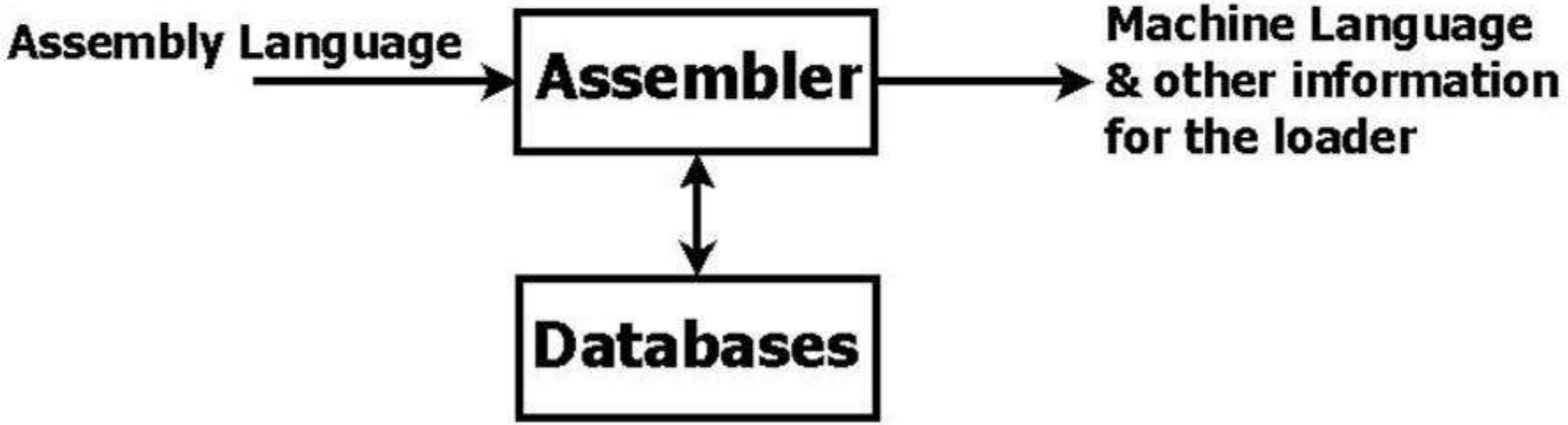
# Assembler- an overview

Assembler accepts ALP as input & produces its machine language equivalent along with other information for the loader ( like externally defined symbols)



# Types of ALP statement

1. Assembler Directives (Pseudo op instruction)
   - Directs the assembler to take action
   - Non executable part of the program
   - Not included in the final object code
   - Examples: ASSUME, USING, START, END etc.
2. Declarative statements
   - Declare symbols & assign values to them
   - Example: First DC F'4'
3. Imperative statements
   - Executable assembly language instructions
   - Assembler translates them into object code
   - Examples: L, A, ST instruction

# Design of assembler- Statement of Problem

Consider ALP of IBM 360/370 machine for addition of 2 numbers/constants:

```
TILAK    START    0
         USING    *, 15
         L        1, First
         A        1, Second
         ST       1, Result
First    DC       F '4'
Second   DC       F '5'
Result   DS       1F
         END
```

# Description of the statement of problem

- START – Indicates to the assembler the start of the ALP
- TILAK – Name of the ALP (assembler passes it to the loader)
- USING- States that register 15 is the base register
- L (Load), A (Add) & ST (Store) are the assembly language instructions using register1 as an accumulator
- L,A & ST are 32 bit instructions (requires 4 locations each for storage)
- DC F'4': Define Constant of full word size & its value = 4
- DS 1F: Define Storage of 1 full word for copying result
- Full word = 32 bit in IBM 360 machine
- END – End of the ALP

# Design of assembler- Statement of Problem

Consider ALP of IBM 360/370 machine for addition of 2 numbers/constants:

```
TILAK    START    0
         USING    *, 15
         L        1, First
         A        1, Second
         ST       1, Result
First    DC       F '4'
Second   DC       F '5'
Result   DS       1F
         END
```

