Simple Assembly Scheme.

Design specification of an assembler:-

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.

Synthesis phase → information requirements:-

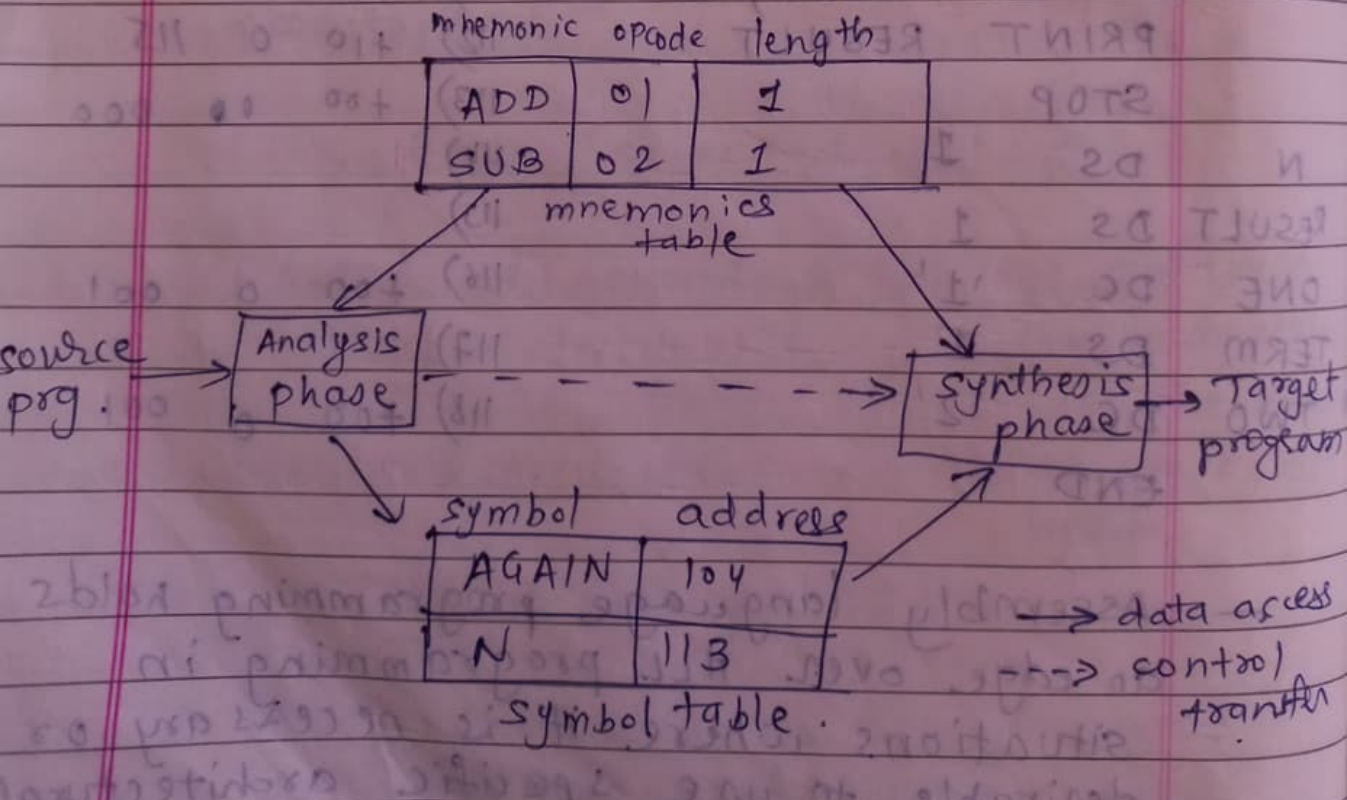


Fig:- Data structures of the assembler

⇒ Analysis Phase :-

- Analysis phase generates symbol table Hence, it must determine address of symbols used in the program.
- To determine the address of symbol N, it must find the address of all the instruction before N in the prg.
- This is called memory allocation.
- Location counter is used for memory allocation
- It contains the add. of next instruction
- whenever a label is found its entry is made in symbol and the contents of LC is also entered in S.T.
- LC is then updated by adding length of instn.
- length of instn depends upon assembly lang.
- mnemonic table has mnemonic code & length.
- Apart from the construction of S.T. & LC processing, analysis phase also checks the validity of mnemonics by refering to mnemonic table.
- It generates intermediate code which acts as i/p to synthesis phase.

