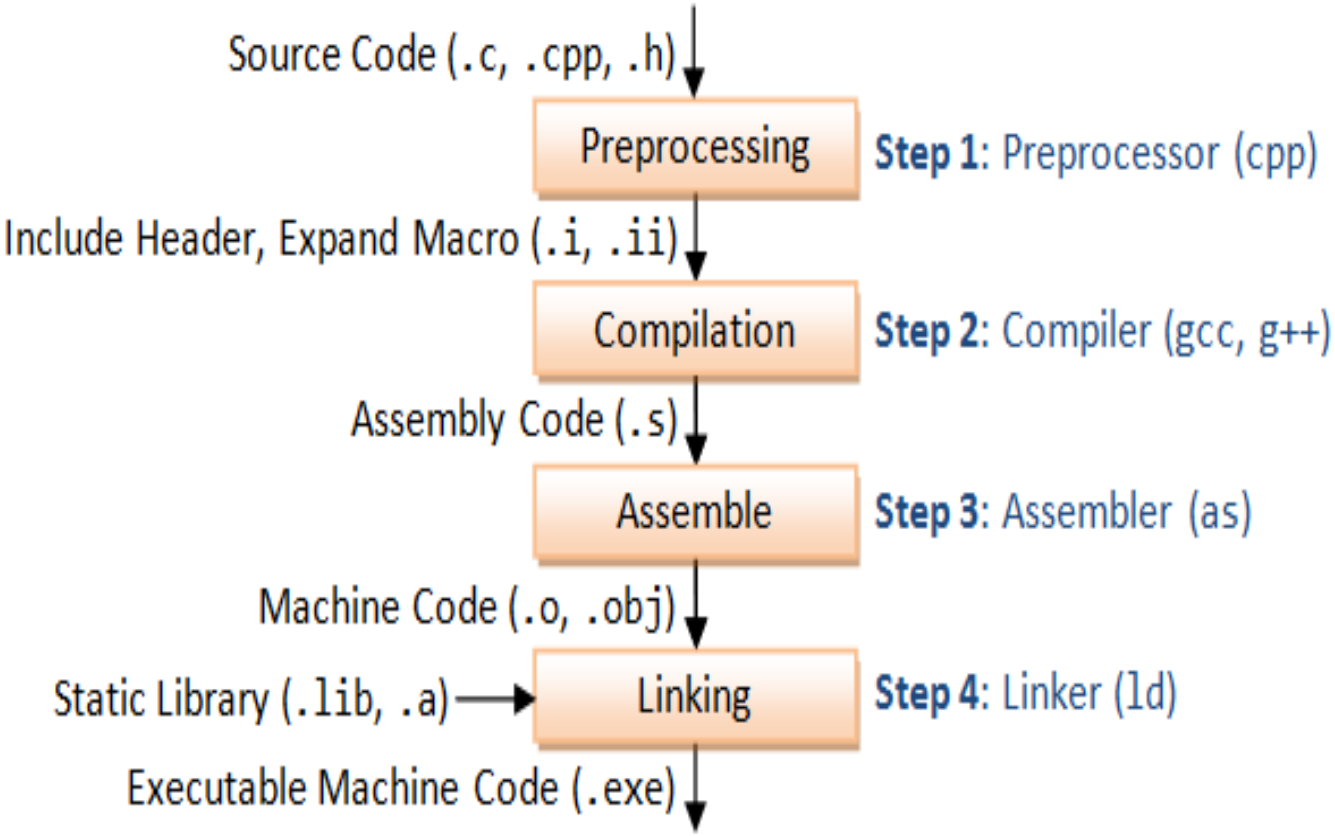
Contents

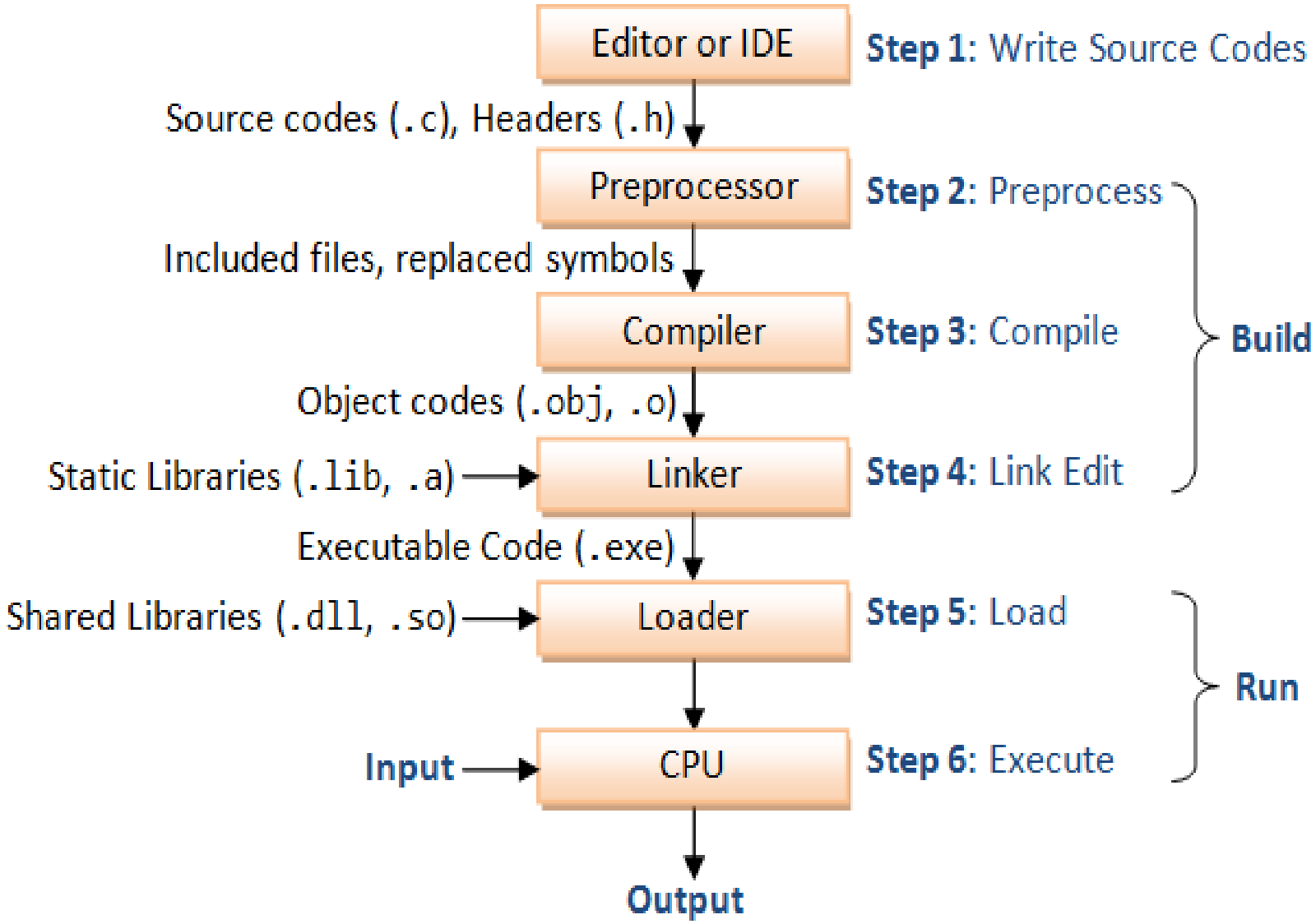
- What is an assembly language programming (ALP)?
- What is Assembler?
- Applications of Assembly Language
- Advantages of AL
- Disadvantages of AL
- Elements of ALP
  - Statement Format and Machine Instruction Format
- Types of AL Statements
- A Simple Assembly Scheme
- Pass Structure of Assembler
- Design of a Two pass Assembler
  - Pass I of the Assembler
  - Intermediate code form (Variant I and Variant II)
  - Pass II of the Assembler

