
# System Programming & Compiler Construction

## Unit 2: System Programming:- Assembler

Faculty Name : Tabassum Maktum
Ekta Sarda

# Index -

# Lecture No: 25
# Two pass assembler Design :PASS2

## Databases used by Pass 2

SYMTAB, LITTAB and POOLTAB

| | | |
|---|---|---|
| LC | : | Location counter |
| *littab_ptr* | : | Points to an entry in LITTAB |
| *pooltab_ptr* | : | Points to an entry in POOLTAB |
| *machine_code_buffer* | : | Area for constructing code for one statement |
| *code_area* | : | Area for assembling the target program |
| *code_area_address* | : | Contains address of *code_area* |

- Label definitions have been handled in Pass 1. So, they are ignored in Pass 2.
- **Imperative Statement:**
- The mnemonic is searched in OP Table: its equivalent binary opcode is read and written into output file.
- LC is incremented by length of instruction.
- **Operand**
- If operand is a literal, its address is taken from LT and written into output file.
- If operand a symbol, its address is taken from ST and written in output file.
- Finally, LC is incremented by length of instruction.

## Algorithm: Pass 2

1. *code_area_address* := address of *code_area*;
   *pooltab_ptr* := 1;
   LC := 0;
2. While the next statement is not an END statement
   (a) Clear *machine_code_buffer*;
   (b) If an LTORG statement
      (i) If POOLTAB [*pooltab_ptr*]. *# literals* > 0 then
         Process literals in the entries LITTAB [POOLTAB [*pooltab_ptr*] .*first*] ... LITTAB [POOLTAB [*pooltab_ptr*+1]−1] similar to processing of constants in a DC statement. It results in assembling the literals in *machine_code_buffer*.
      (ii) *size* := size of memory area required for literals;
      (iii) *pooltab_ptr* := *pooltab_ptr* + 1;

## Algorithm: Pass 2

(c) If a START or ORIGIN statement

(i) LC := 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 the memory area required by the declaration statement;

(e) If an imperative statement

(i) Get address of the operand from its entry in SYMTAB or LITTAB, as the case may be.

(ii) Assemble the instruction in *machine_code_buffer*.

(iii) *size* := size of the instruction;

## Algorithm: Pass 2

(f) If *size* ≠ 0 then

(i) Move contents of *machine_code_buffer* to the memory word with the address *code_area_address* + <LC>;

(ii) LC := LC + *size*;

3. (Processing of the END statement)

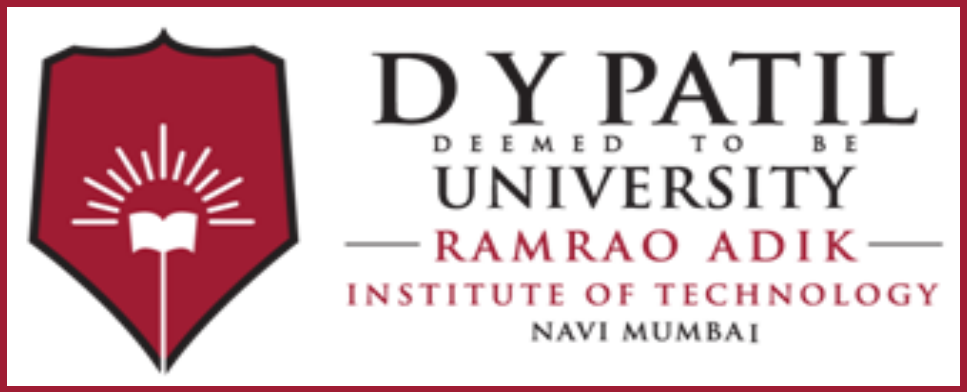
(a) Perform actions (i)–(iii) of Step 2(b).

(b) Perform actions (i)–(ii) of Step 2(f).

(c) Write *code_area* into the output file.