⇒ Synthesis Phase :-

- Consider a statement,
  MOVER BREG, N.

- In order to synthesize the statement we must have 2 things.
1. Add. of instn associated with symbol N
2. m/c opcode correspondind to mnemonic MOVER.
- Add. of symbol depends on source prg. & it is made available by analysis phase
- m/c opcode can be determined by mnemonic table.
- Each entry of symbol table has name & add.
- mnemonic table has mnemonic, opcode, length.
- The synthesis phase uses the info. from these two tables to generate target program.

Pass Structure of Assemblers

→ Two pass translation:-

- It can handle forward reference easily
- LC processing is done in 1st pass & symbols entered in S.T.
- 2nd pass synthesizes the target form using the add. info. found in S.T.
- 1st pass = analysis of source prg.

- 2nd pass = synthesis of T.P.
- 1st pass constructs IR of S.P for use in 2nd pass.
- 2 components in fig.
  - data structures (eg. S.T & processed form of S.T
  - IC.

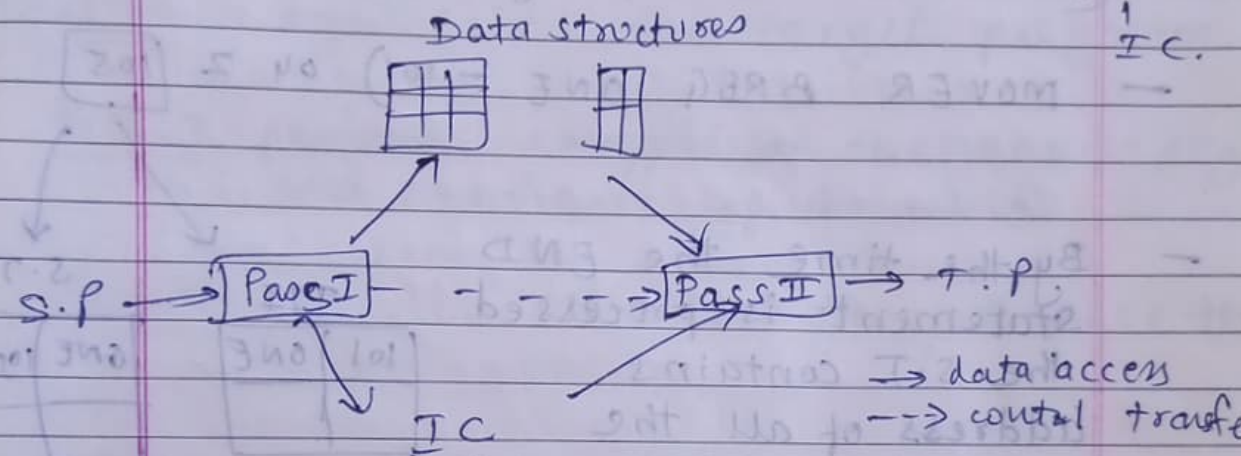


Fig:- overview of 2-pass assembly.

→ Single Pass translation.

