

Pass Structure of Assembler

**Subject : Systems
Programming**

Assembly Language

- ▶ Assembly language is low level/middle level language.
- ▶ An assembly language is machine dependent.
- ▶ It differs from computer to computer.
- ▶ Writing programs in assembly language is very easy as compared to machine(binary) language.

Assembly language programming

Terms:

- ▶ **Location Counter: (LC)** points to the next instruction.

Literals: constant values

A literal is a character string whose value is given by the characters themselves

Symbols: name of variables and labels

Procedures: methods/ functions

Assembly language Statements:

- ▶ **Assembly language Statements:**
 - Imperative Statements:
 - Declarative/Declaration Statements:
 - Assembler Directive:

ASSEMBLY LANGUAGE STATEMENTS

- ▶ IMPERATIVE STATEMENTS
- ▶ These are executable statements
- ▶ They instruct the assembler to do certain task
- ▶ These generate some actions to instruct the assembler to perform certain tasks.
- ▶ Add x,y
- ▶ sub a,b
- ▶ read a
- ▶ Print y
- ▶ Mover, x,Areg

- ▶ Imperative means mnemonics
- ▶ These are executable statements.
- ▶ Each imperative statement indicates an action to be taken during execution of the program.
- ▶ E.g. MOVER BREG, X
- ▶ STOP
- ▶ READ X
- ▶ ADD AREG, Z

- ▶ **Declarative Statements**
- ▶ Declaration statements are for reserving memory for variables.
- ▶ We can specify the initial value of a variable.
- ▶ It has two types:
- ▶ DS // Declare Storage
- ▶ DC // Declare Constant

- ▶ DECLRATIVE STATEMENTS
- ▶ For reserving memory for the variable.
- ▶ One word of memory is reserverd using DS statement.
- ▶ DS- declare storage
- ▶ DC-declare constant
- ▶ A DS 1
- ▶ A DC '5'
- ▶ Or single statement B DC '5'

- ▶ **DS(Declare Storage):**
- ▶ Syntax:
- ▶ [Label] DS <Constant specifying size>
- ▶ E.g.X DS 1
- ▶ **DC (Declare Constant):**
- ▶ Syntax:
- ▶ [Label] DC <constant specifying value>
- ▶ E.g Y DC '5'

- ▶ For variable A is reserved one word of memory
- ▶ Using DC we can initialize the variable with value 5
- ▶ We can do both steps by single statement using B DC '5'
- ▶ By default one word of memory is reserved for the variable B and then it is assigned or initialized with value 5.

- ▶ Assembler directives
- ▶ Directs the assembler to do certain action
- ▶ START < constant >
- ▶ Starts the program and gives the starting address of the program as a constant.
- ▶ END

PROGRAM TERMINATES AFTER THE END STATEMENT

- ▶ **Assembler Directive**
- ▶ Assembler directive instruct the assembler to perform certain actions during assembly of a program.
- ▶ Some assembler directive are:
- ▶ START <address constant>
- ▶ END

Sample Assembly Code

1. START 100 It is an AD statement because it has Assembler directive START
2. MOVER AREG, X It is an IS because it starts with mnemonic.
3. MOVER BREG, Y
4. ADD AREG, Y
5. MOVEM AREG, X

6. X DC '10' It is an DS/ DL statement because it has DC
7. Y DS 1 It is an DS/ DL statement because it has DS
8. END

Identify the types of

State.No	IS	DS	AD
1			
2			
3			
4			
5			
6			
7			
8			

ADVANCED ASSEMBLER DIRECTIVES

- ▶ Modified unique statements to perform some special functions
- ▶ It directs the assembler to perform some specific action.
- ▶ ORIGIN
- ▶ LTORG
- ▶ EQU

- ▶ Start 200
- ▶ MOVE BREG='5' 200
- ▶ LOOP MOVE AREG N 201
- ▶ ADD BREG = '2' 202
- ▶ ORIGIN LOOP+5
- ▶ NEXT BC ANY, LOOP 206
- ▶ LTORG 207
- ▶ ORIGIN NEXT+3
- ▶ N DC 5 209
- ▶ END 210

- ▶ First check where is origin statement
- ▶ Right hand side of origin is constant and operand
- ▶ ORIGIN statement tells us that what should be the address of next immediate instruction
- ▶ In a given program it will evaluate the this address using statement loop+5
- ▶ Loop label is at the address 201 and after adding 5 to it we get address 206 which is the address should be assigned to label NEXT of the next instruction. So the address of instruction immediate after ORIGIN is given the address 206.

Advanced Assembler Directives

LTORG :

The LTORG directive instructs the assembler to assemble the current literal pool immediately.

The Literal Pool contains a collection of anonymous constant definitions, which are generated by the assembler.

The LTORG directive defines the start of a literal pool.

