Lecture 3

Intermediate code forms

- Two criteria for choice of intermediate code
  - Processing efficiency
  - Memory economy
- **Intermediate code consists of a set of IC units**

- Each IC unit consists of following fields
  - Address
  - Representation of mnemonic opcode
  - Representation of operands

Fig : IC unit

Mnemonic field

- Contains a pair of the form

  (statement class, code)

- Statement class can be one of IS, DL and AD
- For imperative statement, code is instruction opcode
- For declarations and assembler directives, code is ordinal (entry) number within class

- Thus, (AD,01) stands for assembler directive number 1 which is directive START

Declaration statements	
DC	01
DS	02

Assembler Directives	
START	01
END	02
ORIGIN	03
EQU	04
LTORG	05

Fig: Codes for declaration statements and directives

Intermediate code for Imperative Statements

- Two variants of IC which differ in information contained in their operand fields

Variant 1 – 1/3

- First operand is represented by a single digit which is a code for a register (1-4 for AREG-DREG) or condition code (1-6 for LT-ANY)
- Second operand is a memory operand represented as a pair (operand class, code)
- Operand class is one of C (constant), S (symbol) and L (literal)
- For constant, code field contains internal representation of constant itself

  START 200 $\Rightarrow$ (C, 200)

Variant 1 – 2/3

- For symbol or literal, *code* field contains ordinal number of operand's entry in SYMTAB or LITTAB
  - Eg. Entries for a symbol XYZ and literal ='25' would be (S,17) and (L,35)

- Entry is made in SYMTAB only when a symbol occurs in label field of an assembly statement
  - Eg. An entry (A,345,1) is made, if symbol A is allocated one word at address 345

	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		(DL,05)	
…				

FIG : Intermediate Code – variant 1

Variant 1 – 3/3

- However, while processing a forward reference

  MOVER AREG, A

it is necessary to enter A in SYMTAB, say in entry n, so that it can be represented by (S,n) in IC

- At this point, *address* and *length* fields of A's entry cannot be filled in
- Two kinds of entries may exist in SYMTAB at any time
  - defined symbols
  - forward references

Variant 2

- Symbolic references in source statement are not processed at all during Pass I

- Literals are entered in LITTAB, and are represented as (L,m) in IC

	START	200	(AD,01)	(C,200)
	READ	A	(IS,09)	A
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		(DL,05)	
…				

FIG : Intermediate Code – variant 2

Comparison of variants – 1/3

- Variant 1
  - Require extra work in Pass I since operand fields are completely processed
  - Simplifies task of Pass II

Comparison of variants – 2/3

- Variant 2
  - Reduces the work of Pass I by transferring burden of operand processing from Pass I to Pass II of assembler
  - Code occupies more memory than code in Pass II
  - Overall memory requirements of assembler is lower
  - Well suited if expressions are permitted in operand fields of an assembly statement

Comparison of variants – 3/3

– Eg.