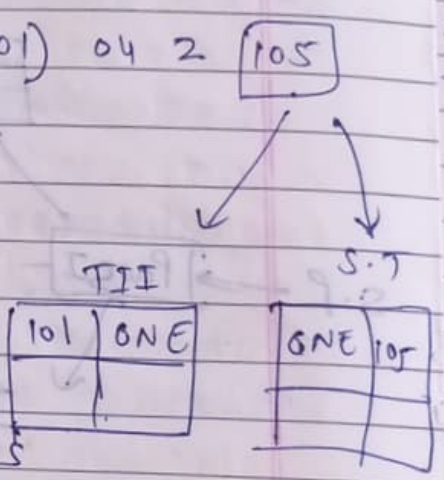
- LC processing and construction of the symbol table.
- problem of forward reference is tackled using backpatching.
- The operand field of an instn containing a forward reference is left blank initially.
- Add. of forward referenced symbol is put into the field when its definition is encountered.

ex:- MOVER BREG, ONE
can be partially synthesized since ONE is a forward reference.
- opcode & add. of BREG will be at 101
- Add. of second operand is added to Table of Incomplete Instructions (TII)

(<instruction add.>, <symbol>) (e.g)(101, ONE)

- MOVER BREG ONE - 101) 04 2 105

- By the time the END statement is processed the S.T contains address of all the symbols & TII contains info. of all the forward references.



eg:- Entry (101, ONE) would be processed by obtaining the address of ONE from symbol table & inserting it in the operand address field of the inst^n with incomplete address.

backpatching refers to process of resolving forward reference that have been placed in the code.

TII - identify the bytes in code where the add. of referenced symbol should be put.
- when the symbol's definition is encountered this entry could be analysed to complete the inst^n

SMVS
Page No.
Date : / /

Design of a Two Pass Assembler

Pass-I
1. Separate the symbol, mnemonic opcode & operand fields.
2. Build the symbol table
3. Perform LC processing
4. Construct intermediate representation

Pass-II synthesize the target program.

Pass-I performs analysis of the source prg. and synthesis of the IR.

Pass-II processed the IR to synthesize the target prg.

Advanced Assembler Directives.

ORIGIN

Syntax ORIGIN <address spec>

is an <operand spec>
or <constant>

- It indicates that LC should be set to the address given by <address spec>
- It is useful when the target program does not consist of consecutive memory words
- The ability to use an <operand spec> in ORIGIN statement provides the ability to perform LC processing in a relative

→ In the prg. ORIGIN LOOP+2, sets LC to add. 204, LOOP has 202 so MULT CREG, B will have address rather than absolute manner. 204 wo

→ ORIGIN LAST+1 sets LC to add. 217

Can also write
ORIGIN 202
ORIGIN 217
} absolute addressed.

EQU

Syntax <symbol> EQU <address spec>

where, <address spec> is an <operand>
or <constant>

- It associates the name symbol with address specification.
- It simply associates the name <symbol> with <address space>.

ex:- BACK EQU LOOP.

LTORG

- The LTORG statement permits the programmer to specify where the literals should be placed.
- By default the assembler places all the literals after the END statement.
- All the literals are allocated memory which is known as literal pool.

ex:-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)	+05 3	218
6		ADD	CREG, ='1'	204)	+01 3	212
7		---				

LT - less than.
GT - greater than.
EQ - equal.

SMVS
Page No.
Date : / /

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

- literals = '5' and '1' are added to literal pool in st-2 & 6.
- First LTORG allocates the addresses 211 and 212 to the values '5' and '1'.
- A new literal pool is now started.
- value '1' is put into pool in statement 15. This value is allocated to add. 219 while processing the END statement
- Literal = '1' used in st-15 refers to location 219 of the second pool of literals rather than location 212 of the first pool.
- All references to literals are forward references by definition.

વાંચવા-વિચારવા કરતાં વર્તનમાં ઉતારવું વધુ જરૂરી છે.

Pass-I of the assembler.

- Pass-I uses the following data structures

OPTAB - A table of mnemonic opcodes and related information.

mnemonic opcode	class	mnemonic info
MOVER	IS	(04,1)
DS	DL	R#7
START	AD	R#11
	⋮	

machine opcode
instn length.
id of a routine to handle the DL & AD.