Syntax :

LTORG

LTORG

- ▶ In the above program the literal occurs 2 times
- ▶ The literals are represented by '' for example in the program the literals are '5' and '2'
- ▶ Literals are stored in literal table
- ▶ But the address are not assigned to them
- ▶ Address are assigned only when LTORG instruction occurs in the program
- ▶ When this instruction runs

Only then the address are assigned to the literal.

index	literal	address	
0	='4'	207	
1	='2'	208	

- ▶ In the above literal table address are not assigned to '2' and '4'
- ▶ When the LTORG instruction occurs then only the literals are assigned the address
- ▶ The address of the literal '4' is assigned the address value which is the address of LTORG instruction i.e 207
- ▶ Next literal '4' is assigned the address 208 which is the next immediate address after LTORG instruction.

LTORG

- ▶ If the LTORG instruction does not occur in the program then literals are not assigned any address till the end of the program.
- ▶ After the program ends then only literals are assigned the addresses which are the next address where program ends.

Advanced Assembler Directives

Equate:

The EQU directive is used to equate a name with an expression, symbolic address, or number. Whenever this name is used as a symbol, it is replaced.

We might do something, such as the following, which makes the symbol R12 to be equal to 12, and replaced by that value when the assembler is run.

E.G. R12 EQU 12

EQU

- ▶ < LABEL> EQU < ADDRESS>
- ▶ X EQU Y
- ▶ Label X is associated with address Y
- ▶ EQU is used to give the label or symbol which is on left side of EQU assign or associate the address given on right hand side of Y.

How LC Operates?

Sr. NO		LC
1	START 100	
2	MOVER AREG, X	100
3	MOVER BREG, Y	101
4	ADD AREG, BREG	102
5	MOVEM AREG, X	103
6	X DC '10'	104
7	Y DC '15'	105
8	END	

Identify symbol, literals, AD, DS, IS , Symbol, Literal Label

- START 100
- MOVER BREG, ='2'
- LOOP MOVER AREG, N
- ADD BREG, ='1'
- ORIGIN LOOP+5
- LTORG
- ORIGIN NEXT +2
- LAST STOP
- N DC '5'
- END

Solution (From Previous Example)

Sr. No	AD	DS	IS	Symbol	Literal	Label
1	AD					
2			IS		=2	
3			IS	N		LOOP
4			IS		=1	
5	AD					
6	AD					
7	AD					
8			IS			LAST
9		DS				
10	AD					

ASSEMBLER

- ▶ **Assembler**
 - ▶ • An Assembler is a translator which translates assembly language code into machine language with help of data structure.
 - ▶ • It has two types
 - ▶ • Pass 1 Assembler.
 - ▶ • Pass 2 Assembler.

ASSEMBLER

- ▶ **General design procedure of assembler**
 - ▶ • Statement of Problem
 - ▶ • Data Structure
 - ▶ • Format of databases
 - ▶ • Algorithms
 - ▶ • Look for modularity.

ASSEMBLER

- ▶ **Data Structure Used**
 - ▶ • Data Structure used are as follows:
 - ▶ • Symbol table
 - ▶ • Literal Table
 - ▶ • Mnemonic Opcode Table
 - ▶ • Pool Table

Format of Databases

- Symbol Table:

Name of Symbol	address

- Literal Table:

Literal	address

- MOT:

Mnemonic	Machine Opcode	Class	Length

- Pool Table:

Literal Number

Pass 1 Assembler

- ▶ **Pass 1 Assembler**
 - ▶ • Pass 1 assembler separate the labels , mnemonic opcode table, and operand fields.
 - ▶ • Determine storage requirement for every assembly language statement and update the location counter.
 - ▶ • Build the symbol table. Symbol table is used to store each label and each variable and its corresponding address.

How pass 1 assembler works?

- ▶ **How pass 1 assembler works?**
- ▶ • Pass I uses following data structures.
- ▶ • 1. Machine opcode table.(MOT)
- ▶ • 2. Symbol Table(ST)
- ▶ • 3. Literal Table(LT)
- ▶ • 4. Pool Table(PT)
- ▶ • Contents of MOT are fixed for an assembler.

- Pass 2 Assembler: Generate the machine code

Observe Following Program

```
+ START 200
+ MOVER AREG, ='5'
+ MOVEM AREG, X
+ L1 MOVER BREG, ='2'
+ ORIGIN L1+3
+ LTORG
+ NEXT ADD AREG, ='1'
+ SUB BREG, ='2'
+ BC LT, BACK
+ LTORG
+ BACK EQU L1
+ ORIGIN NEXT+5
+ MULT CREG, ='4'
+ STOP
+ X DS 1
+ END
```

Apply LC

START 200

	MOVER AREG, ='5'	200
	MOVEM AREG, X	201
L1	MOVER BREG, ='2'	202
	ORIGIN L1+3	
	LTORG	
	='5'	205
	='2'	206

NEXT	ADD AREG, ='1'	207
	SUB BREG, ='2'	208
	BC LT, BACK	209
	LTORG	
	='1'	210
	='2'	211

BACK	EQU L1	
	ORIGIN NEXT+5	
	MULT CREG, ='4'	212
	STOP	213
	X DS 1	214
	END	
	='4'	215

Construct Symbol table

index	Symbol Name	Address
0	X	214
1	L1	202
2	NEXT	207
3	BACK	202

Construct Literal Table

index	Literal	Address
0	5	205
1	2	206
2	1	210
3	2	211
4	4	215

Pool Table.

