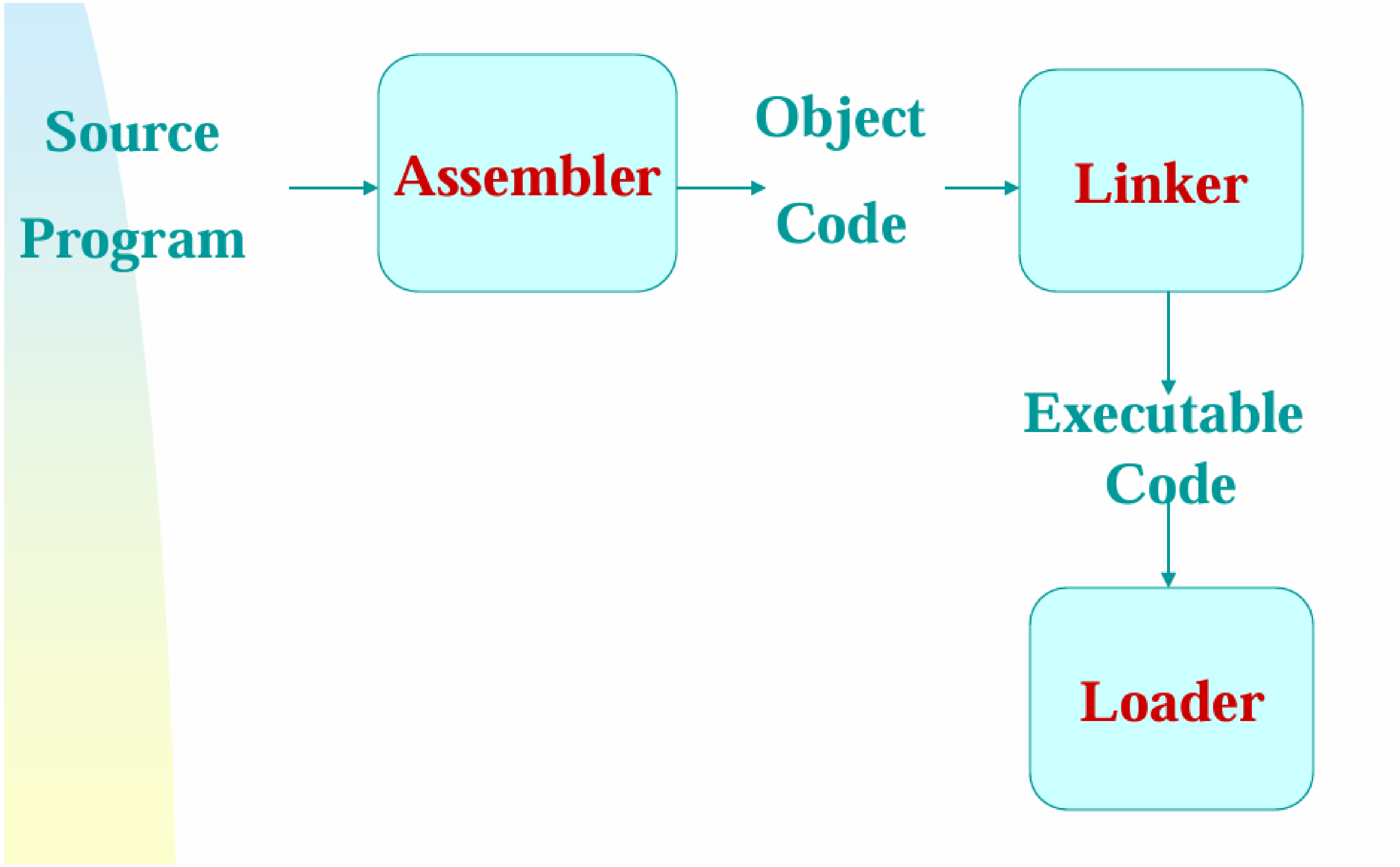
Assembler: Design Overview

CS 348
Implementation of Programming Languages Lab
Computer Science and Engineering Department
Indian Institute of Technology
Guwahati

Contents

Compilation Flow



Assembly to Object Code

```
BITS 32
GLOBAL _start

SECTION .data
   num1 dd 10
   num2 dd 20
   result dd 0

SECTION .text
_start:
   mov eax, [num1]     ; load num1 into EAX
   add eax, [num2]     ; EAX = num1 + num2
   mov [result], eax  ; store result

   mov eax, 1          ; sys_exit
   xor ebx, ebx        ; status = 0
   int 0x80
```

```
H ADD 001000 00001D T 001000 0C 0A000000
14000000 00000000 T 00100C 0D A1100000
030504100000 A308100000 F4 E 00100C
```

