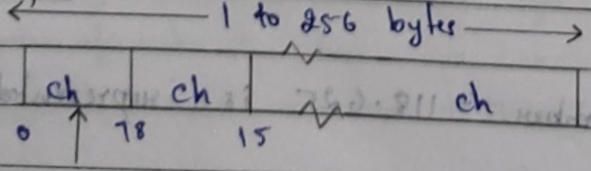
Example:-



vii) Logical (character data):-

Logical format, in the context of floating point numbers, typically refers to a representation that separates the components of a floating-point number explicitly and stores them as individual values rather than using a binary or decimal representation.

Example:-



PAGE NO :
DATE : / /

11. Explain conditional macro expansion with an example.

The sequence of macro expansion can be reordered based on some condition using conditional macro expansion. It allows conditional selection of the machine instruction that appear in expansions of a macro call. For this we use macro pseudo-ops. AIF and AGO.

AIF :-

AIF is a conditional branch pseudo-op. The format of AIF statement is as follows:

AIF <expression> <Sequencing Symbol>

where expression is a relational expression involving ordinary string, formal parameters and their attributes. If the expression evaluates to true, control is transferred to the statement containing sequencing symbol in its label field. If it is false, the very next statement following AIF is executed. AIF statement does not appear in the expanded source code.

AGO :-

AGO is an unconditional branch pseudo-op. The format of AGO statement is as follows.