# Example 1: Output of Pass 2

| LnO. | Label | OPCODE | OP1 | OP2 | LC | Intermediate Code | Machine Code |
|---|---|---|---|---|---|---|---|
| 1 | | START | 400 | | 400 | (AD, 01) (C,400) | ---- |
| 2 | | MOVER | AREG | ='5' | 400 | (IS,04) (RG,01) (L,0) | (04) (01) (405) |
| 3 | | MOVER | BREG | A | 401 | (IS,04) (RG,02) (S,0) | (04) (02) (407) |
| 4 | X | MOVER | CREG | ='3' | 402 | (S,1) (IS,04) (RG,01) (L,1) | (402)(04) (03) (406) |
| 5 | | ORIGIN | X+3 | | 403 | (AD,03) (C,405) | ---- |
| 6 | | LTORG | | | 405 | **(AD,05) (DL,02)(C,5) (AD,05) (DL,02) (C,3)** | **(00)(00)(5) (00)(00)(3)** |
| 7 | A | DS | 3 | | 407 | (S,0) (DL,01) (C,3) | - |
| 8 | | ADD | AREG | B | 410 | (IS,04) (RG,01) (S,2) | 04) (01) (413) |
| 9 | B | DC | '4' | | 411 | (S,2) (DL,02) (C,4) | ---- |
| 10 | C | DC | '6' | | 412 | (S,3) (DL,02) (C,6) | ---- |
| 11 | Y | EQU | '7' | | 413 | (S,4) (AD,04) (C,7) | ---- |
| 12 | | ADD | CREG | Y | 414 | (IS,04) (RG,03) (S,4) | (04) (03) (7) |
| 13 | | END | | | 414 | (AD,02) | ---- |


D Y PATIL
DEEMED TO BE
UNIVERSITY
RAMRAO ADIK
INSTITUTE OF TECHNOLOGY
NAVI MUMBAI

# Thank You

# Lecture No: 26
# Introduction to single pass assembler design

## SINGLE PASS ASSEMBLER for IBM PC

- Forward Reference Problem (FRP) occurs when a symbol is referenced before it gets defined.
- Due to this, the assembler cannot assemble the instruction right at the time when it gets encountered.
- To resolve FRP, we used Multi-pass approach which collects all symbol definitions in Pass 1 and then assembles all instructions in Pass 2.
- But it requires an extra pass to handle forward-referenced symbols, which is inefficient.

## SINGLE PASS ASSEMBLER

- A Single-pass Assembler solves FRP efficiently in one pass. It uses a special data structure called the Forward Reference Table (FRT) and assembles all the instructions completely, right at the time when they are encountered.

- If the instruction involves a forward-referenced symbol (that is not yet defined), such symbols are entered into FRT.

- At the end of pass, when all symbol definitions have been collected in ST, the assembler uses ST to update definition addresses of forward-referenced symbols in FRT.

- It then uses FRT to update the usage locations of forward-referenced symbols in output file. Thus, a single-pass assembler handles forward reference symbols efficiently.

## SINGLE PASS ASSEMBLER

- Single pass assembler for the Intel 8088 processor used in IBM PC.

- Focuses on the design features for handling the forward reference problem in an environment using segment-based addressing.

- Memory

| Units of Memory | Bytes | Length in bits |
|---|---|---|
| **Byte** | **1** | **8** |
| **Word** | **2** | **16** |
| **Double Word** | **4** | **32** |
| **Quadra Word** | **8** | **64** |
| **Tetra Word** | **10** | **80** |

D Y PATIL DEEMED TO BE UNIVERSITY — RAMRAO ADIK — INSTITUTE OF TECHNOLOGY NAVI MUMBAI

# A Single Pass Assembler for IBM PC

- A single pass assembler scans the program **only once** and creates the equivalent binary program.
- The assembler substitute all of the symbolic instruction with machine code in one pass.
- Single pass assembler for the Intel 8088 processor used in IBM PC.
- Focuses on the design features for handling the forward reference problem in an environment using segment-based addressing.