Assembly Language Programming (ALP)

- Assembly language is a kind of low level programming language, which uses symbolic codes or mnemonics as instruction.
- Some examples of mnemonics include ADD, SUB, LDA, and STA that stand for addition, subtraction, load accumulator, and store accumulator, respectively.
- For processing of an assembly language program we need a language translator called assembler
- Assembler- Assembler is a Translator which translates assembly language code into machine code

Position of Assembler




Editor or IDE
Step 1: Write Source Codes
Source codes (.c), Headers (.h)
Preprocessor
Step 2: Preprocess
Included files, replaced symbols
Compiler
Step 3: Compile
Build
Object codes (.obj, .o)
Static Libraries (.lib, .a)
Linker
Step 4: Link Edit
Executable Code (.exe)
Shared Libraries (.dll, .so)
Loader
Step 5: Load
Run
Input
CPU
Step 6: Execute
Output

Applications of Assembly Language

- **assembly language** is used for direct hardware manipulation, access to specialized processor instructions, or to address critical performance issues.
- Typical **uses** are device drivers (CD, HDD), low-level embedded systems (Keyboard, water tank indicator) and real-time systems (computer, notepad).

Advantage and Disadvantages of ALP

- **Advantages-**
- Due to use of symbolic codes (mnemonics), an assembly program can be written faster.
- It makes the programmer free from the burden of remembering the operation codes and addresses of memory location.
- It is easier to debug.

- **Disadvantages-**
- it is a machine oriented language, it requires familiarity with machine architecture and understanding of available instruction set.
- Execution in an assembly language program is comparatively time consuming compared to machine language. The reason is that a separate language translator program is needed to translate assembly program into binary machine code

Elements of ALP

- An assembly language provides the following three basic facilities that simplify programming:
- **Mnemonic operation codes:** The mnemonic operation codes for machine instructions (also called mnemonic opcodes) are easier to remember and use than numeric operation codes. Their use also enables the assembler to detect use of invalid operation codes in a program.

- **Symbolic operands**: A programmer can associate symbolic names with data or instructions and use these symbolic names as operands in assembly statements. This facility frees the programmer from having to think of numeric addresses in a program. We use the term symbolic name only in formal contexts; elsewhere we simply say name.

- **Data declarations**: Data can be declared in a variety of notations, including the decimal notation. It avoids the need to manually specify constants in representations that a computer can understand, for example, specify 5 as (11111011).

Statement Format

[Label] <*Opcode*> <*operand specification*>[,<*operand specification*>..] ;Comments

- [..] enclosed specification is optional
- Ex: AGAIN MULT BREG,AREG; multiply
- Label-label is associated with symbolic name with memory word. ex AGAIN etc.
- Symbolic name= Mnemonic Operation Code (Mnemonic opcode) ex MULT
- Memory word= Operand Specification ex BREG and AREG etc

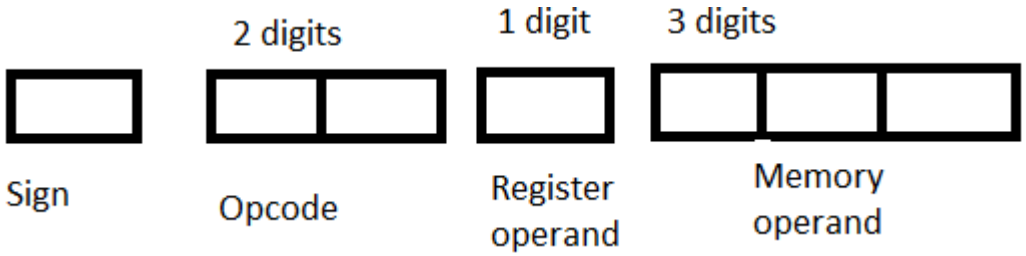
Statement Format cont'd

- Operand spec. syntax-

  <symbolic name> [± *<displacement>*] [(*<index register>*)]

  – The operand AREA refers to the memory word with which the name AREA is associated.
  – The operand AREA+5 refers to the memory word that is 5 words away from the word with the name AREA. Here '5' is the displacement or offset from AREA.
  – The operand AREA(4) implies indexing the operand AREA with index register 4 that is, the operand address is obtained by adding the contents of index register 4 to the address of AREA.
  – The operand AREA+5 (4) is a combination of the previous two specifications.