- Pool table contains starting literal(index) of each pool.

Literal number

0

2

4

NOW CONSTRUCT INTERMEDIATE CODE/MACHINE CODE

- ▶ For constructing intermediate code we need MOT.

Enhanced Machine opcode Table

Table 1.10.1 : An enhanced machine opcode table (MOT)

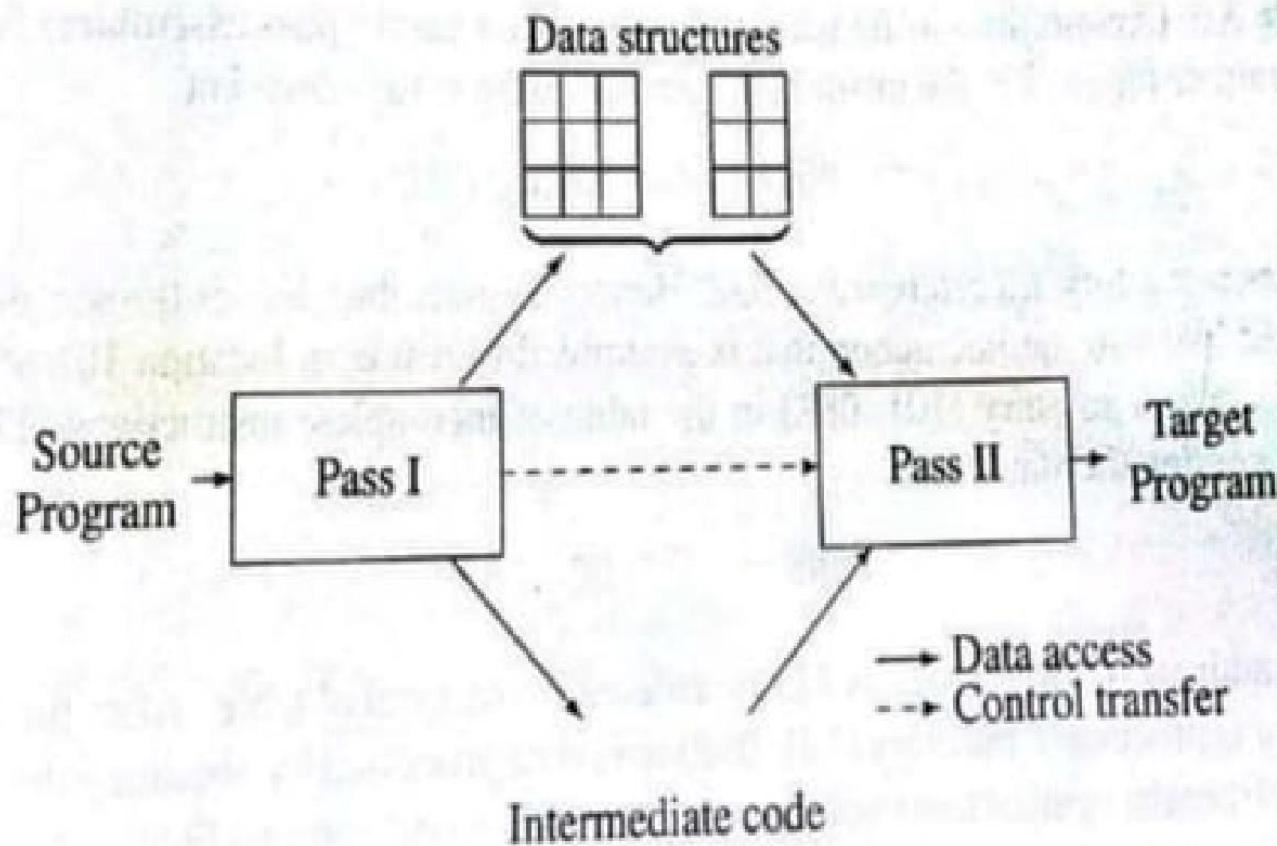
	Mnemonic opcode	Class	Opcode	Length
0	STOP	IS	00	1
1	ADD	IS	01	1
2	SUB	IS	02	1
3	MULT	IS	03	1
4	MOVER	IS	04	1
5	MOVEM	IS	05	1
6	COMP	IS	06	1
7	BC	IS	07	1
8	DIV	IS	08	1
9	READ	IS	09	1
10	PRINT	IS	10	1
11	START	AD	01	-
12	END	AD	02	-
13	ORIGIN	AD	03	-
14	EQU	AD	04	-
15	LTORG	AD	05	-
16	DS	DL	01	-
17	DC	DL	02	1
18	AREG	RG	01	-
19	BREG	RG	02	-
20	CREG	RG	03	-
21	EQ	CC	01	-