Assemblers

- Part of **system software**
- Bridge between **assembly language** and **machine code**

- **Primary responsibilities**

- Convert mnemonic instructions into opcodes
- Assign memory addresses to instructions and data
- Resolve symbolic names (labels)

• Some features:

– Depend on assembly language

– and machine language as well, may be very subtle

• Other features:

– Machine-independent,

– Even are arbitrary decisions by the designers of the language

Functions of an Assembler

- **Core functions**
- Mnemonic opcode → machine opcode
- Symbolic label → memory address
- Instruction formatting
- Data representation (constants)

- **Output**
- Object program (not executable yet)
- Optional assembly listing for debugging

Functions of an Assembler

- **Machine-specific aspects**
- Instruction formats and lengths
- Addressing modes
- Register sets and encodings

- **Implication**
- Structure of assembly program may look similar
- But the assembler itself must be tailored to a specific architecture

Basic Structure of x86 Program

- **Label** – symbolic name for a memory address, used as jump or data reference
- **Opcode** – specifies the operation to be performed (from mnemonic)
- **Operand(s)** – specify data, registers, or memory locations involved
- **Comment** – for human readability; ignored by the assembler
- **Assembler Directives -** Pseudo-instructions that **do not generate machine code**

- The assembler parses each line into these fields and processes them differently depending on whether the opcode is an instruction or a directive

Assembler Output

- **Object program**
  - Machine code in a relocatable format
  - Not directly executable

- **Assembly listing (optional)**
  - Source statements with addresses and object code
  - Used for debugging and understanding translation

Assembler Output

- **Object program**

```
a1 00 00 00 00
03 05 04 00 00 00
a3 08 00 00 00
b8 01 00 00 00
31 db
cd 80
```

- **Assembly Listing**

```
00000000 <_start>:
   0:   a1 00 00 00 00         mov    eax,ds:0x0
   5:   03 05 04 00 00 00      add    eax,ds:0x4
   b:   a3 08 00 00 00         mov    ds:0x8,eax
  10:   b8 01 00 00 00         mov    eax,0x1
  15:   31 db                  xor    ebx,ebx
  17:   cd 80                  int    0x80
```

Address Assignment

- Instructions and data share the **same address space**
- Instruction sizes may **vary in length**
- Address of a statement depends on **all preceding statements**
- Labels cannot be resolved in isolation

Address Assignment

```
section .text
    mov eax, [VALUE]          ; load integer from memory into eax
    add eax, 1                ; increment the value of eax by 1
    imul eax, [COUNT]         ; multiply the current eax value by the value in count
    jmp DONE                  ; unconditional jump to label

DONE:
    mov [RESULT], eax         ; writes the value from eax to memory

section .data
VALUE   dd 5                  ; integer values specified
COUNT   dd 2

section .bss
RESULT  resd 1
```

Address Assignment

```
section .text
    mov eax, [VALUE]          ; uses VALUE before definition (forward reference)
    add eax, 1                ; short instruction
    imul eax, [COUNT]         ; longer instruction
    jmp DONE                  ; forward reference, variable-length instruction

DONE:
    mov [RESULT], eax         ; memory store (longer encoding)

section .data
VALUE   dd 5                  ; defined later
COUNT   dd 2

section .bss
RESULT  resd 1
```

Forward Reference Problem

Forward reference: use of a symbol (label or variable) before its definition

Common in:

- Jumps and branches (loops, conditionals)
- Accessing data defined later in the program
- Address/value of the symbol is **unknown at first encounter**
- Assembler cannot immediately generate correct machine code