Machine Instruction Format



Ex : + 09 0 113

- Opcode, reg.operand and Memory Operand occupy 2,1,and 3 digits.
- This is a Machine code which will produce by assembler.

Types of Assembly Language statements

- **Imperative statements (IS)**
  - An imperative statement in assembly language indicates the action to be performed during execution of assembly statement
  - Ex:- A 1,FOUR
- **Declarative Statement (DL)**
  - These statements declares the storage area or declares the constant in program.
  - [Lable] DS <constant> ex : A DS 1
  - [Lable] DC '<Value>' ex : ONE DC '1'
- Declaration statements are for reserving memory for variables. DS- Declare storage and DC- Declare constant
- (A DS 1) statement reserves 1 word of memory for variable A
- (ONE DC '1') statement associates the name ONE with a memory word containing the value '1'

Types of Assembly Language statements cont'd

- **Assembler Directives**
  - It instructs the assemble to perform certain actions during the assembly of a program.
  - START <constant > ex: START 200
  - END [<operand spec.>] ex: END
  - **Advance Assembler Directives**
  - ORIGIN <address spec.> ex: ORIGIN LOOP+2
  - <symbol> EQU <address spec.> ex: BACK EQU LOOP
  - LTORG <='value'> ex LTORG ='1'
    - Where <Address spec.> can be <constant> or <symbol name> + <displacement>

Advance assemble directives

- ORIGIN- This directive instructs the assembler to put the address given by <address specification> in the location counter
- EQU- The statement simply associates the name <symbol> with the address specified by <address specification>. However, the address in the location counter is not affected.
- LTORG- The LT0RG directive, which stands for 'origin for literals', allows a programmer to specify where literals should be placed
  - If a program does not use an LTORG statement, the assembler would enter all literals used in the program into a single pool and allocate memory to them when it encounters the END statement.

Use of ORIGIN, EQU and LTORG

1		START	200		
2		MOVER	AREG, =’5’	200)	+04 1 211
3		MOVEM	AREG, A	201)	+05 1 217
4	LOOP	MOVER	AREG, A	202)	+04 1 217
5		MOVER	CREG, B	203)	+04 3 218
6		ADD	CREG, =’1’	204)	+01 3 212
7		…			
12		BC	ANY, NEXT	210)	+07 6 214
13		LTORG			
			=’5’	211)	+00 0 005
			=’1’	212)	+00 0 001
14		…			
15	NEXT	SUB	AREG, =’1’	214)	+02 1 219
16		BC	LT, BACK	215)	+07 1 202
17	LAST	STOP		216)	+00 0 000
18		ORIGIN	LOOP + 2		
19		MULT	CREG, B	204)	+03 3 218
20		ORIGIN	LAST + 1		
21	A	DS	1	217)	
22	BACK	EQU	LOOP		
23	B	DS	1	218)	
24		END			
25			=’1’	219)	+00 0 001

•ORIGIN- Statement number 18 of the program viz. ORIGIN LOOP + 2 puts the address 204 in the location counter because symbol LOOP is associated with the address 202. The next statement MULT CREG, B is therefore given the address 204.

•EQU-On encountering the statement BACK EQU LOOP, the assembler associates the symbol BACK with the address of LOOP i.e. with 202

• LTORG- The literals ='5' and ='1' are added to the literal pool in Statements 2 and 6, respectively. The first LTORG statement (Statement 13) allocates the addresses 211 and 212 to the values '5' and ‘1’.

A Simple Assembly Scheme

- Design Specification of an assembler
  - Four step approach to develop a design specification
  1) Identify the information necessary to perform a task
  2) Design a suitable data structure to record the information
  3) Determine the processing necessary to obtain and maintain the information
  4) Determine the processing necessary to perform the task