# Pass-1 operations for given problem

| Pass-1 operation of assembly process | |
|---|---|
| Relative address | Mnemonic instructions |
| 0 | L 1, - (0,15) |
| 4 | A 1, - (0,15) |
| 8 | ST 1, - (0,15) |
| 12 | 4 |
| 16 | 5 |
| 20 | - |

# Pass-2 operations for given problem

| Pass-2 operation of assembly process | |
|---|---|
| Relative address | Mnemonic instructions |
| 0 | L 1, 12 (0,15) |
| 4 | A 1, 16 (0,15) |
| 8 | ST 1, 20 (0,15) |
| 12 | 4 |
| 16 | 5 |
| 20 | - |

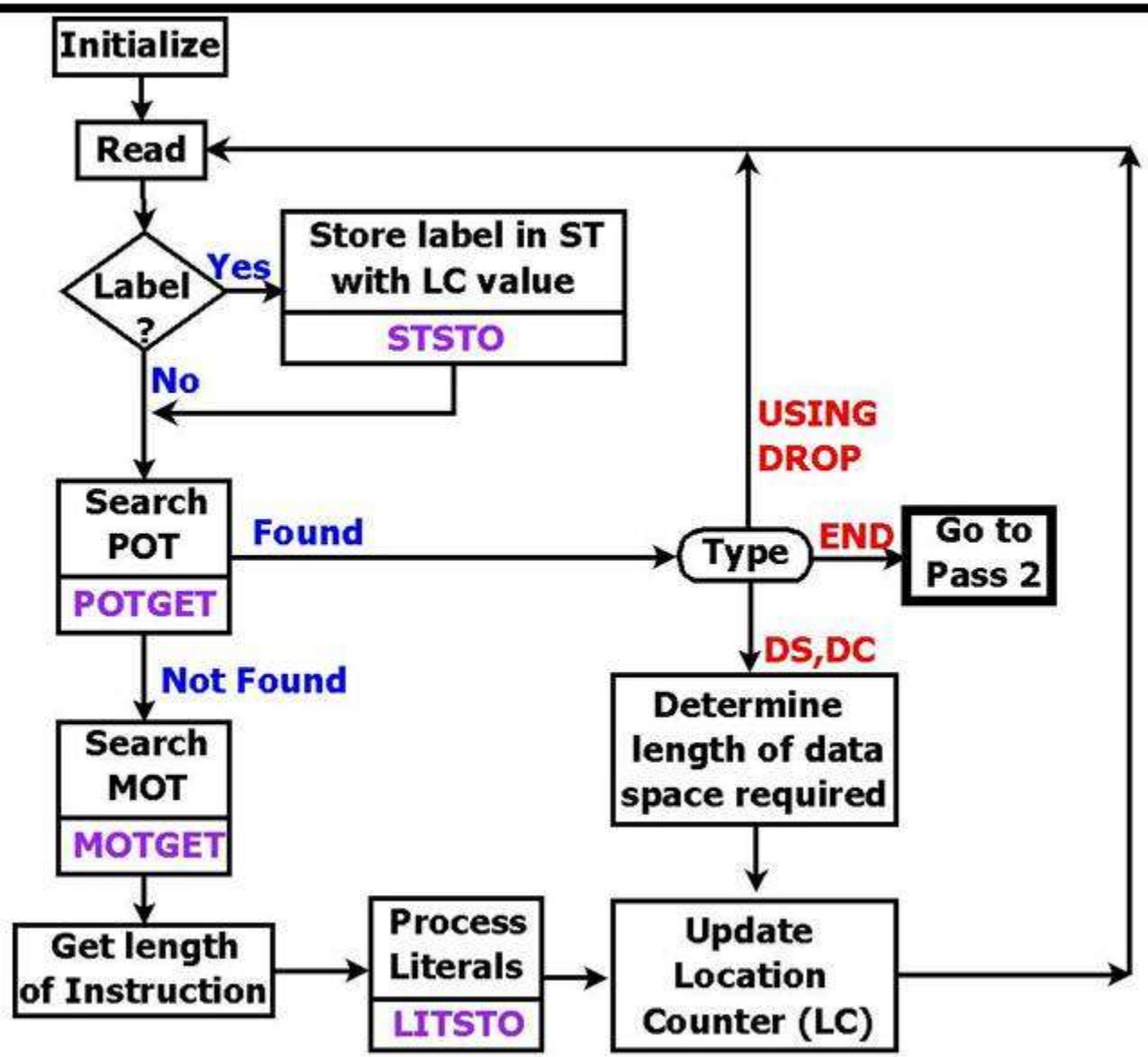
# Functions of Pass-1 of the assembler

1. Determine length of machine instructions (MOTGET1)
2. Keep track of Location Counter (LC)
3. Remember values of symbols until Pass-2 (STSTO)
4. Process some pseudo-ops like DS, DC (POTGET1)
5. Remember literals (LITSTO)

