LECTURE 2

PASS STRUCTURES OF ASSEMBLERS

- Two pass translation
- Single pass translation

Two Pass translation – 1/2

- Can handle forward references easily
- LC processing is performed in first pass and symbols defined in the program are entered into the symbol table
- Second pass synthesizes the target form using the address information found in symbol table
- First pass perform analysis of source program
- Second pass perform synthesis of target program

Two Pass translation – 2/2

- First pass constructs an intermediate representation (IR) of source program for use by second pass

- Representation consists of two main components
  - Data structures eg. Symbol table
  - Processed form of source program. Called intermediate code (IC)

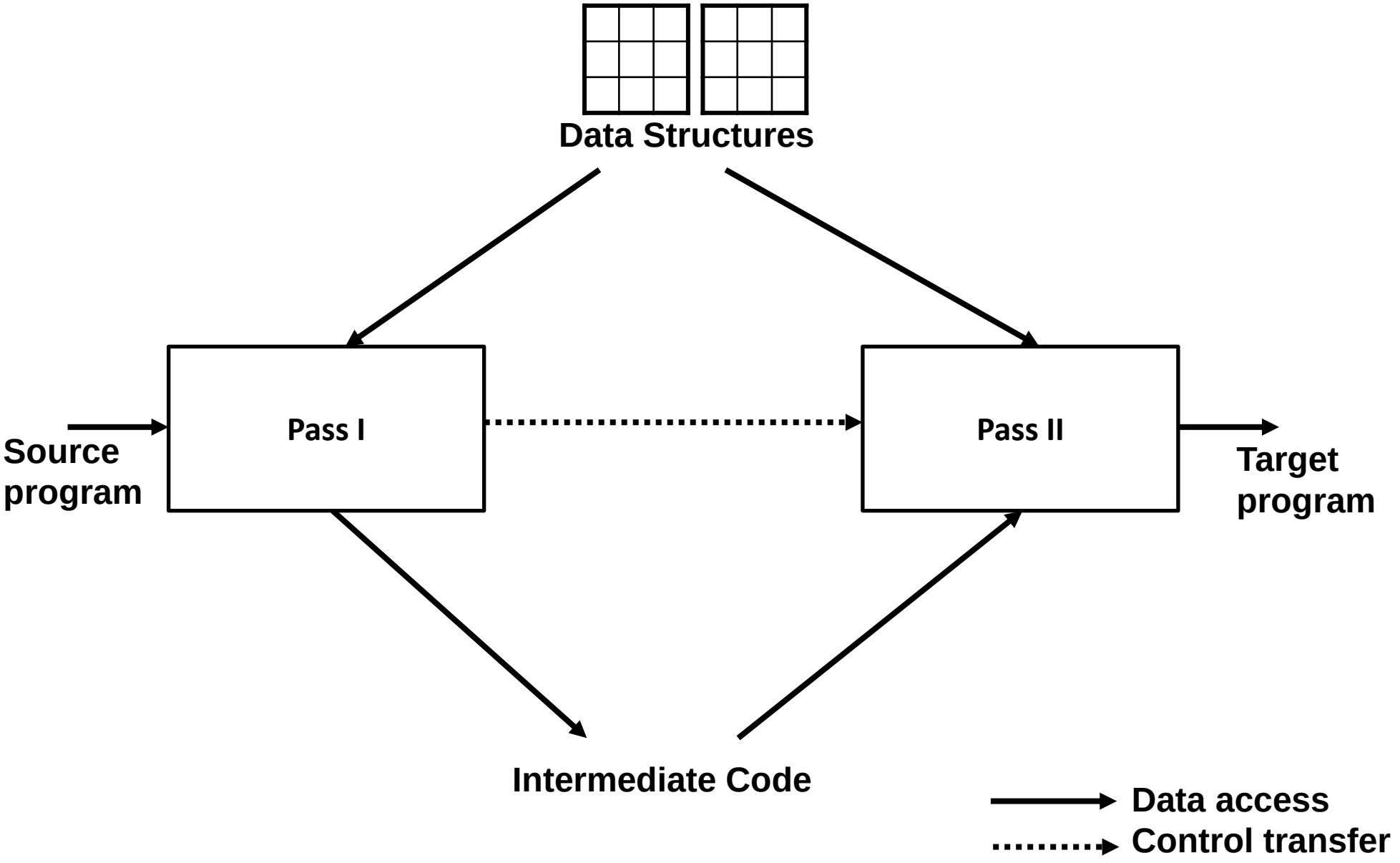

Fig : Overview of two pass assembly

Single pass translation – 1/3

- LC processing and construction of symbol table proceed as in two pass translation
- Problem of forward references is tackled using a process called **backpatching**
- Operand field of instruction containing a forward reference is left blank initially
- Address of forward referenced symbol is put into this field when its definition is encountered

Single pass translation – 2/3

- Instruction corresponding to the statement

  MOVER BREG, ONE

- Instruction opcode and address of BREG will be assembled to reside in location 101

- Need for inserting operand's address at a later stage can be indicated by adding an entry to the **Table of Incomplete Instructions (TII)**

- Entry is a pair **(<instruction address>,<symbol>)**

  eg. (101, ONE)

Single pass translation – 3/3

- By the time the END statement is processed, symbol table would contain the addresses of all symbols defined in source program

- **TII would contain information describing all forward references**

DESIGN OF A TWO PASS ASSEMBLER – 1/2

- Tasks performed by the passes of a two pass assembler
  - Pass I
    - Separate the symbol, mnemonic opcode and operand fields
    - Build the symbol table
    - Perform LC processing
    - Construct intermediate representation
  - Pass II
    - Synthesize the target program