MOVER AREG, A+5

– Would appear as

(IS,05) (1) (S,01)+5 in variant 1 of IC

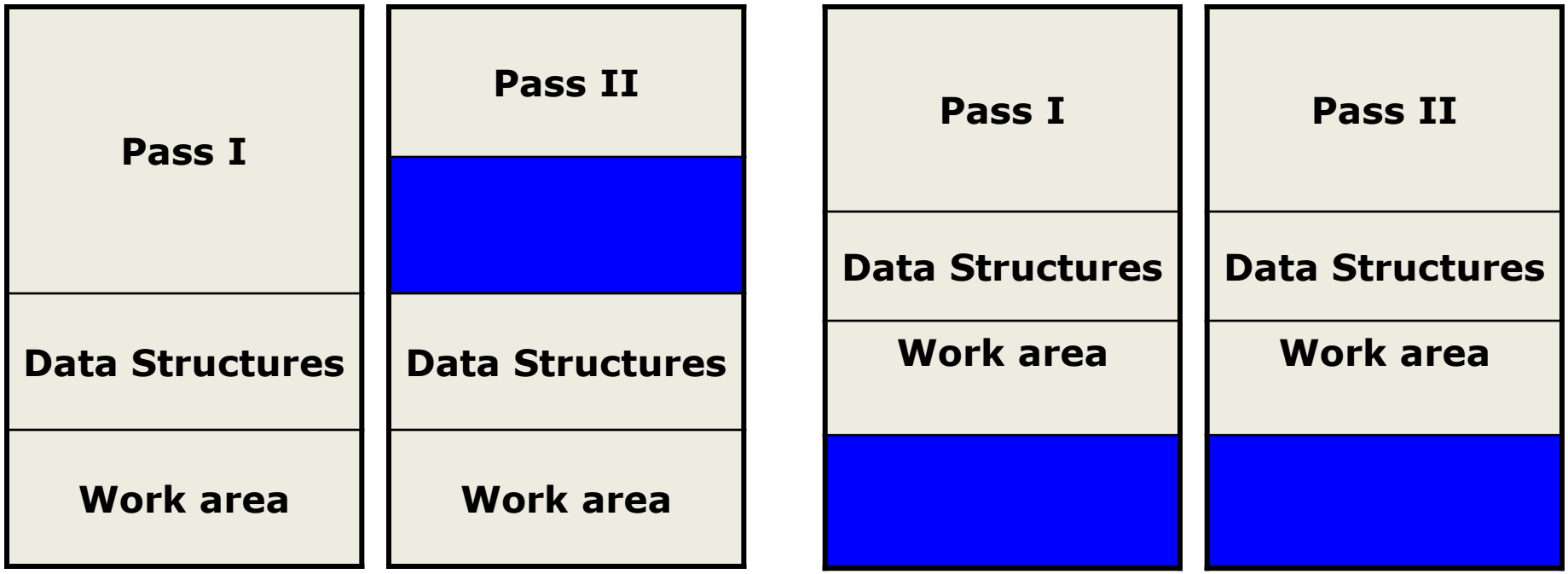

Fig: Memory requirements using (a) variant 1 and (b) variant 2

Processing of Declarations and Assembler Directives – 1/4

- DC statement
  - Must be represented in IC
  - Mnemonic field contains pair (DL,01)
  - Operand field may contain value of constant in source form
  - If DC statement defines many constants,

    DC ‘5, 3, -7’
  - A series of (DL,01) units can be put in IC

Processing of Declarations and Assembler Directives – 2/4

- START and ORIGIN
  - Set new values in LC
- LTORG
  - Pass I checks for presence of a literal reference in operand field of every statement
  - If one exists, enters literal in current literal pool in LITTAB
  - When LTORG statement appears in source program, it assigns memory addresses to literals in current pool
  - Addresses are entered in address field of their LITTAB entries

Processing of Declarations and Assembler Directives – 3/4

- Pass I simply construct an IC unit for LTORG statement and leave all subsequent processing to Pass II
- Values of literals can be inserted in target program when this IC unit is processed in Pass II
- Requires use of POOLTAB and LITTAB
- Pass I could itself copy out literals of the pool into the IC
- This avoids duplication of Pass I actions in Pass II

Processing of Declarations and Assembler Directives – 4/4

- IC for literal can be made identical to the IC for a DC statement so that no special processing is required in Pass II
- This alternative increases tasks to be performed by Pass I
- Also increase size
- Also, literals have to exist in two forms simultaneously
  - In LITTAB along with address information
  - In intermediate code

Pass II of Assembler – 1/4

- Target code is to be assembled in area named *code_area*

Algorithm for Assembler Second Pass

1. *code_area_address*:=address of *code_area*;
   pooltab_ptr:=1;
   loc_cntr:=0;

2. While next statement is not END statement
   (a) Clear *machine_code_buffer*;

Pass II of Assembler – 2/4

(b) If an LTORG statement

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

Pass II of Assembler – 3/4

(d) If a declaration statement then

(i) if a DC statement then

Assemble constant in *machine_code_buffer*;

(ii) *size*:= size of memory area required by DC/DS

(e) If imperative statement

(i) get operand address from SYMTAB or LITTAB

(ii) assemble instruction in *machine_code_buffer*

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

Pass II of Assembler – 4/4

(f) If *size* $\neq$ 0 then

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

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

3. Processing of END statement
   a) Perform steps 2(b) and 2(f)
   b) Write *code_area* into output file

Output interface of assembler

- Assumption made is that assembler produces a target program which is machine language of target computer
- This is rarely the case
- **Assembler produces an object module in the format required by a linkage editor or loader**

- Producing listing in first pass has advantage that the source program need not be preserved till Pass II
  - Conserves memory
  - Avoids some amount of duplicate processing

Listing and Error Reporting – 1/2

- Listing produced in Pass I can report only certain errors in most relevant place, i.e. against source statement itself

- Errors
  - syntax errors - like missing commas or parentheses
  - semantic errors - like duplicate definition of symbols

Listing and Error Reporting – 2/2

- **Errors like references to undefined variables can only be reported at the end of source program**
- Necessary to report all errors against erroneous statement itself