A Simple Assembly Scheme cont'd

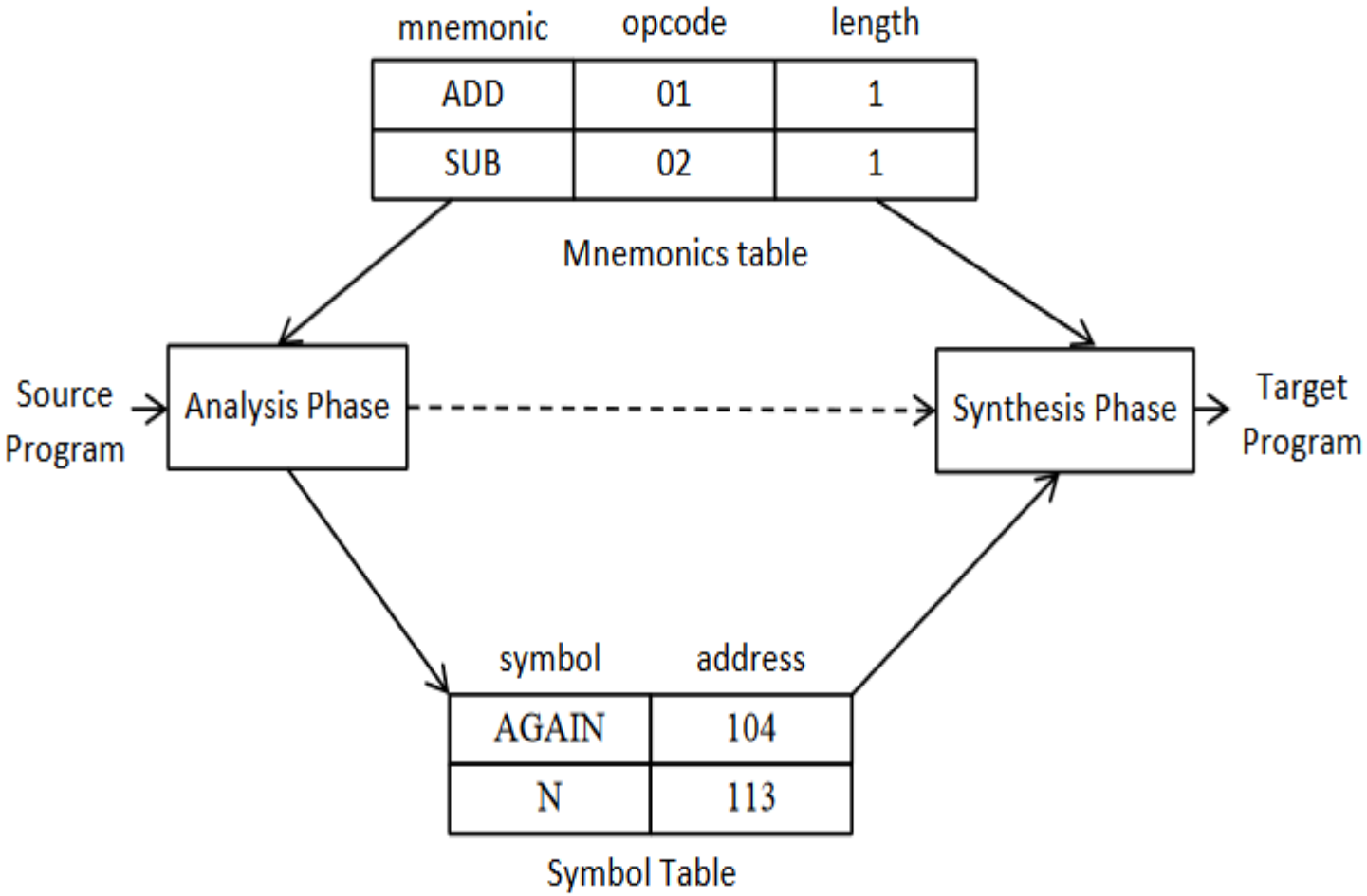


Fig.-Design of assembler

Analysis phase

- The primary function performed by the analysis phase is the building of the symbol table.
- For this purpose it must determine address of the symbolic name. This function is called memory allocation.
- To implement memory allocation a data structure called location counter (LC) is used, it is initialized to the constant specified in the START statement.
- We refer the processing involved in maintaining the location counter as LC processing.
- **Tasks performed Analysis phase**

  1.Isolate the label, mnemonics opcode, and operand fields of a constant.

  2.If a label is present, enter the pair (symbol, <LC content>) in a new entry of symbol table.

  3.Check validity of mnemonics opcode.

  4.Perform LC processing.

Synthesis phase

- Consider the assembly statement,
  - MOVER BREG, ONE
- We must have following information to synthesize the machine instruction corresponding to this statement:
  - 1.Address of name ONE
  - 2.Machine operation code corresponding to mnemonics MOVER.

- The first item (ONE)of information depends on the source program; hence it must be available by analysis phase.
- The second item (MOVER)of information does not depend on the source program; it depends on the assembly language.

- Based on above discussion, we consider the use of two data structure during synthesis phase:
  - Symbol table-Each entry in symbol table has two primary field name and address. This table is built by analysis phase
  - Mnemonics table- An entry in mnemonics table has two primary field mnemonics and opcode.

Synthesis phase cont’d

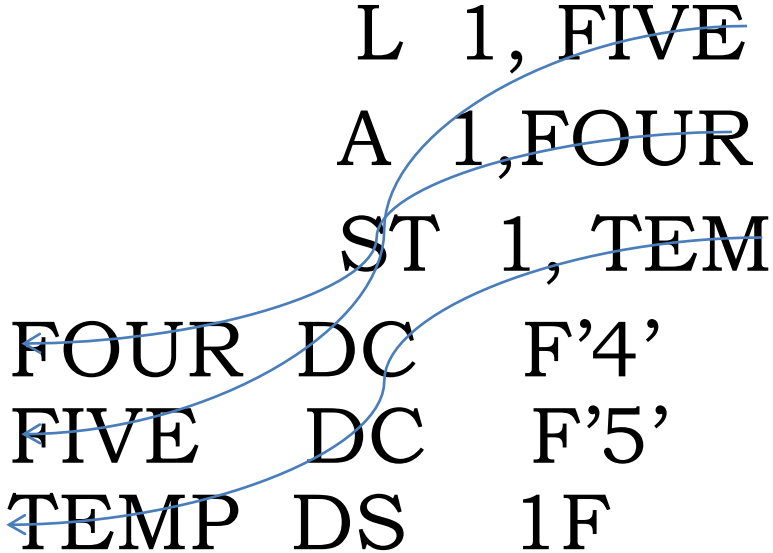
- **Task performed by Synthesis phase**
  - 1.Obtain machine opcode through look up in the mnemonics table.
  - 2.Obtain address of memory operand from the symbol table.
  - 3.Synthesize a machine instruction.

Pass Structure of Assembler

- **Single pass translation**
  - A one pass assembler requires 1 scan of the source program to generate machine code.
  - In one Pass translation forward reference problem occures
- Problem of Forward Reference
  - When the variables are used before their definition at that time problem of forward reference accurse.

Problem of Forward Reference

```
JOHN   START    0
       USING *, 15
        L  1, FIVE
        A  1,FOUR
        ST  1, TEMP
FOUR  DC    F'4'
FIVE   DC    F'5'
TEMP  DS    1F
       END
```



Solution of forward reference problem

- The process of forward references is tackled using a process called back patching.
- The operand field of an instruction containing forward references is left blank initially.
- A table of instruction containing forward references is maintained separately called table of incomplete instruction (TII).
- This table can be used to fill up the addresses in incomplete instruction.
- The address of the forward referenced symbols is put in the blank field with the help of back patching list

General Design Procedure of Two Pass Assembler

1. Specify the problem
2. Specify data structures
3. Define format of data structures
4. Specify algorithm
5. Look for modularity [capability of one program to be subdivided into independent programming units.]
6. Repeat 1 through 5 on modules.

Specify the problem pass I

Pass1: Define symbols & literals.