Solution

Scan the Code twice i.e Two pass

- **Pass one**

Assign addresses to all statements in source code

Save values assigned to labels for use in pass two

Perform some processing of assembler directives

- **Pass two**

Assemble instructions

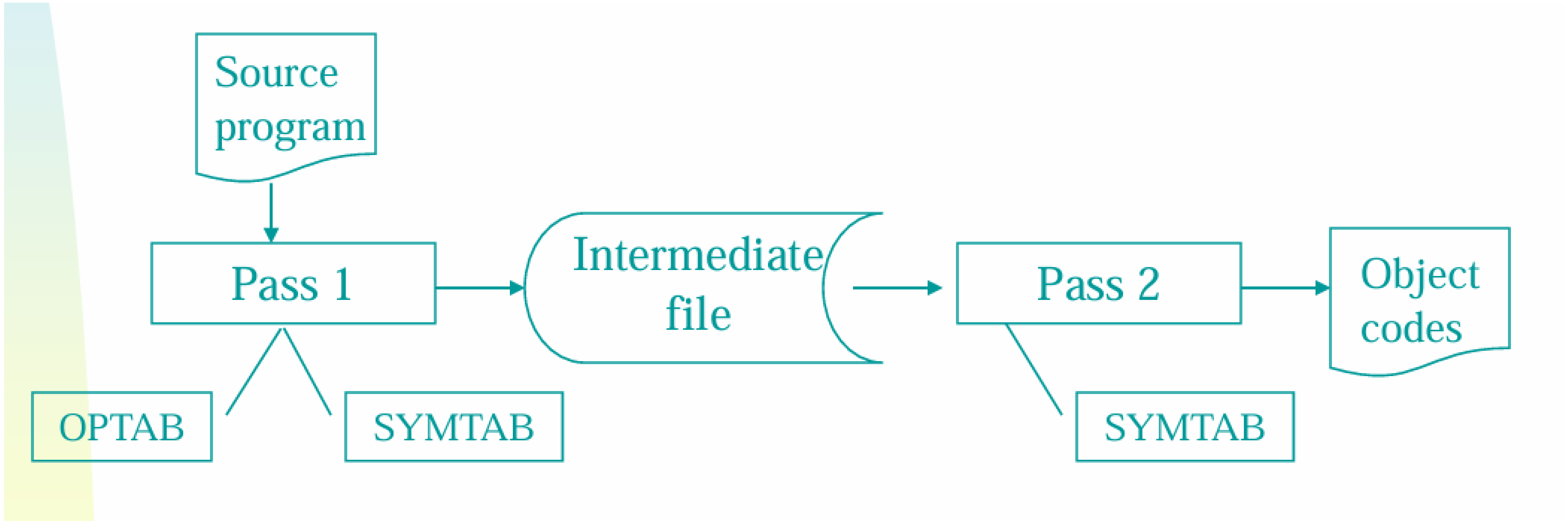
Generate data values defined by BYTE, WORD

Perform processing of assembler directives not done in

Pass 1

Write the object program and the assembly listing

Two Pass Assembler Diagram



Data Structures

- Operation Code Table (OPTAB)
- Symbol Table (SYMTAB)
- Location Counter(LOCCTR)

OPTAB (operation code table)