- Can be achieved by delaying program listing and error reporting till Pass II

Source organizational issues – 1/2

- Tables
  - SYMTAB must remain in main memory throughout Passes I and II of assembler
  - LITTAB is not accessed as frequently as SYMTAB
  - If memory is at a premium, possible to hold any part of LITTAB in memory because only literals of current pool need to be accessible at any time
  - **OPTAB should be in memory during Pass I**

Source organizational issues – 2/2

- Source program and Intermediate code
  - Source program would be read in Pass I on a statement by statement basis
  - Source statement can be written into a file for use in Pass II
  - IC generated for it would also be written to another file
  - Target code and program listings can be written out as separate files by Pass II

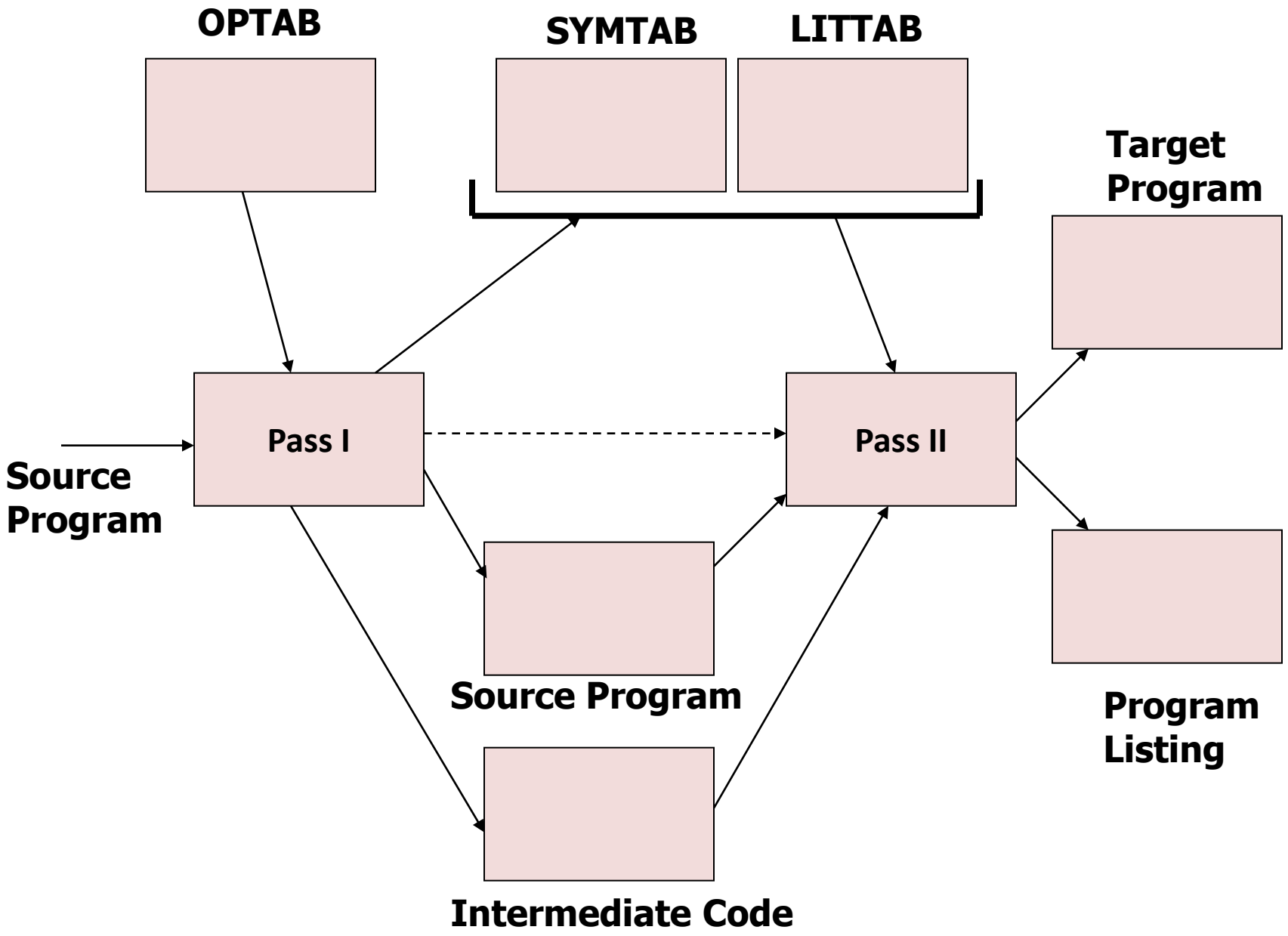

Fig: Data Structures and files in a two pass assembler