OPTAB.

- The class field indicates whether opcode corresponds to an imperative statements (IS), a declaration statement (DL) or assembler directive (AD).

SYMTAB - Symbol Table.

symbol	address	length
LOOP	202	1
NEXT	214	1
LAST	216	1
A	217	1
BACK	202	1
B	218	1

SYMTAB.

- SYMTAB contains address and length.
- Processing of an assembly statement begins with the processing of its label field.
- If it contains symbol, the value & the LC is copied into new entry of SYMTAB.
- After that the functioning of PASS-I centers around the interpretation of OPTAB entry for mnemonic.
- If statement is imperative, the length of machine instruction is added to LC.
- Length is also entered in SYMTAB.
- For other statements the resp. routines are called. For DS → R#7 is called. and it will updates LC.

LITTAB — A table of literals used in the program.

- A LITTAB entry contains the fields literal and address.

	literal	address
1	='5'	
2	='1'	
3	='1'	

LITTAB.

- The first pass uses LITTAB to collect all literals used in the program.
- Awareness of different literal pools is maintained using the auxillary table POOLTAB

- This table contains the literal number of the starting literal of each literal pool.
- The current literal pool is the last pool in LITTAB.
- When LTORG or END statement occurs literals in the current pool are allocated addresses starting with the current value in LC and LC is incremented.
- In the prg. when LTORG occurs, first two literals will be given addresses 211 and 212.
- At END, the third literal will be given address 219.

Algo – (Assembler First Pass)

1. loc_cntr = 0 (default value)
   pooltab_ptr = 1, POOLTAB[1] = 1;
   littab_ptr = 1;
2. While next statement is not an END statement
   (a) If label is present then
   this_label = symbol in label field
   Enter (this_label, loc_cntr) in SYMTAB.
   (b) If an LTORG statement then
   (i) Process literals LITTAB[POOLTAB[pooltab_ptr]]... LITTAB[littab_ptr-1] to allocate memory & put the address in the address field.
   Update loc_cntr accordingly.

(ii) pooltab_ptr = pooltab_ptr + 1;
(iii) POOLTAB [pooltab_ptr] = littab_ptr;

(c) If a START or ORIGIN statement then
loc_cntr = value specified in operand field.

(d) If an EQU statement then.
(i) this_addr = value of <address spec>;
(ii) correct the symtab entry for this-label to (this_label, this_addr).

(e) If a declaration statement then
(i) code = code of the declaration statement
(ii) size = size of memory area required by DC/DS.
(iii) loc_cntr = loc_cntr + size;
(iv) Generate IC '(DL, code)...'

(f) If an imperative statement then
(i) code = machine opcode from OPTAB;
(ii) loc_cntr = loc_cntr + instn length from OPTAB

(iii) If operand is a literal then
this_literal = literal in operand field
LITTAB [littab_ptr] = this_literal;
littab_ptr = littab_ptr + 1;
]... else (i.e. operand is a symbol)
this_entry = SYMTAB entry no. of opera
Generate IC '(IS, code) (S, this_entry);

3. (Processing of END statement)

(a) Perform step 2(b).
(b) Generate IC '(AD,02)'.
(c) Go to Pass-II.

Intermediate Code Forms :

- Two criteria of choice of IR :

① processing efficiency
② memory economy.

- The IC consists of set of IC units, each IC unit consists of 3 fields.

1. Address.
2. Representation of the mnemonic opcode
3. Representation of operands.

Address	opcode	operands

Fig. An IC unit

Mnemonic Field

— Info. of this field remains same in all variants.
— The mnemonic field contains a pair of (statement class, code).

where Statement class can be one of IS, DL and AD.
- For IS code is the instn opcode in the machine language.
- For DL & AD code is an ordinal number within the class.

ex:- (AD,01) - means AD no. 1 which is START.