Mnemonic opcode	Class	Opcode	Length
LT	CC	02	-
GT	CC	03	-
LE	CC	04	-
GE	CC	05	-
NE	CC	06	-
ANY	CC	07	-

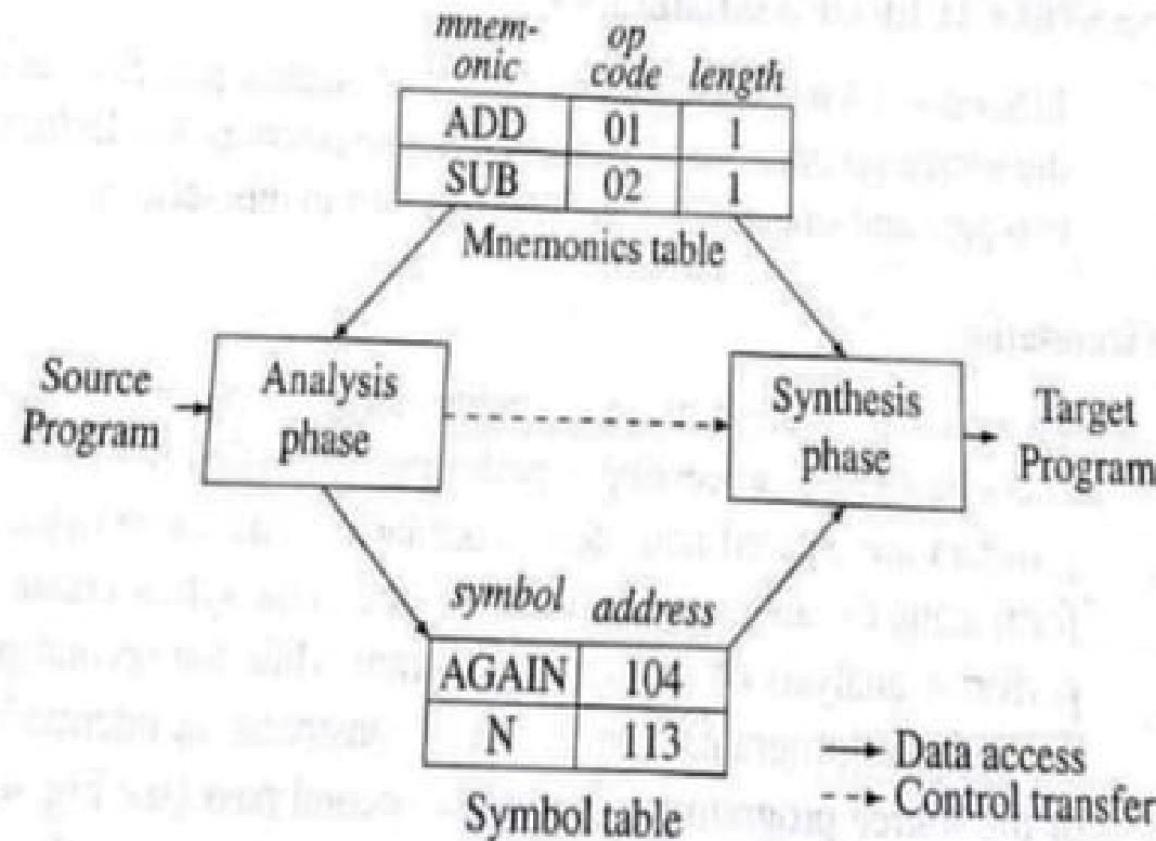
Design of two pass assembler

- Tasks performed by the passes of a two pass assembler are as follows:
- **Pass 1:**
 1. Separate the symbol, mnemonic opcode, and operand fields.
 2. Build the symbol table.
 3. Perform LC processing.
 4. Construct intermediate representation(or IC).
- **Pass 2:**
 1. Synthesize the target program.

Two Pass Assembler



Analysis Phase Vs. Synthesis Phase



Comparison between Pass 1 and Pass2

Sr. No	Pass 1	Pass 2
01	It requires only one scan to generate machine code	It requires two scan to generate machine code.
02	It has forward reference problem.	It don't have forward reference problem.
03	It performs analysis of source program and synthesis of the intermediate code.	It process the IC to synthesize the target program.
04	It is faster than pass 2.	It is slow as compared to pass 1.

ASSEMBLER

- An assembly language program can be translated into machine language.
- It involves following steps:
 - 1. Find addresses of variable.
 - 2. Replace symbolic addresses by numeric addresses.
 - 3. Replace symbolic opcodes by machine operation codes.
 - 4. Reserve storage for data.