DESIGN OF A TWO PASS ASSEMBLER – 2/2

- Pass I perform analysis of source program and synthesis of intermediate representation
- Pass II processes the intermediate representation to synthesize the target program

Advanced Assembler Directives – 1/6

- ORIGIN

  ORIGIN <address spec>

  - <address spec> is an <operand spec> or <constant>
  - Indicates that LC should be set to the address given by <address spec>
  - Useful when target program does not consist of consecutive memory words
  - Ability to use <operand spec> in ORIGIN statement provides the ability to perform LC processing in a relative rather than absolute manner

Advanced Assembler Directives – 2/6

- Example: Statement 18 of assembly program (next slide) viz. ORIGIN LOOP+2, sets LC to value 204, since symbol LOOP is associated with address 202. The next statement,

```
MULT        CREG, B
```

- is therefore given the address 204.
- The statement ORIGIN LAST+1 sets LC to address 217.
- Equivalent effect have been achieved by using statements ORIGIN 202 and ORIGIN 217 at these two places in program, however absolute addresses used in these statements would need to be changed if the address specification in START statement is changed

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
…	…	…	…	…	…
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

Advanced Assembler Directives – 3/6

- EQU

  <symbol> EQU <address spec>

  - <address spec> is <operand spec> or <constant>
  - Defines symbol to represent <address spec>
  - Differs from DS/DC statements as no LC processing is implied
  - EQU simply associates the name <symbol> with <address spec>

Advanced Assembler Directives – 4/6

- Example :- Statement 22 i.e. BACK EQU LOOP introduces symbol BACK to represent the operand LOOP.

Advanced Assembler Directives – 5/6

- LTORG
  - LTORG permits a programmer to specify where literals should be placed
  - Default, assembler places the literals after the END statement
  - At every LTORG statement, as also at the END statement, assembler locates memory to the literals of a literal pool
  - Pool contain all literals used in the program since the start of program or since the last LTORG statement

Advanced Assembler Directives – 6/6

- Example: The literals =‘5’ and =‘1’ are added to literal pool in statements 2 and 6 respectively. The first LTORG statement (no 13) allocates the addresses 211 and 212 to values ‘5’ and ‘1’
- A new literal pool is now started. Value ‘1’ is put into this pool in statement no 15. This value is allocated address 219 while processing END statement
- Literal =‘1’ used in statement no 15 refers to location 219 of second pool of literals rather than location 212 of first pool.

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
...	...	...	...	...	...
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

Pass I of the Assembler – 1/7

- Uses following data structures
  - OPTAB :- A table of mnemonic opcodes and related information
  - SYMTAB :- Symbol table
  - LITTAB :- A table of literals used in program

Pass I of the Assembler – 2/7

- OPTAB
  - Contains fields *mnemonic opcode, class* and *mnemonic info*
  - *Class* field - indicates whether opcode corresponds to imperative statement (IS), a declaration statement (DL) or assembler directive (AD)
  - If imperative statement, mnemonic info field contains pair (*machine opcode, instruction length*)
  - Otherwise it contains id of a routine to handle declaration or directive statements

Pass I of the Assembler – 3/7

- SYMTAB
  - Contains fields *address* and *length*
- LITTAB
  - Contains fields *literal* and *address*

Pass I of the Assembler – 4/7

- Processing of assembly statement begins with processing of its label field
- If it contains a symbol, symbol and value in LC is copied into a new entry of SYMTAB
- Functioning of Pass I centers around interpretation of OPTAB entry for mnemonic
- The *class* field of entry is examined to determine whether mnemonic belongs to class of imperative, declaration or assembler directive statements

Pass I of the Assembler – 5/7

- For imperative statement
  - *length* of machine instruction is simply added to LC
  - *length* is also entered in SYMTAB entry of symbol
- For declaration or assembler directive statement
  - *mnemonic info* field is called to perform appropriate processing of statement

Pass I of the Assembler – 6/7

- Use of LITTAB
  - First pass uses LITTAB to collect all literals used in program
  - Awareness of different literal pools is maintained using the auxiliary table POOLTAB
  - This table contains *literal no* of starting literal of each literal pool
  - Current literal pool is the last pool in LITTAB

Pass I of the Assembler – 7/7

- On encountering LTORG (or END) statement, literals in current pool are allocated addresses starting with current value in LC and LC is incremented
- Example : At LTORG statement, first two literals will be allocated addresses 211 and 212
- At END statement, third literal will be allocated address 219

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

OPTAB

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

SYMTAB

LITTAB

literal	address
='5'	
='1'	
='1'	

literal no
#1
#3

POOLTAB

FIG: Data structures of assembler Pass I

Algorithm of Assembler - First Pass [1/4]

1. *loc_cntr* := 0; (default value)

   pooltab_ptr := 1;

   POOLTAB[1]:=1;

   littab_ptr:=1;

2. While next statement is not END statement

   a) If label is present then

      this_label:=symbol in label field

      Enter (*this_label, loc_cntr*) in SYMTAB

Algorithm of Assembler - First Pass [2/4]

b) If an LTORG statement then

(i) Process literals LITTAB[POOLTAB[*pooltab_ptr*]] … LITTAB [*littab_ptr* – 1] to allocate memory and put address in 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*)

Algorithm of Assembler - First Pass [3/4]

(e) If a declaration statement then

(i) *code* := code of 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 imperative statement then

(i) *code* := machine opcode from OPTAB;

(ii) *loc_cntr* := *loc_cntr* + instruction length from OPTAB;

Algorithm of Assembler - First Pass [4/4]

(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 symbol)

this_entry := SYMTAB entry no of operand;

Generate IC '(IS, code) (S, *this_entry*)' ;

3. Processing END statement
   a) Perform step 2(b)
   b) Generate IC '(AD,02)'
   c) Goto pass II