1) Determine length of m/c instruction [MOTGET1]
2) Keep track of Location Counter [LC]
3) Remember values of symbols [STSTO]
4) Process some pseudo ops[EQU,DS etc] [POTGET1]
5) Remember Literals [LITSTO]

Specify the problem pass II

Pass2: Generate object program

1) Look up value of symbols [STGET]
2) Generate instruction [MOTGET2]
3) Generate data (for DS, DC & literals)
4) Process pseudo ops[POTGET2]

Step 2. Data structure:-

Pass1: Databases

- Input source program
- “LC” location counter used to keep track of each instructions addr.
- M/c operation table (MOT) contain a field [mnemonic opcode, class and mnemonic information (information code of IS, length)]
- Pseudo operation table [POT] contain a field [mnemonic opcode, class and mnemonic information (R#routine number)]
- Symbol Table (ST or SYMTAB) to store each lable & it’s value.
- Literal Table (LT or LTTAB), to store each literal (variable) & it’s location.
- Literal Pool Table (POOLTAB)
- Copy of input to used later by PASS-2.

MOT+POT=OPTAB

- OPTAB contains the field mnemonics opcodes, class and mnemonics info.
- The class field indicates whether the opcode belongs to an imperative statement (IS), a declaration statement (DS), or an assembler directive (AD).
- If an imperative, the mnemonics info field contains the pair (machine code, instruction length), else it contains the id of a routine to handle the declaration statement or assembler directive statement

1		START	200		
2		MOVER	AREG, =’5’	200)	+04 1 211
3		MOVEM	AREG, A	201)	+05 1 217
4	LOOP	MOVER	AREG, A	202)	+04 1 217
5		MOVER	CREG, B	203)	+04 3 218
6		ADD	CREG, =’1’	204)	+01 3 212
7		…			
12		BC	ANY, NEXT	210)	+07 6 214
13		LTORG			
			=’5’	211)	+00 0 005
			=’1’	212)	+00 0 001
14		…			
15	NEXT	SUB	AREG, =’1’	214)	+02 1 219
16		BC	LT, BACK	215)	+07 1 202
17	LAST	STOP		216)	+00 0 000
18		ORIGIN	LOOP + 2		
19		MULT	CREG, B	204)	+03 3 218
20		ORIGIN	LAST + 1		
21	A	DS	1	217)	
22	BACK	EQU	LOOP		
23	B	DS	1	218)	
24		END			
25			=’1’	219)	+00 0 001

Mnemonic Operation Table (OPTAB)

Mnemonics Opcode	Class	Mnemonics information
START	AD	R#1
MOVER	IS	(04,1)
MOVEM	IS	(05,1)
ADD	IS	(01,1)
BC	IS	(07,1)
LTORG	AD	R#5
SUB	IS	(02,1)
STOP	IS	(00,1)
ORIGIN	AD	R#3
MULT	IS	(03,1)
DS	DL	R#7
EQU	AD	R#4
END	AD	R#2

Machine opcode table (MOT)

Mnemonics Opcode	Class	Mnemonics information
MOVER	IS	(04,1)
MOVEM	IS	(05,1)
ADD	IS	(01,1)
BC	IS	(07,1)
SUB	IS	(02,1)
STOP	IS	(00,1)
MULT	IS	(03,1)

M/c operation table (MOT) contain a field [mnemonic opcode, class and mnemonic information (information code of IS, length)]

Pseudo operation table (POT)

- Pseudo operation table [POT] contain a field [mnemonic opcode, class and mnemonic information (R#routine nomber)]

Mnemonics Opcode	Class	Mnemonics information
START	AD	R#1
LTORG	AD	R#5
ORIGIN	AD	R#3
DS	DL	R#7
EQU	AD	R#4
END	AD	R#2

Symbol Table (SYMTAB)

Index no.	Symbol	Address	Length
1	LOOP	202	1
2	NEXT	214	1
3	LAST	216	1
4	A	217	1
5	BACK	202	1
6	B	218	1

- A SYMTAB entry contains the symbol name, field address and length.
- Some address can be determining directly, e.g. the address of the first instruction in the program, however other must be inferred.
- To find address of other we must fix the addresses of all program elements preceding it. This function is called memory allocation

Literal Table (LITTAB)

Index no.	Literal	Address
1	=‘5’	211
2	=‘1’	212
3	=‘1’	219

- A table of literals used in the program.
- A LITTAB entry contains the field literal and address.
- The first pass uses LITTAB to collect all literals used in a program

Pool Table (POOLTAB)

Literal no.
#1
#3

- Awareness of different literal pools is maintained using the auxiliary table POOLTAB.
- This table contains the literal number of the starting literal of each literal pool.
- At any stage, the current literal pool is the last pool in the LITTAB.
- On encountering an LTORG statement (or the END statement), literals in the current pool are allocated addresses starting with the current value in LC and LC is appropriately incremented.

Step 2. Data structure:-

- **Pass2: Databases**
- Copy of source program input to Pass1.
- Location Counter (LC)
- MOT [Mnemonic, length, binary m/c op code, etc.]
- POT [Mnemonic & action to be taken in Pass2
- ST [prepared by Pass1, label & value]
- Base Table [or register table] indicates which registers are currently specified using ‘USING’ pseudo op & what are contents.
- Literal table prepared by Pass1. [Lit name & value].