Table used for the mapping of mnemonics to machine opcodes

- **Content**

  Opcode value

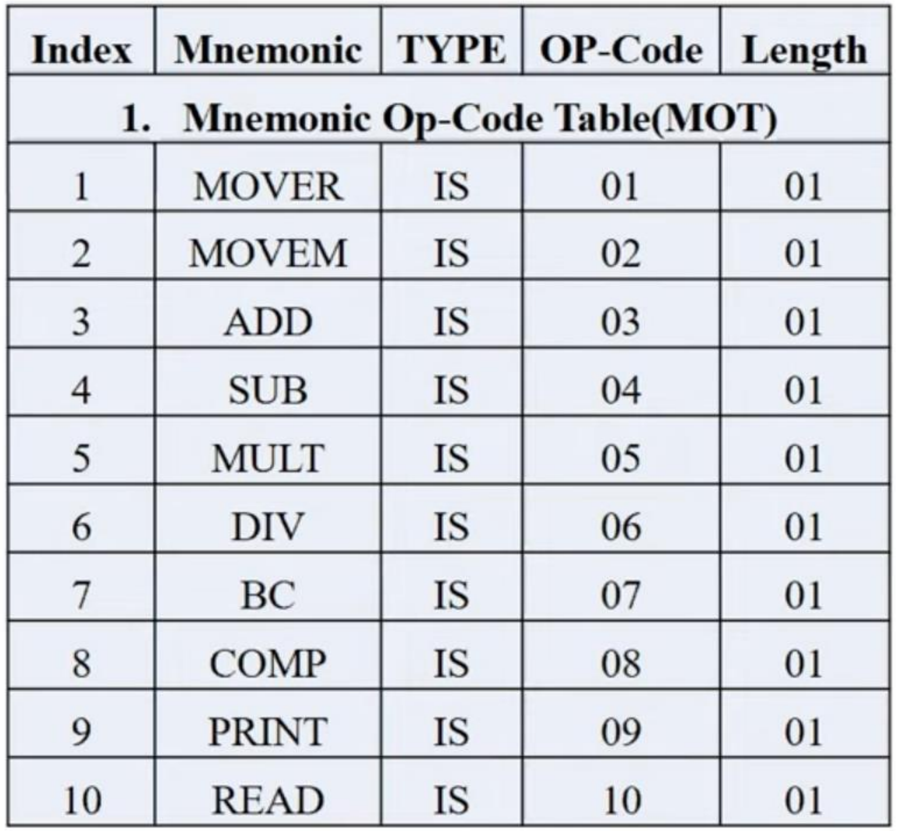
  Instruction format

  Instruction length
- **Characteristic**

  Static table

  Fixed for a given architecture
- **Implementation**

  Array or hash table, easy for search



Index	Mnemonic	TYPE	OP-Code	Length
1. Mnemonic Op-Code Table(MOT)				
1	MOVER	IS	01	01
2	MOVEM	IS	02	01
3	ADD	IS	03	01
4	SUB	IS	04	01
5	MULT	IS	05	01
6	DIV	IS	06	01
7	BC	IS	07	01
8	COMP	IS	08	01
9	PRINT	IS	09	01
10	READ	IS	10	01

SYMTAB (symbol table)

Table used for mapping symbolic names (labels) to addresses and values

- **Content**

  Symbol name

  Address / value

  Status flag (defined / undefined)

  Type (relative / absolute)

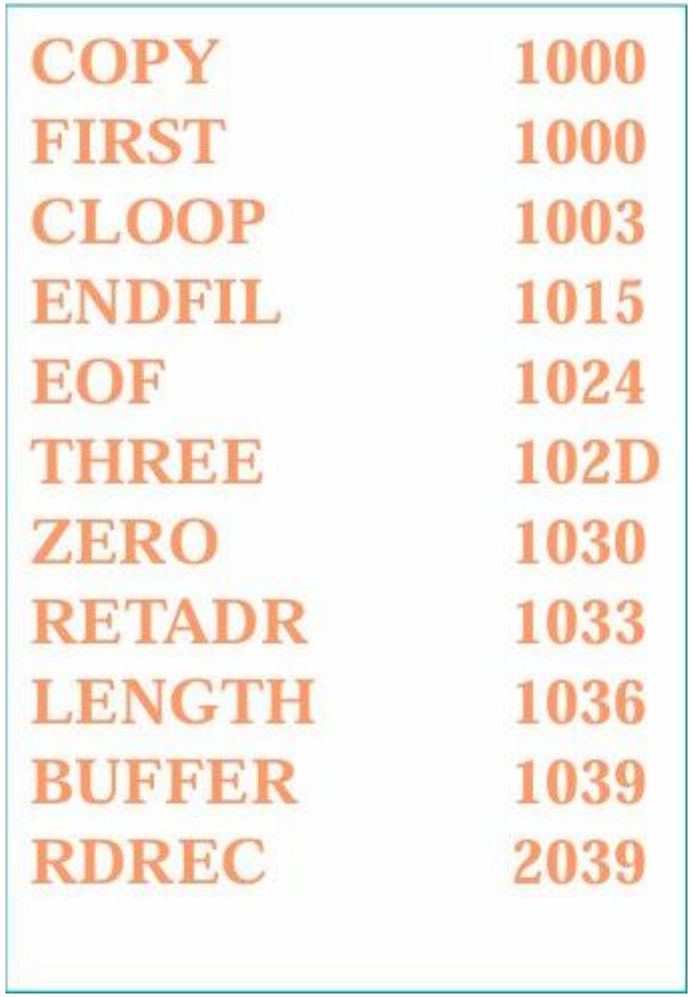
  Optional attributes (size, length)
- **Characteristic**

  Dynamic table

  Entries added as symbols are encountered during assembly
- **Implementation**

  Hash table (most common)

  Supports efficient **insert, search, and update** operations



COPY	1000
FIRST	1000
CLOOP	1003
ENDFIL	1015
EOF	1024
THREE	102D
ZERO	1030
RETADR	1033
LENGTH	1036
BUFFER	1039
RDREC	2039

LOCCTR (Location counter)