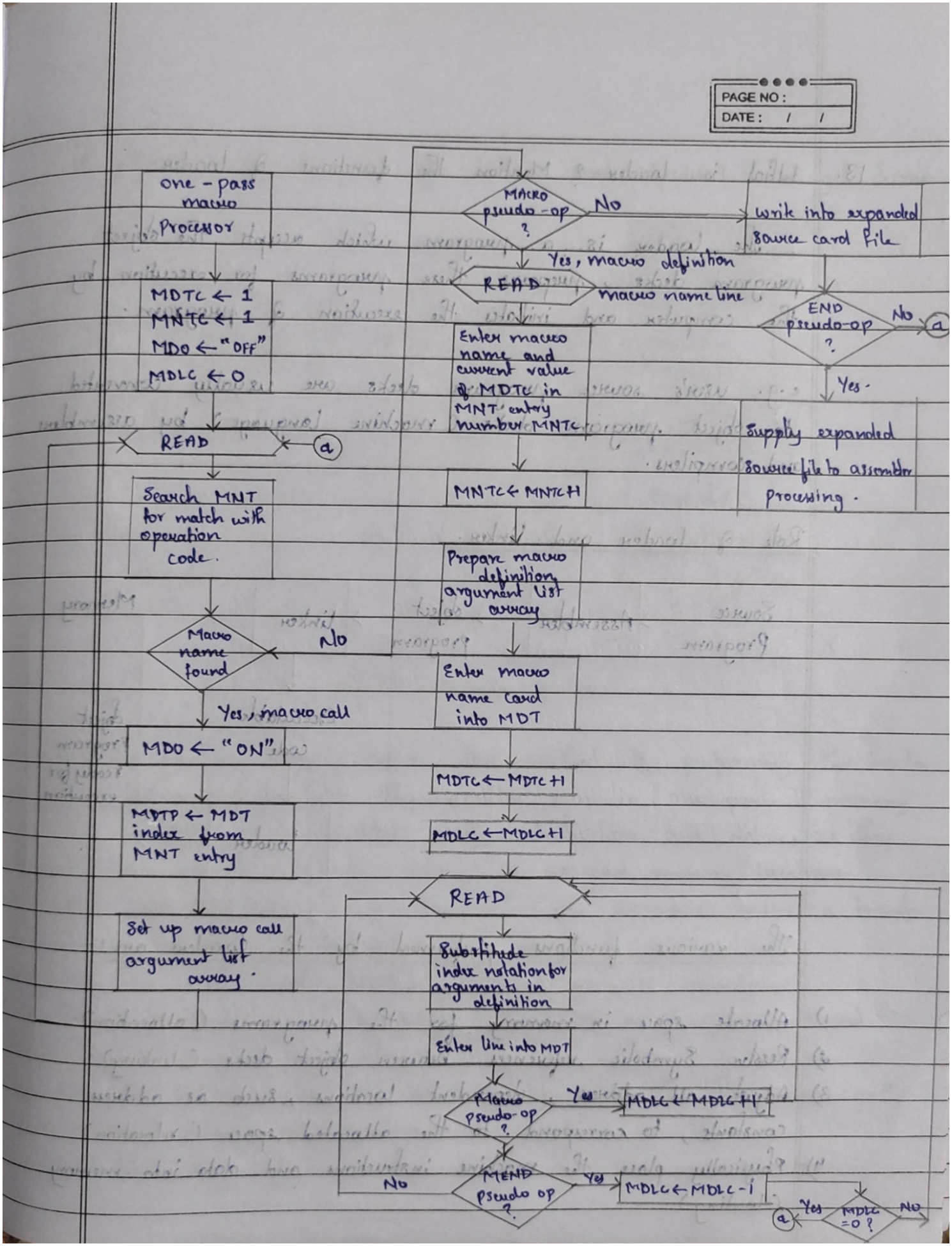
AGO <Sequencing Symbol>

It unconditionally transfers control to the statement containing sequencing symbol in its label field. AGO statement does not appear in the expanded source code.

12. Write a neat flowchart for Single Pass Macro processor.

A Single - Pass Algorithm.

1) Suppose we need to provide for macro definitions within macros.
2) The basic problem here is that the inner macro is defined only after the outer one has been called: in order to provide for any use of the inner macro, we would have to repeat both the macro - definition and macro - call passes, hence
3) We use all macro processing to single pass.
4) In order to keep track of more than one macro we will use MDLC (Macro Definition level counter) and MDI (Macro Definition Input) indicator.
5) MDI = "ON" during expansion of Macro calls and MDI = OFF at all other times.
6) Actual expansion of macro calls is performed in READ box, which is detailed in second flowchart.
7) READ test the switch MDI if it is ON lines are read from MDT during this MEND indicates the end of the macro then MDI = "OFF" that is it will be reset.
8) Macro Definition level Counter is incremented by 1 when a MACRO Pseudo - OP is encountered and decremented by 1 when MEND Pseudo - OP is encountered.

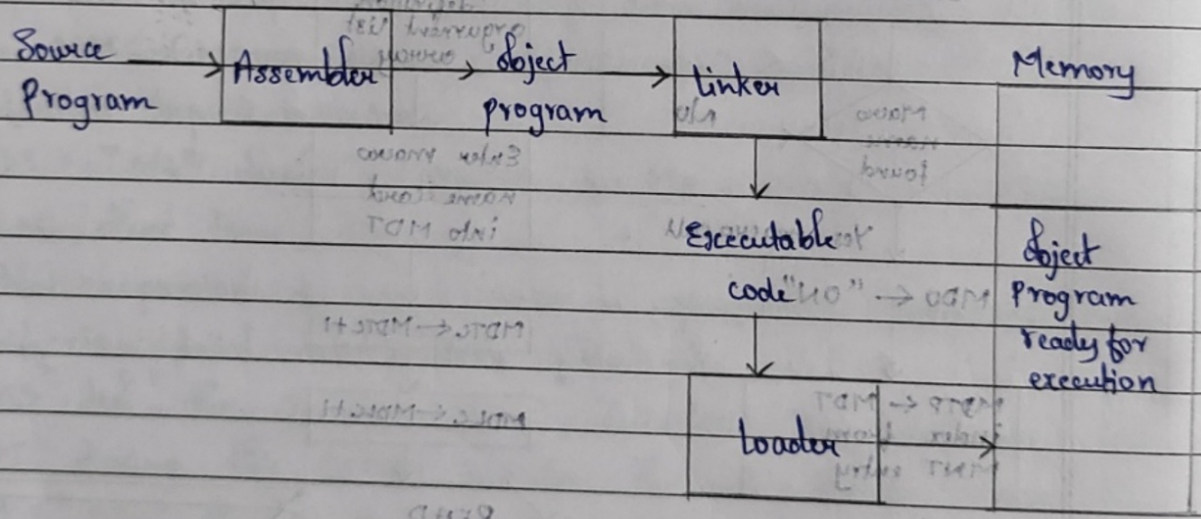

one - pass macro Processor
MDTC ← 1
MNTC ← 1
MDO ← "OFF"
MDLC ← 0
READ
a
Search MNT for match with operation code.
Macro name found
No
Yes, macro call
MDO ← "ON"
MDTP ← MDT index from MNT entry
Set up macro call argument list array.
MACRO pseudo-op ?
No
write into expanded source card file
Yes, macro definition
READ
macro name line
Enter macro name and current value of MDTC in MNT entry number MNTC
MNTC ← MNTC+1
Prepare macro definition, argument list array
Enter macro name card into MDT
MDTC ← MDTC+1
MDLC ← MDLC+1
READ
Substitude index notation for arguments in definition
Enter line into MDT
Macro pseudo-op ?
Yes
MDLC ← MDLC+1
MEND pseudo op ?
No
Yes
MDLC ← MDLC-1
a
Yes
MDLC =0 ?
NO
END pseudo-op ?
No
a
Yes.
Supply expanded source file to assembler processing.