- The CPU contains following features –
    1. Data registers AX, BX, CX and DX
    2. Index registers SI and DI
    3. Stack pointer registers BP and SP
    4. Segment registers Code, Stack, Data and Extra

a)

| AH | AL |
|---|---|
| BH | BL |
| CH | CL |
| DH | DL |

b)

| BP |
|---|
| SP |

c)

| SI |
|---|
| DI |

d)

| Code |
|---|
| Stack |
| Data |
| Extra |

Fig:- a) Data b) Base
c) Index
d) Segment registers

## Elements of IBM PC -8088 Architecture

- ✓ Each data register is 16 bits in size, split into upper and lower halves.
- ✓ Either half can be used for 8 bit arithmetic, while the two halves together constitute the data register for 16 bit arithmetic.
- ✓ Architecture supports stacks for storing subroutine and interrupt return addresses, parameters and other data.
- ✓ The index registers SI and DI are used to index the source and destination addresses in string manipulation instructions.
- ✓ Two stack pointer registers called SP and BP are provided to address the stack. Push and Pop instructions are provided.

## continued..

- ✓ The Intel 8088 provides addressing capability for 1 MB of primary memory.
- ✓ The memory is used to store three components of program, Program code, Data and Stack.
- ✓ The Code, Stack and Data segment registers are used to contain the start addresses of these three components.
- ✓ The Extra segment register points to another memory area which can be used to store data.
- ✓ The size of each segment is limited to 216 i.e 64 K bytes.

# 3. The Assembly Language of Intel 8088

1) Statement Format

[Label:] opcode operand(s) ; comment string

2) Assembler Directives

a) Declarations

- Declaration of constants and reservation of storage are both achieved in the same direction

```
A   DB   25    ; Reserve byte & initialize 25
B   DW   ?     ; Reserve word, no initialization
C   DD  6DUP(0)   ; 6 Double words, all 0's
```

## continued..

### b) EQU and PURGE

**EQU** defines symbolic names to represent values

**PURGE** undefined the symbolic names. That name can be reused for other purpose later in the program.

```
Example:-    XYZ   DB     ?
             ABC   EQU   XYZ      ; ABC represents name XYZ
              PURGE   ABC  ;        ABC no longer XYZ
             ABC   EQU    25       ; ABC now stands for '25'
```

**ASSUME :** This directive tells the assembler which segment register contains the segment base.

**ASSUME <register> :<segment name>**

**SEGMENT :** It indicates start of segment

**ENDS** : It indicates end of segment

# Lecture No: 27
# Data Structure used in single pass assembler design

## Databases required

1. Source program
2. **Mnemonic Operation Table** (MOT). This table indicates the symbolic mnemonic for each instruction.
3. **Symbol Table** (ST) which stores each label along with its relevant information.
4. **Segment Register Table** (SRTAB) which stores information about segment name and segment register.
5. **Forward Reference Table** (FRT) which stores information about forward references.
6. **Cross reference table** (CRT) which list out all references to a symbol in ascending order of statements.