In brief,

Pass-1 defines symbols & literals

# Pass-1 of assembler: An overview



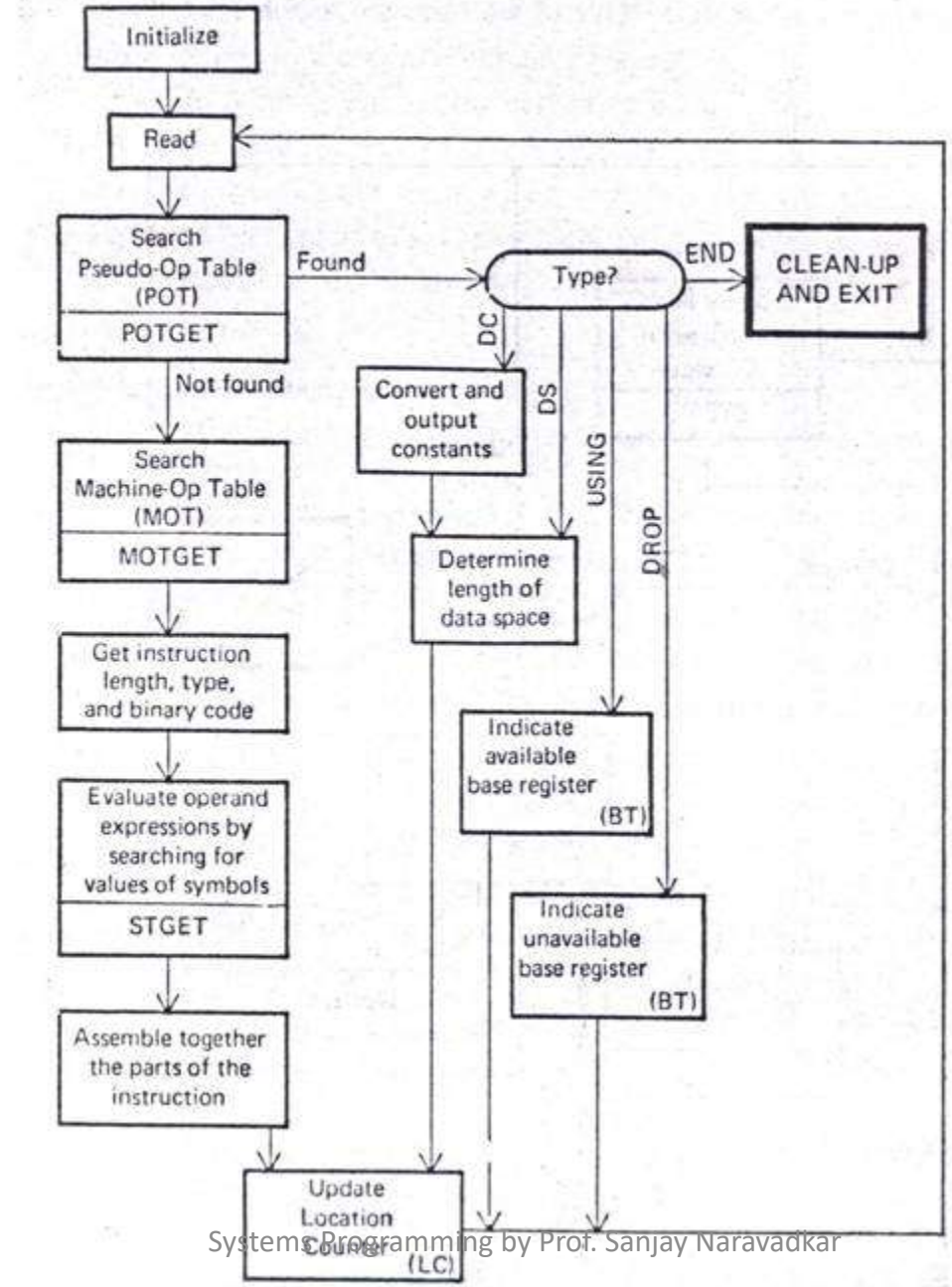
# Functions of Pass-2 of the assembler

1. Look up value of symbols (STGET)
2. Generate object code (MOTGET2)
3. Generate data (for DS, DC & literals)
4. Process Pseudo-ops (POTGET2)