Step 1

- We can find out addresses of variable using LC.
- First identify all variables in your program.
- START 100
- MOVER AREG, X
- MOVER BREG, Y
- ADD AREG, X
- MOVEM AREG, X
- X DC '10'
- Y DC '15'
- END

Step 1

Sr. NO		LC
1	START 100	
2	MOVER AREG, X	100
3	MOVER BREG, Y	101
4	ADD AREG, X	102
5	MOVEM AREG, X	103
6	X DC '10'	104
7	Y DC '15'	105
8	END	

Sr. No	Name of Variable(Symbol)	Address
1	X	104
2	Y	105

Step2: Replace all symbolic address with numeric address.

- START 100
 - MOVER AREG, **104**
 - MOVER BREG, **105**
 - ADD AREG, **104**
 - MOVEM AREG, **104**
 - X DC '10'
 - Y DC '15'
 - END
- 
- Memory is reserved but no code is generated.

Step3: Replace symbolic opcodes by machine operation codes.

LC	Assembly Instruction	Machine Code
101	MOVER AREG, 104	04 01 104
102	MOVER BREG, 105	04 02 105
103	ADD AREG, 104	01 01 104
104	MOVEM AREG, 104	05 01 104
105		
106		
107		

Question For u

```
START 101
READ N
MOVER BREG, ONE
MOVEM BREG,
AGAIN   TERM MULT BREG,
        TERM MOVER CREG,
        TERM ADD CREG,
        ONE MOVEM CREG,
        TERM COMP CREG,
        N
        BC LE, AGAIN
        MOVEM BREG, RESULT
        PRINT RESULT
        STOP
N DS 1
RESULT DS 1
ONE DC '1'
TERM DS 1
```

Question For U

```
START 102
READ X
READ Y
MOVER AREG, X
ADD AREG, Y
MOVEM AREG, RESULT
PRINT RESULT
STOP
X DS 1
Y DS 1
RESULT DS 1
END
```

Forward Reference Problem

- Using a variable before its definition is called as forward reference problem.
- E.g.
- START 100
- MOVEM AREG, X
- MOVER BREG, Y
- ADD AREG, Y
- X DC '4'
- Y DC '5'
- END

- In example variable X, Y are making forward reference.
- So, We can solve it by using back patching.

Consider another example

	START	100
	MOVER	AREG, X
L1	ADD	BREG, ONE
	ADD	CREG, TEN
	STOP	
X	DC	'5'
ONE	DC	'1'
TEN	DC	'10'
	END	

The diagram illustrates the memory locations for the variables defined in the assembly code. It shows three vertical columns: variable names (X, ONE, TEN), their corresponding data values (DC), and the character representations ('5', '1', '10'). Arrows point from the variable names to their respective memory locations.

- X → DC → '5'
- ONE → DC → '1'
- TEN → DC → '10'

Apply LC

	START	100	
	MOVER	AREG, X	100
L1	ADD	BREG, ONE	101
	ADD	CREG, TEN	102
	STOP		103
X	DC	'5'	104
ONE	DC	'1'	105
TEN	DC	'10'	106
	END		

Try to convert into machine code

	START	100				
	MOVER	AREG, X	100	04	1	---
L1	ADD	BREG, ONE	101	01	2	---
	ADD	CREG, TEN	102	06	3	---
	STOP		103	00	0	000
X	DC	'5'	104			
ONE	DC	'1'	105			
TEN	DC	'10'	106			
	END					

Backpatching

- The operand field of instruction containing a forward reference is left blank initially.
- Step 1: Construct TII(Table of incomplete instruction)

Instruction Address	Symbol Making a forward reference
100	X
101	ONE
102	TEN

- **Step 2 :** After encountering END statement symbol table would contain the address of all symbols defined in the source program.

SYMBOL	NAME	ADDRESS
	X	104
	ONE	105
	TEN	106

- Now we can generate machine code...

04	1	104
01	2	105
01	03	106
00	0	000

Single Pass Assembler

- Forward reference: reference to a label that is defined later in the program.

What is

Forward
reference

??Eg..

Instruction	Address	Symbol
START	200	
ADD	AREG	A
..	..	
A	DS	2
..	..	
END		
200		A
..	..	

- Table of Incomplete Instruction

Single Pass Assembler

- Forward reference: reference to a label that is defined later in the program.

Symbol	Address	START	200	
A		ADD	AREG	A
		
	A	DS	2	
		
	END			
Instruction Address		Symbol		
200		A		
..		

- Table of Incomplete Instruction

Single Pass Assembler

- Forward reference: reference to a label that is defined later in the program.

STAR 200

T

ADD AREG A

DS 2

END

Target Code

200 01 01 -

...

202

- Table of Incomplete Instruction

Instruction Address	Symbol
200	A
..	..

Symbol	Address
A	202
..	..