DS	AD
DC 01	START 01
DS 02	END 02
	ORIGIN 03
	EQU 04
	LTORG 05

Intermediate Code for Imperative Statements.

- There are 2 variants of IC which differ in information containing in their operand fields.
- Address field is identical in both variants.

Variant - I

- First operand — single digit number which is a code for register (1-4 for AREG - DREG). or condition code (1-6 for LT-ANY)

LT - 1
2
3
GT - 4
ગાંચવા-વિચારવા કરતાં વર્તનમાં ઉતારવું વધુ જરૂરી છે 5
ANY - 6

- Second operand - memory operand

(operand class, code).

C, S, L
- C → constant
- S → symbol
- L → 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 symbol or literal, the code field contains the ordinal number of the operand's entry in SYMTAB or LITTAB.
- Thus entries for a symbol XYZ and a literal = '25' would be (S, 17) & (L, 35) resp.

```
        START   200         (AD,01) (C,200)
        READ    A           (IS,09) (S,01)
LOOP    MOVER   AREG, A     (IS,04) (1) (S,01)
        ---                    :
        SUB     AREG, ='1'  (IS,02) (1) (L,01)
        BC      GT, LOOP    (IS,07) (4) (S,02)
        STOP                (IS,00)
A       DS      1           (DL,02) (C,1)
        LTORG               (AD,05)
        ---                    ---
```

Variant-II It differs from variant-I of the IC in that the operand fields of the source statements are selectively replaced by their processed forms.

- For DS and AD, processing of the operand fields is essential to support LC processing.
- Hence these fields contain the processed forms.
- For IS, the operand field is processed only to identify literal references.
- Literals are entered in LITTAB, and are represented as (L, m) in IC.
- symbolic references ~~are~~ in S.P. are not processed at all during Pass-I.

```
        START  200      (AD,01) (C,200)
        READ   A        (IS, 09)
LOOP    MOVER  AREG, A  (IS, 04) AREG, A
        :
        SUB    AREG, ='1'  (IS,02) AREG,(L, 01)
        BC     GT, LOOP    (IS,07)  GT, LOOP
        STOP               (IS, 00)
A       DS     1           (DL, 02) (C, 1)
        LTORG              (AD, 05)
        ----               ----
```

Comparision of the variants :-

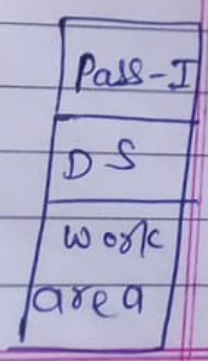
Variant - I

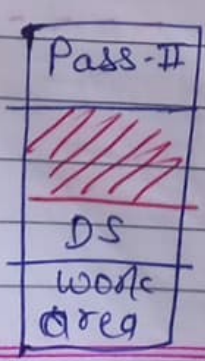
(1) Variant - I of the IC appears to require extra work in Pass-I since operand fields are completely processed.

(2) This processing simplifies the tasks of Pass-II.

(3) The IC is quite compact - it can be as compact as the target code itself if each operand reference like (S, n) can be represented in the same no. of bits as an operand address in a machine instruction.

(4) memory requirements





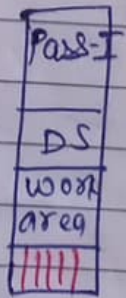
Variant - II

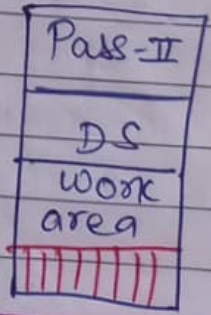
(1) Variant - II reduces the work of Pass - I by transferring the burden of operand processing from Pass - I to Pass - II of the assembler.

(2) This processing makes Pass - II to perform more work.

(3) The IC is less compact since the memory operand of a typical imperative stat is in the source form itself.

(4) memory requirements





- Pass-I performs much more processing than Pass-II, its code occupies more memory than the code of Pass-II.	- The code sizes of the two passes are now comparable, hence the overall memory requirement of the assembler is lower.