In brief,

Pass-2 generates the object program

# Pass-2 of assembler: An overview

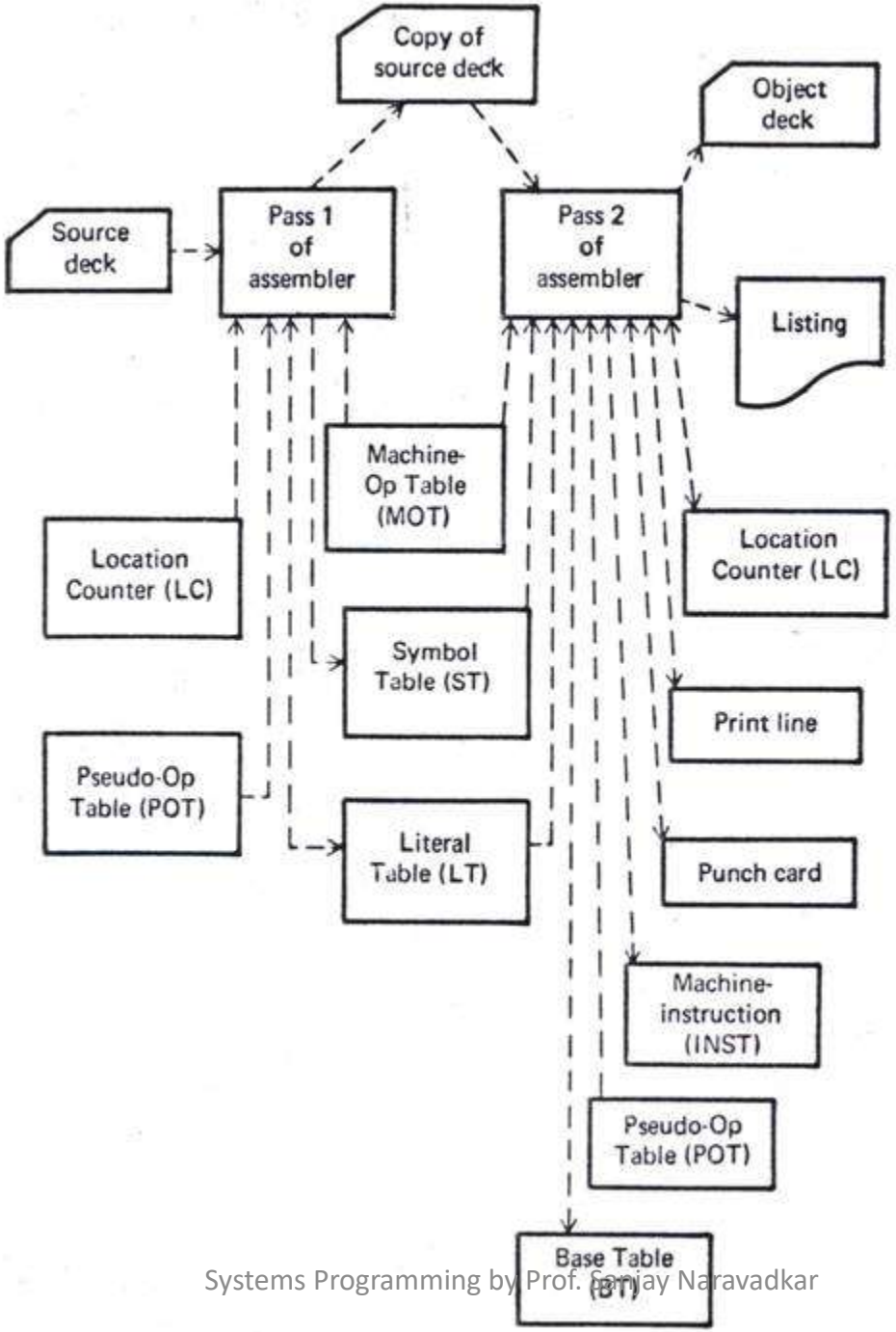


# Data bases/structures used by assembler phases

- Input source program
- Location Counter (LC) - used to keep track of each instruction's location
- Machine Opcode Table (MOT)- define opcode & size of each instruction (2/4/6 bytes)
- Pseudo Opcode Table (POT)- define assembler directives & action to be taken
- Symbol Table (ST)- store each label & its value
- Literal Table (LT)- Store each literal & its assigned location

## Additional data structures in pass-2:

- Base Table (BT)- Indicates availability of base register
- Work-space INST- Hold various parts of each instruction like binary opcode, register fields, length, displacements etc.
- Work-space PRINT LINE – used to produce a printed listing
- Work-space PUNCH CARD – used to produce a punched card


Copy of source deck
Object deck
Source deck
Pass 1 of assembler
Pass 2 of assembler
Listing
Machine-Op Table (MOT)
Location Counter (LC)
Location Counter (LC)
Symbol Table (ST)
Print line
Pseudo-Op Table (POT)
Literal Table (LT)
Punch card
Machine-instruction (INST)
Pseudo-Op Table (POT)
Base Table (BT)

# Format of MOT (IBM 360 machine)

- Fixed sized table
- Contents are not filled or altered during assembly process
- Number of entries = number of assembly language instructions
- Size of each entry of MOT = 6 bytes