13. What is loader ? Mention the functions of loader.

The loader is a program which accepts the object program decks, prepares these programs for execution by the computer and initiates the execution of program.

e.g. user's source program decks are usually converted to object program decks (machine language) by assemblers and compilers.

Role of loader and linker.

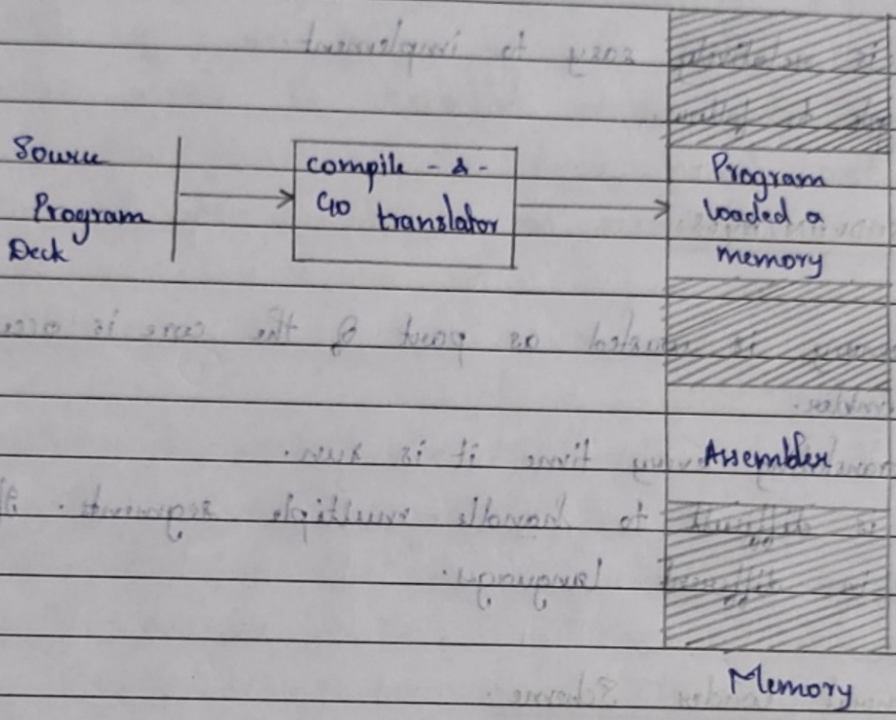


The various functions performed by the loaders are :-

1) Allocate space in memory for the programs (allocation).
2) Resolve symbolic references between object decks (linking).
3) Adjust all address, dependent locations, such as address constants, to correspond to the allocated space (relocation)
4) Physically place the machine instructions and data into memory (loading).

PAGE NO :
DATE : / /

14. Explain Compile - and - go loader and General loading Scheme.



Compile - & - Go loader.

1) Compile - and - Go loaders One method of performing the loader functions is to have the assembler run in one part of memory and place the assembled machine instructions and data, as they are assembled, directly into their assigned memory locations.
2) When the assembly is completed, the assembler causes a transfer to the starting instruction of the program.
3) This is a simple solution, involving no extra procedures. It is used by the WATFOR FORTRAN compiler and several other language processors.
4) Such a loading scheme is commonly called " compile - and - go" or " assemble - and - go".