Tracks the address of the next instruction or data item to be placed in memory

- **Content**

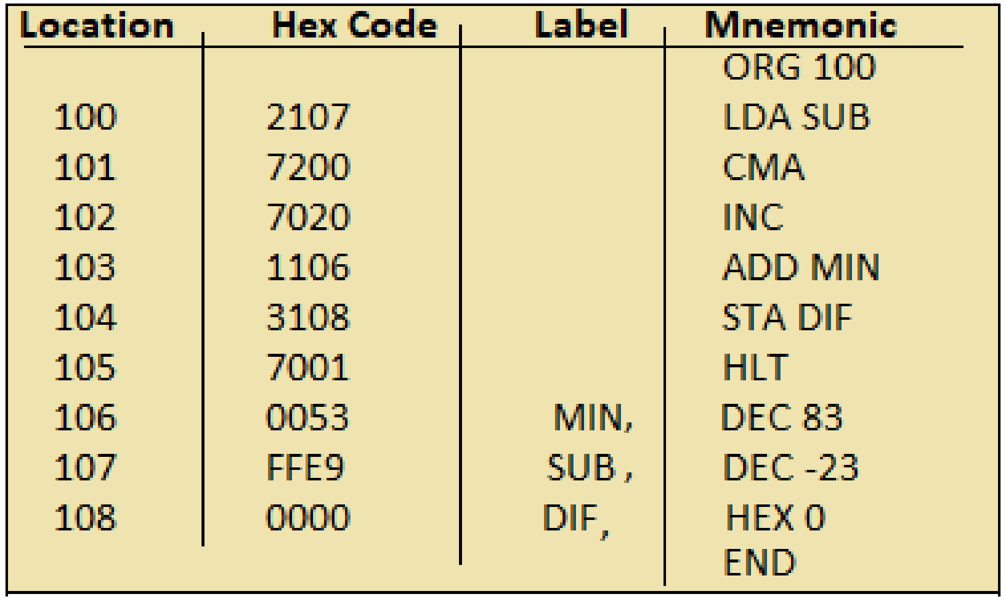
  Current memory address value
- **Characteristic**

  Single variable, not a table

  Updated sequentially as the source program is scanned
- **Implementation**

  Integer variable maintained by the assembler

  Incremented based on instruction length or data size



Location	Hex Code	Label	Mnemonic
			ORG 100
100	2107		LDA SUB
101	7200		CMA
102	7020		INC
103	1106		ADD MIN
104	3108		STA DIF
105	7001		HLT
106	0053	MIN,	DEC 83
107	FFE9	SUB ,	DEC -23
108	0000	DIF,	HEX 0
			END

One Pass Assembler

- Processes source program **only once**
- Assigns addresses and generates code simultaneously
- No separate symbol-discovery phase
- Designed for speed and low memory usage

Key Problem

Assembler must make decisions **before all information is available (again forward reference)**

One Pass Assembler

- Symbols may be used before they are defined
- Actual address is unknown at the point of use
- Assembler inserts temporary placeholder values
- Maintains a forward-reference (fix-up) list per symbol
- All pending references must be patched later
- Increases assembler complexity and bookkeeping

Data Structures in a One-Pass Assembler

- Symbol Table (SYMTAB)
- Location Counter (LOCCTR)
- Operation Code Table (OPTAB)
- **Forward Reference List**

Data Structures in a One-Pass Assembler

- Symbol Table (SYMTAB)

  Includes a **defined / undefined flag**

  Undefined symbols indicate forward references

  Entry updated when symbol is defined

- Location Counter (LOCCTR)

  Tracks current address during scan

- Operation Code Table (OPTAB)

  Determines instruction size

  Enables address advancement

Forward Reference (Fix-Up) List

Table used for tracking unresolved symbol references during assembly

- **Content**

  Symbol name (referenced but not yet defined)

  List of instruction locations requiring correction

  Type of reference (e.g., jump target, data address)

- **Characteristic**

  Temporary data structure

  Exists only while the symbol remains undefined

  Cleared once the symbol is defined

- **Implementation**

  Linked list or list embedded within SYMTAB entry

  Enables efficient addition of new references and updates when resolved

One Pass Vs Two Pass

Aspect	One Pass Assembler	Two Pass Assembler
Source Scans	Single	Two
Symbol Handling	Used Fix up Lists	Symbol resolved before code generation
Complexity	Higher	Lower
Flexibility	Limited	High