# MOT

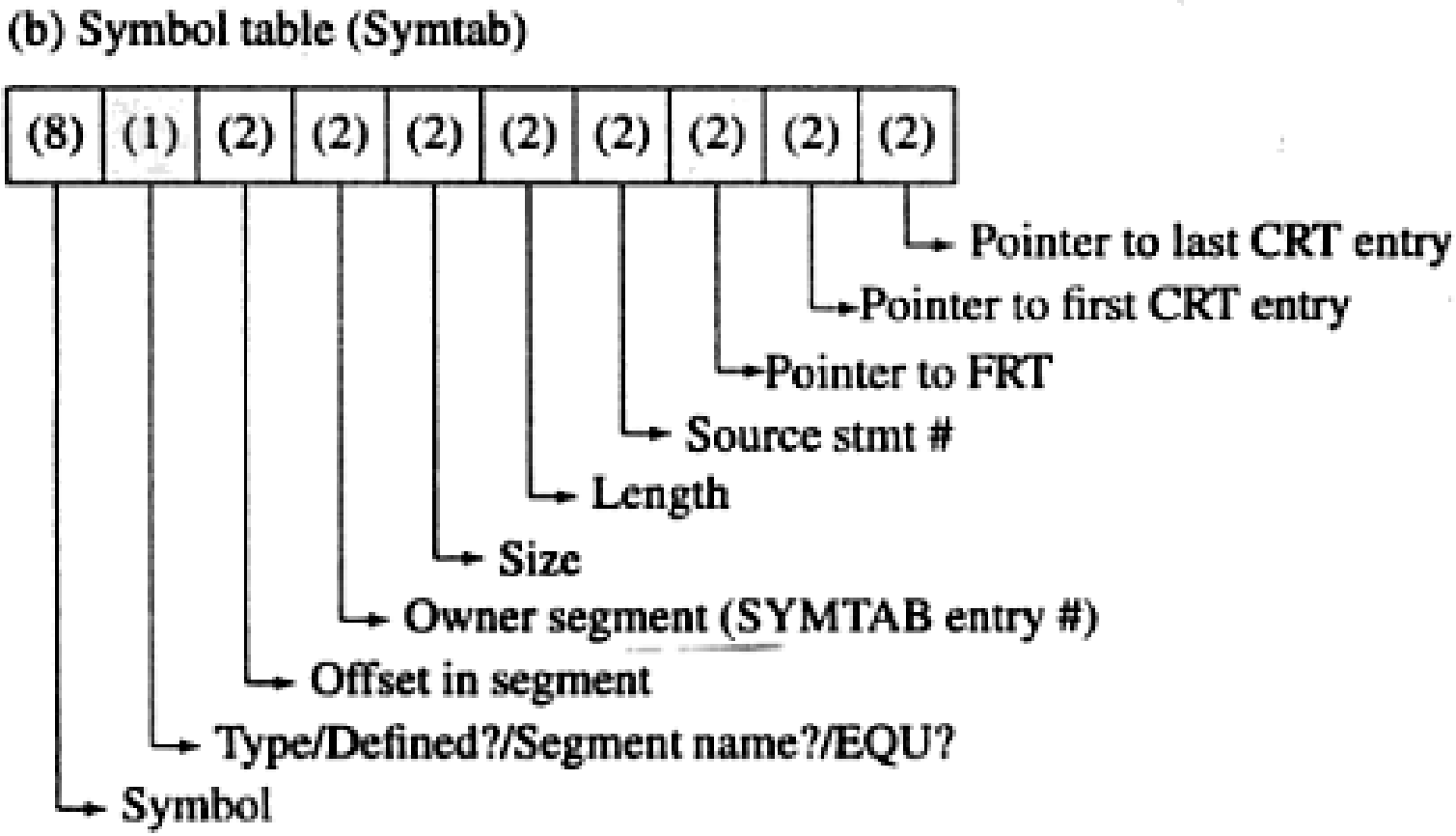
- **Mnemonic Table (MOT)**

| Mnemonic op-codes (6) | Machine op-codes (2) | Alignment/format information (1) | Routine id (4) binary |
|---|---|---|---|
| JNE | 75 H | 00H | R2 |
| | | | |
| …… | ……. | …… | …… |
| | | | |
| | | | |

## MOT table (Mnemonic Opcode Table)

- It contains field Mnemonic opcode, machine opcode, alignment/format info and routine id.
- The Routine id filed of an entry specifies the routine which processes the opcode.
- Alignment /format info is specific to given routine.
  - for example the code of '00H' for routine R2 implies that only one instruction format is supported (Self Relative displacement instruction).
  - FFH for same routine implies that all the formats are supported, hence routine must decide which machine opcode to use.

# Symbol Tabel



## Symbol Table (ST )

- The SYMTAB (symbol table) is also a hash organized and contains all relevant information about symbols defined and used in the source program.
- Contents of some important fields are:

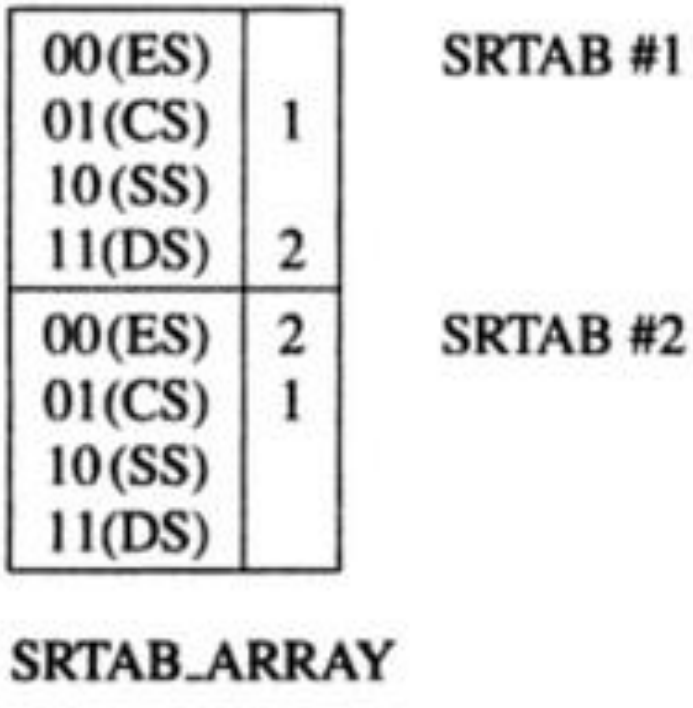
  a) the owner segment field : Indicates id of segment in which segment is defined.

  b) Type/Defined/Segment name?EQU : for non EQU symbol the type field indicates the alignment information. For EQU symbol , type field indicates whether the symbol is to be given a numeric value or texual value.

  c) Offset in segment : contains offset value.

## SRTAB (segment Register table)

- An SRTAB can contains up to four entries, one for each register.
- The current SRTAB exists in the last entry of SRTAB_ARRAY.SRTAB
- Example :



## Forward Reference Table (FRT)

| Pointer (2) | SRTAB # (1) | Instruction Address (2) | Usage Code (1) | Source statement # (2) |
|---|---|---|---|---|
| | | | | |

## Forward reference table (FRT)

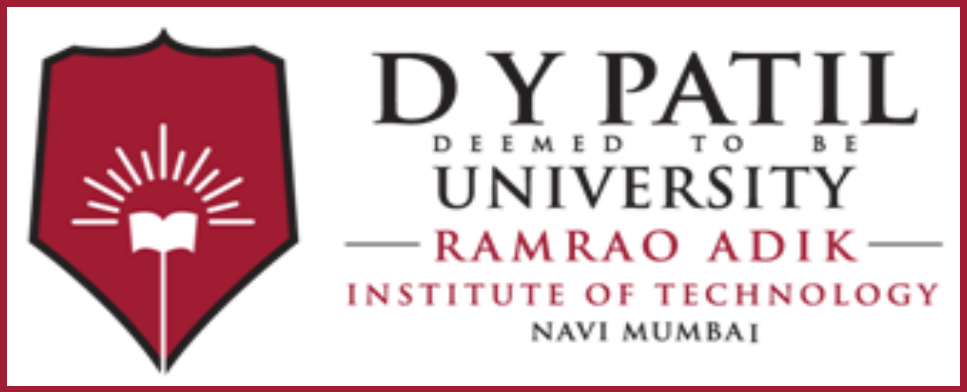
- Information  concerning forward references to symbol is organized in the form of linked list., Thus forward reference table contains a set of linked lists.
- The FRT pointer field of symbol table entry points to the head of this linked list.
- Each FRT entry contains SRTAB# to be used to assemble the forward reference.
- Usage field of FRT entries what information is required in the referencing instruction.  e.g. Data address (D), self relative address('S'), length (L), Offset (F)

## Cross reference table (CRT)

| Pointer to next entry (2) | Source statement # (2) |
|---|---|
| | |

## CROSS REFERENCE TABLE (CRT )

- A cross reference directory is a report produced by the assembler which lists all references to symbol sorted in ascending order of statement numbers.
- Assembler uses CRT to collect information concerning references to all symbols in the program.
- Each symbol table entry points to the head and tail of the linked list in the CRT. CRT and FRT can be organized in to single memory area


D Y PATIL
DEEMED TO BE
UNIVERSITY
RAMRAO ADIK
INSTITUTE OF TECHNOLOGY
NAVI MUMBAI

# Thank You