| Mnemonic opcode (4 bytes) (characters) | Binary opcode (1 byte) (Hex) | Instruction length (2 bits) (binary) | Instruction format (3 bits) (Binary) | Not used in this design (3 bits) |
|---|---|---|---|---|
| | | | | |

# Format of POT (IBM 360 machine)

- Also fixed sized table like MOT
- Pseudo-op = Assembler directive
- Pseudo-op is standard label (USING, EQU, END etc)
- Each pseudo-op is listed with an associated pointer (24 bit physical address of the AL program)
- Size of each entry of POT = 8 bytes

| Pseudo-op (5-bytes) (Characters) | Address of routine to process Pseudo-op ( 24 bits) |
|---|---|
| | |

# Format of Symbol Table (ST) & Literal Table (LT)

- Lists all the symbols in ST & literals in LT
- Same format for ST & LT
- Each entry of ST/LT = 14 bytes
- Symbol table for the problem taken is:

| Symbol (8 bytes) (characters) | Value (4 bytes) (Hex) | Length (1 byte) (Hex) | Relative/ Absolute (1 byte) (Character) |
|---|---|---|---|
| TILAKbbb | 0000 | 01 | "R" |
| Firstbbb | 000C | 04 | "R" |
| Secondbb | 0010 | 04 | "R" |
| Resultbb | 0014 | 04 | "R" |

# Format of Base Table (BT) for Pass-2

- BT gives availability/unavailability of base register
- Total 15 entries in BT (one each for base register)
- Each entry = 4 bytes

| Availabilty indicator (1 byte) (Character) | Designated relative address (Contents of base register) (24 bits) |
|---|---|
| "N" | ------ |
| "Y" | 00 00 00 |

- Y = Register available as base register by USING pesudo-op
- N= Register unavailable as base register by DROP pesudo-op