Target Code
200 01 01 202

..
202

Single pass Assembler Examples

SOURCE PROGRAM			TARGET CODE			TII	
						INSTRUCTI	SYMBOL
						ON Address	
	START	101					
	READ	N	101	09	-	101	N
	MOVER	BREG, ONE	102	04	02 -	102	ONE
	MOVEM	BREG, TERM	103	05	02 -	103	TERM
AGAIN	MULT	BREG, TERM	104	03	02 -	104	TERM
	MOVER	CREG, TERM	105	04	03 -	105	TERM
	ADD	CREG, ONE	106	01	03 -	106	ONE
	MOVEM	CREG, TERM	107	05	03 -	107	TERM
	COMP	CREG, TERM	108	06	03 -	108	TERM
	BC	LE, AGAIN	109	07	02 104	110	RESULT
	MOVEM	BREG, RESULT	110	05	02 -	111	RESULT
	PRINT	RESULT	111	10	-		
	STOP		112	00			
N	DS	1	113				
RESULT	DS	1	114				
ONE	DC	'1'	115				
TERM	DS	1	116				
	END						

Single pass Assembler Examples

SOURCE PROGRAM

	START	101
	READ	N
	MOVER	BREG, ONE
	MOVEM	BREG, TERM
AGAIN	MULT	BREG, TERM
	MOVER	CREG, TERM
	ADD	CREG, ONE
	MOVEM	CREG, TERM
	COMP	CREG, TERM
	BC	LE, AGAIN
	MOVEM	BREG, RESULT
	PRINT	RESULT
	STOP	
N	DS	1
RESULT	DS	1
ONE	DC	'1'
TERM	DS	1
	END	

TARGET CODE

101	09	11	3
102	04	02	115
103	05	02	116
104	03	02	116
105	04	03	116
106	01	03	115
107	05	03	116
108	06	03	116
109	07	02	104
110	05	02	114
111	10	11	4
112	00		
113			
114			
115			
116			

TII

INSTRUCTI ON Address	SYMBOL
101	N
102	ONE
103	TERM
104	TERM
105	TERM
106	ONE
107	TERM
108	TERM
110	RESULT
111	RESULT

Symbol Table

SYMBOL	ADDRES S
N	113
ONE	115
TERM	116
AGAIN	104
RESULT	114

1.3 Design of Two Pass Assembler

- PASS I
 - Separate the symbol, mnemonic opcode, and operand fields.
 - Build the symbol table
 - Perform LC processing
 - Construct intermediate code
- PASS II
 - Synthesize the target code

Three Main Data Structures

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

OPTAB operation code table

- Content
 - The mapping between mnemonic and machine code. Also include the instruction format, available addressing modes, and length information.
- Characteristic
 - Static table. The content will never change.
- Implementation
 - Array or hash table. Because the content will never change, we can optimize its search speed.
- In pass 1, OPTAB is used to look up and validate mnemonics in the source program.
- In pass 2, OPTAB is used to translate mnemonics to machine instructions.

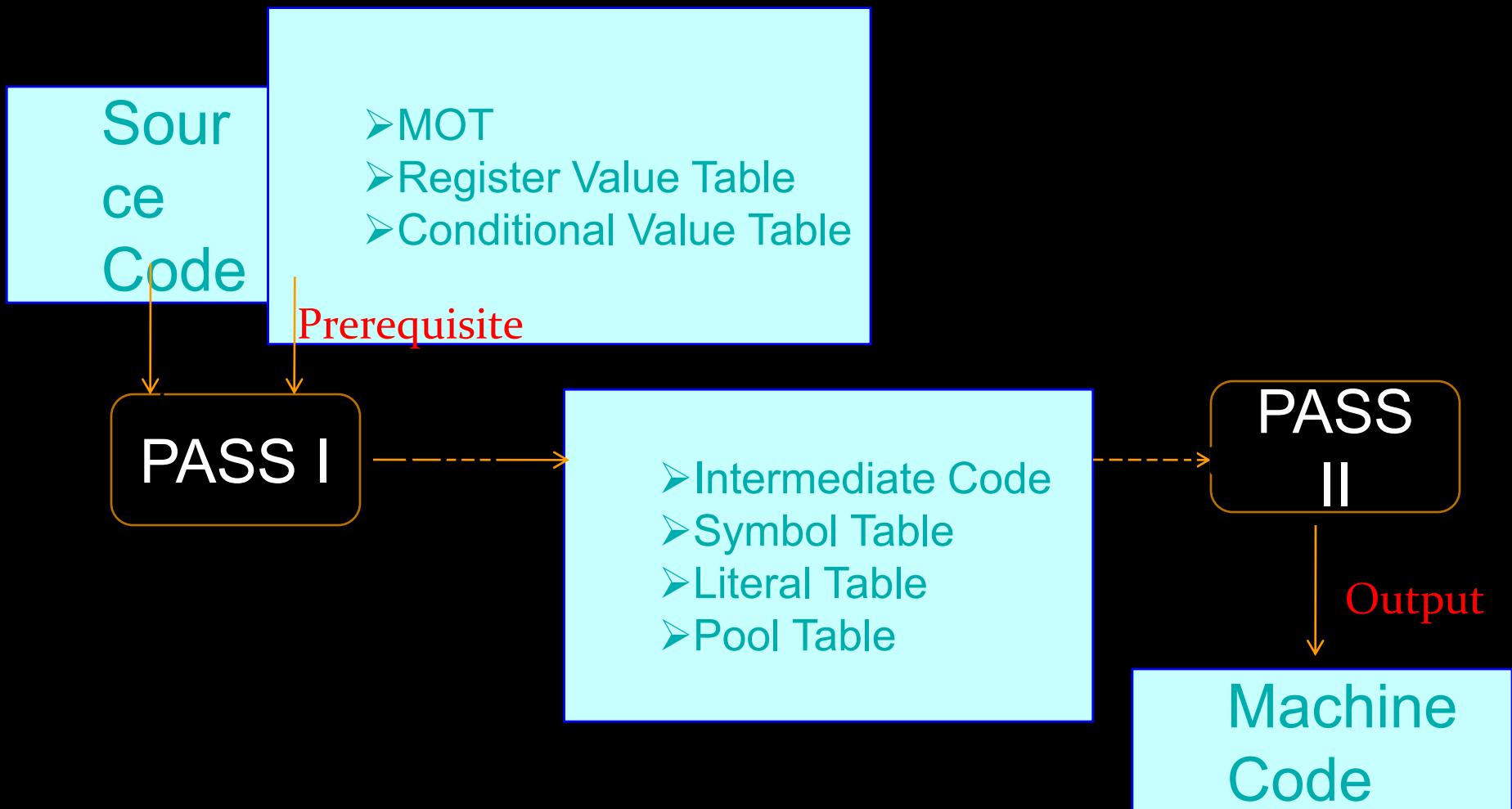
Location Counter (LOCCTR)