Format of Data Structures pass I

- **Intermediate Code**
- Intermediate code consist of a set of IC units, each unit consisting of the following three fields
- 1.Address
- 2.Representation of mnemonics opcode
- 3.Representation of operands
- Intermediate code can be in variant I or variant II form

Variant I

- **Mnemonics field**
- The mnemonics field contains a pair of the form
  - (statement class, code)
  - Where statement class can be one of IS, DL, and AD
- For imperative statement, code is the instruction opcode in the machine language.
- For declarations and assembler directives, code is an ordinal number within the class.
- (AD, 01) stands for assembler directive number 1 which is the directive START

Variant I

Declaration statement	Instruction code
DC	01
DS	02

Assembler directive	Instruction code
START	01
END	02
ORIGIN	03
EQU	04
LTORG	05

Imperative statements (mnemonics)	Instruction code
STOP	00
ADD	01
SUB	02
MULT	03
MOVER	04
MOVEM	05
COMP	06
BC	07
DIV	08
READ	09
PRINT	10
JUMP	11

Variant I

Condition	Instruction code
LT	1
LE	2
EQ	3
GT	4
GE	5
ANY	6

Register	Instruction code
AREG	1
BREG	2
CREG	3
DREG	4

- First operand is represented by a single digit number which is a code for a register or the condition code
- The second operand, which is a memory operand, is represented by a pair of the form (operand class, code)
- Where operand class is one of the C, S and L standing for constant, symbol and literal

For a constant, the code field contains the internal representation of the constant itself.
Ex: the operand descriptor for the statement START 200 is (C,200).
For a symbol or literal, the code field contains the ordinal number of the operand's entry in SYMTAB or LITTAB

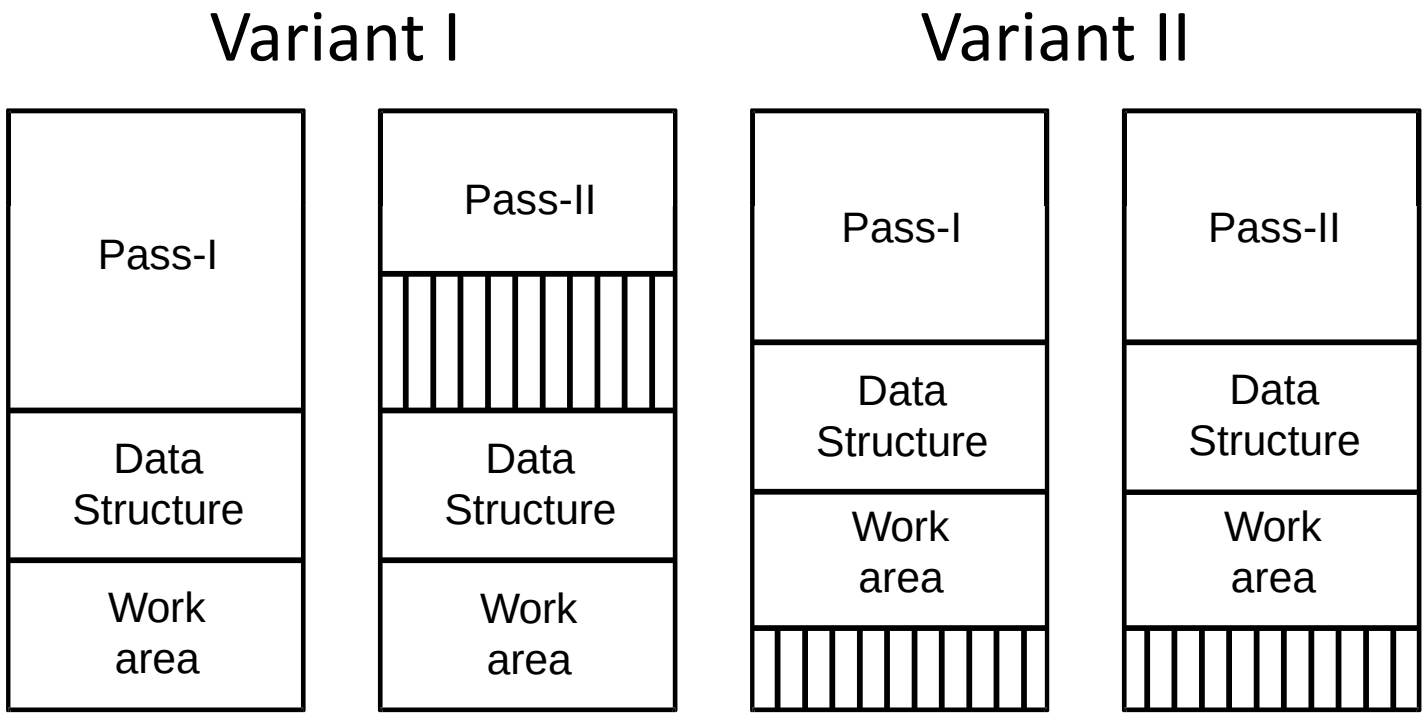
Variant II

- This variant differs from variant I of the intermediate code because in variant II symbols, condition codes and CPU register are not processed.
- So, IC unit will not generate for that during pass I.

Variant I	Variant II
IS, DL and AD all statements contain processed form.	DL and AD statements contain processed form while for Is statements, operand field is processed only to identify literal references.
Extra work in pass I	Extra work in pass II
Simplifies tasks in pass II	Simplifies tasks in pass I
Occupies more memory than pass II	Memory utilization of two passes gets better balanced.

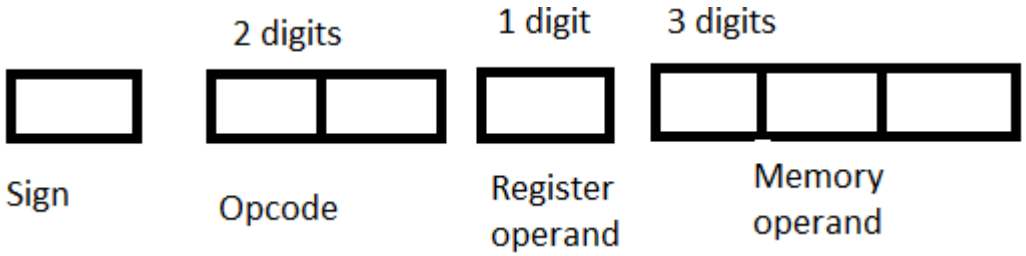
Intermediate Code Forms

- Comparison of Variant-I and Variant-II
  - Memory requirement using Variant-I and Variant-II.