# Detailed Algorithms of Assemblers

## Pass-1 Algorithm:

Page 74/system programming / JOHN J. DONOVAN

## Pass-2 Algorithm:

Page 75/system programming / JOHN J. DONOVAN

# Pass-1 algorithm

1. Initialize Location Counter LC = 0
2. Read card pointed by LC
3. Search POT for particular Pseudo-op
   a) If DC or DS, then update LC to proper value, Goto step 5
   b) If EQU, then evaluate operand field & assign value to the label field, Goto step 7
   c) If USING or DROP, then Goto step 7
   d) If END, then assign storage for literals, rewind & reset copy
      Goto Pass-2
4. Search MOT
   a) Obtain instruction length L from MOT
   b) Process literals (if present) & enter them to LTSTO
5. If instruction has label, then assign value of LC to symbol in STSTO
6. Update LC i.e. LC = LC + L
7. Write copy of card on file for use by Pass-2
   Goto step 2

# Pass-2 algorithm

1. Initialize Location Counter LC = 0
2. Read card pointed by LC
3. Search POT for particular Pseudo-op
   a) If DC, update LC to proper value
   b) If DS, define constant & insert in assembled program, Goto step 5
   c) If EQU, then Print listing, Goto step 2
   d) If USING, define base reg & value in BT, Print listing & Goto step-2
   e) If DROP, define base reg not available, Print listing & Goto step-2
   f) If END, then generate literals in literal table (LTGEN), STOP
4. Search MOT for Instruction type, Op-code & length (L)
   a) If type = SI or RR, then evaluate registers (both) & enter into 2nd byte, Goto step 5
   b) If type = RX, then evaluate register & index expression & enter in 2nd byte

   Compute Eff. Address (EA) = Base reg. contents (B) + Displacement (D)

   Record B & D into 3rd & 4th byte
5. Punch & print assembly listing line

   Update LC i.e. LC = LC + L

   Goto step 2

## Look for modularity: Modular design of Assembler

- Assembler is a coordinated collection of number of subroutines/functions each of relatively small size & complexity (instead of single program of thousands of source statements)
- Typical functions in assembler passes:

  MOTGET1, MOTGET2, POTGET1, POTGET2, STSTO, READ1, READ2, STGET etc.
- Being smaller sizes of functional modules:
  - Small software development lifecycle
  - Simpler debugging

# Assembler Pass-1 functional routines

| Function | Operation |
|---|---|
| READ1 | Reads source assembly card |
| POTGET1 | Search POT for a match with that of source card |
| MOTGET1 | Search MOT for a match with that of source card |
| STSTO | Store a label & its associated value in Symbol Table (ST) |
| LTSTO | Store a literal in Literal Table (LT) |
| WRITE1 | Write a copy of source on storage device for use in pass-2 |
| DLENGTH | Determine storage space required in DS & DC pseudo-ops |
| EVAL | Evaluate expression having constants & symbols |
| STGET | Search ST for specific symbol |
| LITASS | Assign storage locations to each literal in the LT |

# Assembler Pass-2 functional routines

| Function | Operation |
|---|---|
| READ2 | Reads source card from the file copy |
| POTGET2 | Same as in pass 1 |
| MOTGET2 | Same as in pass 1 |
| EVAL | Same as in pass 1 |
| PUNCH | Punch the card with the object code |
| PRINT | Print relative location & generated object code |
| DCGEN | Process the field of DC pseudo-op to generate object code |
| DLENGTH | Same as in pass 1 |
| BTSTO | Insert data into appropriate entry of Base Table (BT) |
| BTDROP | Insert "unavailable" indicator into proper BT entry |
| BTGET | Search BT for available base registers for calculation of effective address |
| LTGEN | Generate code for literals |

# Drawback of single-pass assembler

- In Single pass assembler: Forward Reference problem
- **Forward reference problem:**
  - Refers symbols/variable before their declaration
  - Such symbol can not be recognized
  - In earlier example, First, Second & Result are address symbols referred in ALP before declaration
  - Hence, 2-pass assembler is designed….