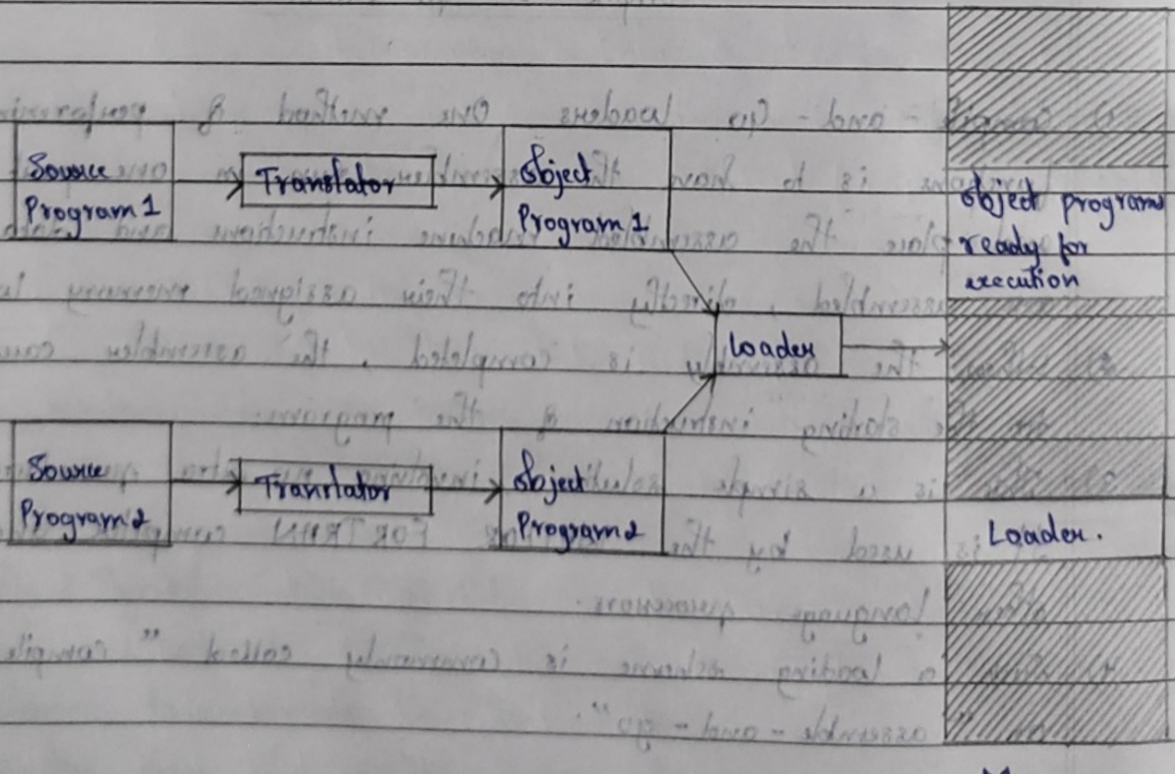
ADVANTAGES:

- It is relatively easy to implement.
- Simple to follow.

DISADVANTAGES:

- Memory is wasted as part of the core is occupied by Assembler.
- Retranslating every time it is run.
- It is difficult to handle multiple segments. If soure programs are in different languages.

> General loader Scheme.



General loader

PAGE NO :
DATE : / /

1) The use of an object deck as intermediate data to avoid one disadvantage of the preceding "compile and go" scheme requires the addition of a new program to the system, a loader.
2) The loader accepts the assembled machine instructions, data and other information present in the object format, and places machine instructions and data in core in an executable computer form.

ADVANTAGE:-

- The loader is assumed to be, smaller than the assembler, so that more memory is available to the user.
- A further advantage is that reassembly is no longer necessary to run the program at a later stage.

15. Write a flowchart for single pass assembler (PASS-1 of an Assembler)

Algorithm: (Detailed pass 1 of an assembler):-

1) Initialize the location counter (LC) to the first location in the program (Set LC to relative address 0)
2) Read the source statement.
3) Examine the operation code field to determine:-

(a) If it is a pseudo-op:

1) If it is USING OR DROP:-

pass 1 it's only concerned with the pseudo-ops that either define the symbols or affect LC. USING and DROP do neither, the assembler only has to save the USING and DROP cards and go to step 2.

2) If it is a START pseudo-op.

examine the label field and store that label along with LC value in Symbol table, save the card for pass 2 and go to step 2.

3) If it is EQU.

concerned with defining the Symbol in label field, evaluate the expression in the operand field. Store the Symbol and expression in the operand field of EQU into Symbol table. Save this card and go to step 2.

4) If it is DC and DS statement.

it will affect both LC and definition of Symbol in determining the number of bytes of storage required. Adjust LC and store the label with the LC value before the updation in Symbol table. Save the card for pass 2 and go to step 2.

5) If it is LTORG pseudo-op.

determine the number of bytes of storage required for the literals used from the start of the program till LTORG statement is encountered, correspondingly update. Save this card for pass 2 and go to step 2.

6) If it is an END pseudo-op

pass 1 will be terminated. But before transfering the control to pass 2, some house keeping operations are to be performed. They are (a) assign memory to literals (b) Rewind and reset the source copy file.

(b) If it is machine op-code, search MOT (machine op-table) for a match with the op-code. Get the length of the instruction. Enter the operand into the literal table.

(c) Then label field of the source statement is examined for the presence of the label. If there is a label save the label along with

LC value into the Symbol table.

(d) Update the LC with length of the instruction and save the Source Statement for pass 2.

(e) Go to step 2 (This process is repeated until it encounters END pseudo-op