Stmt no	Lable	mn opcode	operands	LC	sign op-code	Reg ope.	Memory Ope.	Var I	Var II
1		START	200				(AD,1) (C,200)		(AD,1) (C,200)
2		MOVER	AREG, ='5'	200)	+04 1 211		(IS,4) (1) (L,1)		(IS,4) AREG (L,1)
3		MOVEM	AREG, A	201)	+05 1 217		(IS,5) (1) (S,4)		(IS,5) AREG (S,4)
4	LOOP	MOVER	AREG, A	202)	+04 1 217		(IS,4) (1) (S,4)		(IS,4) AREG (S,4)
5		MOVER	CREG, B	203)	+04 3 218		(IS,4) (3) (S,6)		(IS,4) CREG (S,6)
6		ADD	CREG, ='1'	204)	+01 3 212		(IS,1) (3) (L,2)		(IS,1) CREG (L,2)
7		...							
12		BC	ANY, NEXT	210)	+07 6 214		(IS,7) (6) (S,2)		(IS,4) ANY, NEXT
13		LTORG					(AD,5)		(AD,5)
			='5'	211)	+00 0 005		(L,1)		(L,1)
			='1'	212)	+00 0 001		(L,2)		(L,2)
14		...							
15	NEXT	SUB	AREG, ='1'	214)	+02 1 219		(IS,2) (1) (L,3)		(IS,2) AREG (L,3)
16		BC	LT, BACK	215)	+07 1 202		(IS,7) (1) (S,5)		(IS,7) LT, BACK
17	LAST	STOP		216)	+00 0 000		(IS,00)		(IS,00)
18		ORIGIN	LOOP + 2				(AD,3) (S,1)+2		(AD,3) LOOP+2
19		MULT	CREG, B	204)	+03 3 218		(IS,3) (3) (S,6)		(IS,3) CREG ,B
20		ORIGIN	LAST + 1				(AD,3) (S,3)+1		(AD,3) LAST+1
21	A	DS	1	217)			(DL,02) (C,1)		(DL,02) A
22	BACK	EQU	LOOP				(AD,04) (S,1)		(AD,04) (S,1)
23	B	DS	1	218)			(DL,02) (C,1)		(DL,02) B
24		END					(AD,02)		
25			='1'	219)	+00 0 001		(L,3)		(L,3)

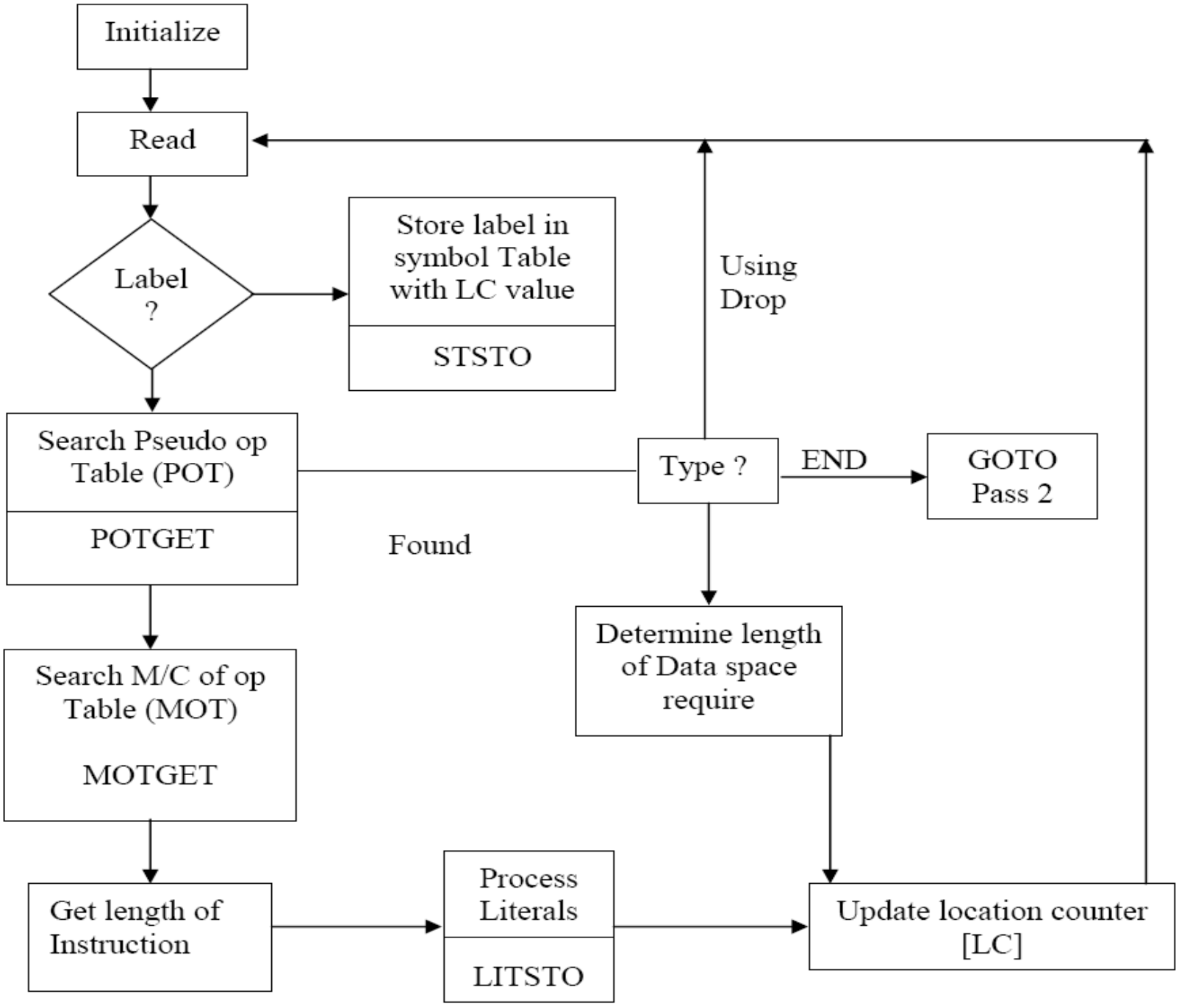
Format of Data Structures pass II Machine Instruction Format



Ex : + 09 0 113

- Opcode, reg.operand and Memory Operand occupy 2,1,and 3 digits.
- This is a Machine code which will produce by assembler.