- This variable can help in the assignment of addresses.
- It is initialized to the beginning address specified in the START statement.
- After each source statement is processed, the length of the assembled instruction and data area to be generated is added to LOCCTR.
- Thus, when we reach a label in the source program, the current value of LOCCTR gives the address to be associated with that label.

Symbol Table(SYMTAB)

- Content
 - Include the label name and value (address) for each label in the source program.
 - Include type and length information (e.g., int64)
 - With flag to indicate errors (e.g., a symbol defined in two places)
- Characteristic
 - Dynamictable (I.e., symbols may be inserted, deleted, or searched in the table)
- Implementation
 - Hash table can be used to speed up search
 - Because variable names may be very similar (e.g., LOOP1, LOOP2), the selected hash function must perform well with such non-random keys.

Required Data Structures for Two Pass Assembler



Intermediate Code Forms

- Variant I
- Variant II : Symbols, registers and conditional code

SOURCE PROGRAM		
	START	200
	READ	A
LOOP	MOVER	AREG, A
:		
:		
SUB	AREG,	=‘1’

VARIANT -I		
(AD,01)	(C,200)	
(IS,09)	(S,01)	
(IS,04)	1	(S,01)
:		
:		
(IS,02)	1	(L,01)

VARIANT -II		
(AD,01)	(C,200)	
(IS,09)	A	
(IS,04)	AREG,	A
:		
:		
(IS,02)	AREG,	(L,01)

Assembler Pass I

SOURCE PROGRAM		
	START	200
	MOVER	AREG, =‘5’
	MOVEM	AREG, A
LOOP	MOVER	AREG, A
	MOVER	CREG, B
	ADD	CREG, =‘1’
	LTORG	
NEXT1	SUB	AREG, ‘=1’
	ORIGIN	LOOP+1
	MULT	CREG, B
A	DS	2
BACK	EQU	LOOP
B	DC	1
	END	

INTERMEDIATE CODE			
	(AD,01)		(C,200)
200	(IS,04)	1	(L,01)
201	(IS,05)	1	(S,01)
202	(IS,04)	1	(S,01)
203	(IS,04)	3	(S,03)
204	(IS,01)	3	(L,02)
	(AD,05)		
205			005
206			001
207	(IS,02)	1	(L,03)
	(AD,03)	(S,02)+(C,1)	
)	
203	(IS,03)	3	(S,03)
204	(DL,02)	(C,02)	
	(AD,04)	(S,02)	
206	(DL,02)	(C,02)	
	(AD,02)		
207			001

Assembler Pass I

SOURCE PROGRAM			
	START	200	
	MOVER	AREG, =‘5’	
	MOVEM	ADD, A	
LOOP	MOVER	AREG, A	
	MOVER	CREG, B	
	ADD	CREG, =‘1’	
	LTORG		
NEXT1	SUB	AREG, ‘=1’	
	ORIGIN	LOOP+1	
	MULT	CREG, B	
A	DS	1	
BACK	EQU	LOOP	
B	DS	1	
	END		