* Processing of Declarations and Assembler Directives :-

- We need to find alternative way of processing DS & AD. How are they processed in Pass-I.

1. Is it necessary to represent the address of each source statement in IC?
2. Is it necessary to have an explicit representation of DS statements & assembler directives in IC?

Ans :- YES.

ex :-

```
    START  200                 ⟶ (AD,01) (C,200)
AREA1  DS     20      ⇒    200) (DL,02) (C,20)
SIZE   DC     5           220) (DL,01) (C,5)
```

- It is not necessary to have a representation for DS & AD in IC if the IC contains an address field.

- If address field is omitted, representation of DS & AD is essential.

- Pass-II can determine the address for SIZE only after analyzing the IC units for the START and DS.

DC statement.

- A DC statement must be in IC. the mnemonic field contains the pair (DL, 01).
- operand field has constant in source form
- No processing advantage exists in either case since conversion of the constant into machine code is required anyway.

DC '5, 3, -7'.

a series of (DL, 01) units can be put in the IC.

START & ORIGIN.

- These directives set new values into the LC. It is not necessary to retain START and ORIGIN statements in the IC. if the IC contains an address field.

LTORG

- Pass-I checks for the presence of a literal reference in the operand field of every statement.
- If one exists, it enters the literal in the current literal pool in LITTAB.
- When LTORG statement appears in S.P, it assigns memory addresses to the literals in the current pool.
- These addresses are entered in the address field of their LITTAB entries.
- Pass-I could simply construct an IC unit for LTORG statement and leave all subsequent processing to Pass-II.
- values of literals can be inserted in the target program when this IC unit is processed in Pass-II.
- This requires the use of POOLTAB and LITTAB in a manner analogous to Pass-I

ex:- IC for the first half of earlier program

	START 200	(AD,01) (C,200)
	MOVER AREG, ='5'	(IS,04) (1) (L,01)
	MOVEM AREG, A	(IS,05) (1) (S,01)
LOOP	MOVER AREG, A	(IS,04) (1) (S,01)
	BC ANY, NEXT	(IS,07) (6) (S,04)
	LTORG	(DL,01) (C,5)
		(DL,01) (C,1)

- This increases the task of Pass-I & size increases.
- Literals have to exists at 2 places LITTAB and IC both.

Pass-II of the Assembler

Algo :-

1. code_area_address (area of target code.) = address of code-a
   pooltab_ptr = 1;
   loc_cntr = 0;

2. While next statement is not an END statement.

   (a) Clear machine_code_buffer;
   (b) If an LTORG statement.

   (i) Process literals in LITTAB[POOLTAB[pooltab_ptr]] to LITTAB[POOLTAB[pooltab_ptr+1]]-1 similar to processing of constants in a DC statement. i.e, assemble the literals in machine_code_buffer.

   (ii) size = size of memory area required for literals.

   (iii) pooltab_ptr = pooltab_ptr + 1;

(c) If a START or ORIGIN statement then.

(i) loc_cntr = value specified in operand field.

(ii) size = 0;

(d) If a declaration statement.

(i) If a DC statement then
Assemble the constant in machine-code_buffer.

(ii) size = Size of memory area required by DC/DS.

(e) If an imperative statement.

(i) Get operand address from SYMTAB or LITTAB.

(ii) Assemble instruction in machine_code_buffer.

(iii) size = size of instruction;

(f) If size ≠ 0 then

(i) Move contents of machine_code_buffer to the address code_area_address + loc_cntr;

(ii) loc_cntr = loc_cntr + size;

SMVS
Page No.
Date : / /

3. (Processing of END statement).

(a) Perform step 2(b) and 2(f)

(b) write code-area into output file.

- The assembler produces an object modul in the format required by a linkage editor or loader.
- Assembler produces a target program which is the machine language of the target computer.