Pass – I of ASSEMBLER Flow Chart


Initialize
Read
Label ?
Store label in symbol Table with LC value
STSTO
Using Drop
Search Pseudo op Table (POT)
POTGET
Type ?
END
GOTO Pass 2
Found
Search M/C of op Table (MOT)
MOTGET
Determine length of Data space require
Get length of Instruction
Process Literals
LITSTO
Update location counter [LC]

Pass – llof ASSEMBLER Flow Chart

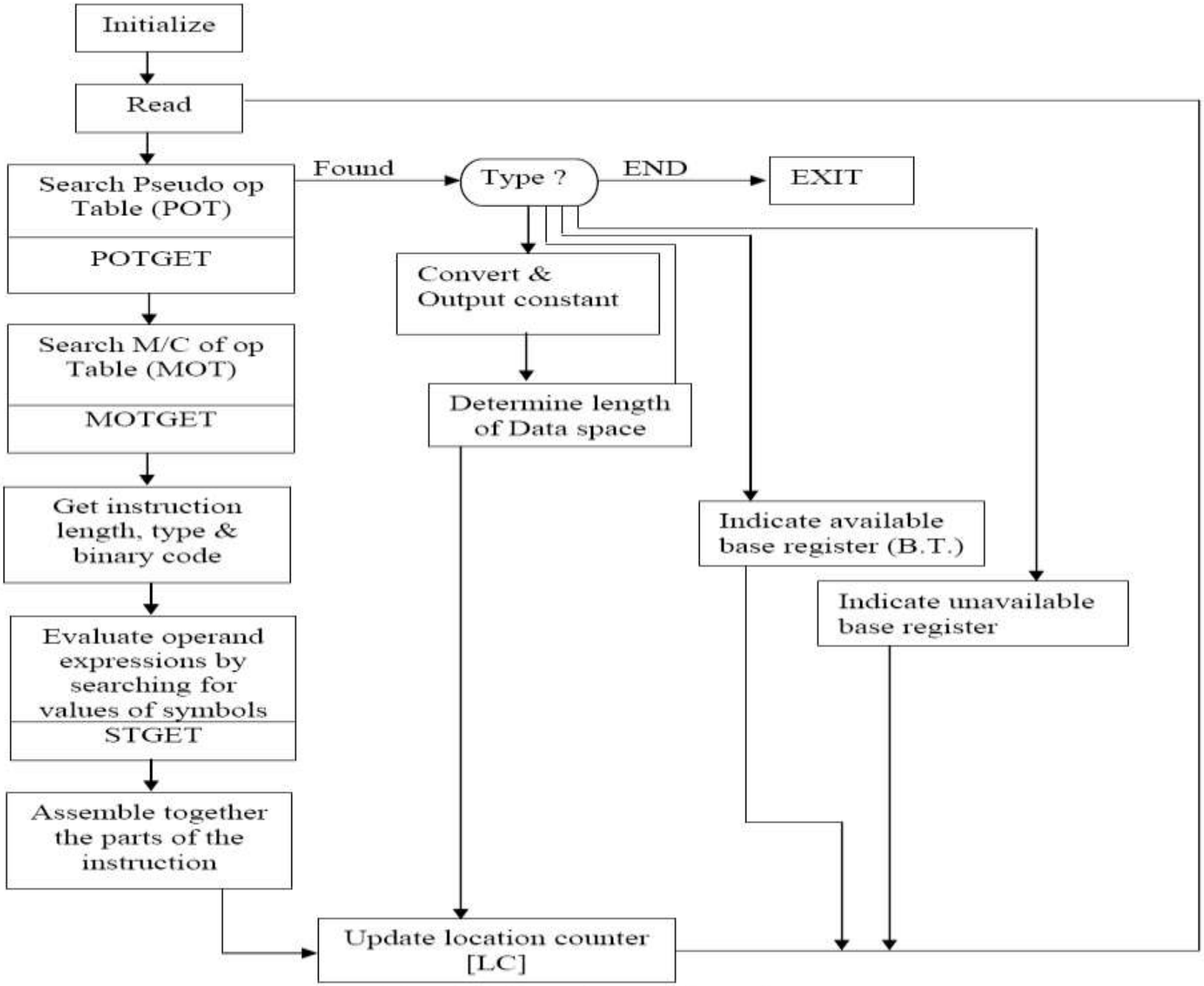


Fig. 1.5 Flow chart of Pass -2

Error reporting of assembler

- **Error reporting in pass I**
- Listing an error in first pass has the advantage that source program need not be preserved till pass II.
- But, listing produced in pass I can only reports certain errors not all.
- From the below program, error is detected at statement 9 and 21.
- Statement 9 gives invalid opcode error because MVER does not match with any mnemonics in OPTAB.
- Statement 21 gives duplicate definition error because entry of A is already exist in symbol table.
- Undefined symbol B at statement 10 is harder to detect during pass
- I, this error can be detected only after completing pass I

Error Table

Sr.no	Statements	Address
1	START 200	
2	MOVER AREG,A	200
3	. .	. .
9	MVER BREG, A	207
	**ERROR* Invalid opcode	
10	**ADD BREG, B**	208
14	A DS 1	209
. .	. .	. .
21	A DC '5'	227
	ERROR dulicate defination of symbol A	
	. .	
35	END	
	ERROR undefined symbol B in statememt 10	

Error reporting in pass II

- During pass II data structure like SYMTAB is available.
- Error indication at statement 10 is also easy because symbol table is searched for an entry B.
- if match is not found, error is reported

EXERCISE

Assembly Program to compute N! & write its intermediate code and machine code

1		START	101
2		READ	N
3		MOVER	BREG, ONE
4		MOVEM	BREG, TERM
5	AGAIN	MULT	BREG,TERM
6		MOVER	CREG, TERM
7		ADD	CREG, ONE
12		MOVEM	CREG, TERM
13		COMP	CREG, N
14		BC	LE, AGAIN
15		MOVEM	BREG, RESULT
16		PRINT	RESULT
17		STOP	
18	N	DS	1
19	RESULT	DS	1
20	ONE	DC	'1'
21	TERM	DS	1
22		END	