INTERMEDIATE CODE			
(AD,01)			(C,200)
200	(IS,04)	1	(L,01)
201	(IS,05)	1	(S,01)
202	(IS,04)	1	(S,01)
203	(IS,04)	3	(S,03)
204	(IS,01)	3	(L,02)
	(AD,05)		
205		005	
206		001	
207	(IS,02)	1	(L,03)
	(AD,03)	(S,02)+(C,1))
203	(IS,03)	3	(S,03)
204	(DL,02)	(C,01)	
	(AD,04)	(S,02)	
205	(DL,02)	(C,01)	
	(AD,02)		
206		001	

Assembler Pass I

INTERMEDIATE CODE			
(AD,01)		(C,200)	
200	(IS,04)	1	(L,01)
201	(IS,05)	1	(S,01)
202	(IS,04)	1	(S,01)
203	(IS,04)	3	(S,03)
204	(IS,01)	3	(L,02)
(AD,05)			
205		005	
206		001	
207	(IS,02)	1	(L,03)
	(AD,03)	(S,02)+(C,1))
203	(IS,03)	3	(S,03)
204	(DL,02)	(C,02)	
	(AD,04)	(S,02)	
206	(DL,02)	(C,02)	
	(AD,02)		
207		001	

SYMBOL TABLE			
INDEX	SYMBOL	ADDRESS	LENGTH
01	A	204	2
02	LOOP	202	1
03	B	205	1
04	NEXT1	207	1
05	BACK	202	1

LITERAL TABLE		
INDEX	LITERAL	ADDRESS
01	='5'	205
02	='1'	206
03	='1'	207

POOL TABLE	
LIT_INDEX	
#01	
#03	

Assembler Examples Convert Source code into Pass I & Pass II

	START	100	
	MOVER	AREG,	='5'
	ADD	CREG,	='1'
	A	DS	3
L1	MOVER	AREG,	B
	ADD	AREG,	C
	MOPVEM	AREG,	D
	LTORG		
	D	EQU	A+1
L2	PRINT	D	
	ORIGIN	A-1	
	SUB	AREG,	='1'
	MULT	CREG,	B
	C	DS	'5'
	ORIGIN	L2+1	
	STOP		
	B	DC	'19'
	END		

EXAMPLE 2			
SIMPLE	START	100	
	BALR	15,	0
	USING	*,	15
LOOP	L	R1,	TWO
	A	R1,	FOUR
	ST	R1,	FOUR
	CLI	FOUR+3,	4
	BNE	LOOP	
	BR	14	
	R1	EQU	1
	TWO	DC	F'2'
	FOUR	DS	F
	END		

ASSUME:-BALR & BR are of two bytes....
 L,A,ST,CLI,BNE are of four byte instructions...

Assembler Pass II

INTERMEDIATE CODE			
(AD,01)		(C,200)	
200	(IS,04)	1	(L,01)
201	(IS,05)	1	(S,01)
202	(IS,04)	1	(S,01)
203	(IS,04)	3	(S,03)
204	(IS,01)	3	(L,02)
(AD,05)			
205		005	
206		001	
207	(IS,02)	1	(L,03)
(AD,03)		(S,02)+(C,1)	
203	(IS,03)	3	(S,03)
204	(DL,02)	(C,02)	
(AD,04)		(S,02)	
206	(DL,02)	(C,02)	
(AD,02)			
207		001	

TARGET CODE			
200	04	1	205
201	05	1	204
202	04	1	204
203	04	3	205
204	01	3	206
205		005	
206		001	
207	02	1	207
203	03	3	205
204			
206			
207		001	

Assembler Examples

SOURCE PROGRAM		
START	1000	
READ	N	
MOVER	B,	='1'
MOVEM	B,	TERM
AGAIN	MULT	B,
		TERM
MOVER	C,	TERM
ADD	C,	='1'
MOVEM	C,	TERM
COMP	C,	N
BC	LE,	AGAIN
MOVEM	B,	RESULT
LTORG		
PRINT		RESULT
STOP		
N	DS	1
RESUL	DS	20
T		
TERM	DS	1
END		

INTERMEDIATE CODE		
(AD,01)		(C,1000)
1000	(IS,09)	(S,01)
1001	(IS,04)	2 (L,01)
1002	(IS,05)	2 (S,02)
1003	(IS,03)	3 (S,02)
1004	(IS,05)	3 (S,02)
1005	(IS,01)	3 (L,01)
1006	(IS,05)	3 (S,02)
1007	(IS,06)	3 (S,01)
1008	(IS,07)	2 (S,03)
1009	(IS,05)	2 (S,04)
	()AD,05)	
1010		(L,01)
1011	(IS,10)	(S,04)
1012	(IS,00)	
1013	(DL,02)	(C,01)
1014	(DL,02)	(C,20)
1034	(DL,02)	(C,01)
	(AD,02)	