TUTORIAL 6

Page No. 31
Date 15 02 22

U19CS012

1.> Explain two-pass assembler functions with example program.

1.> ① The two-pass assembler scans the input assembly language twice.

② These scans are normally called Pass I and Pass II.

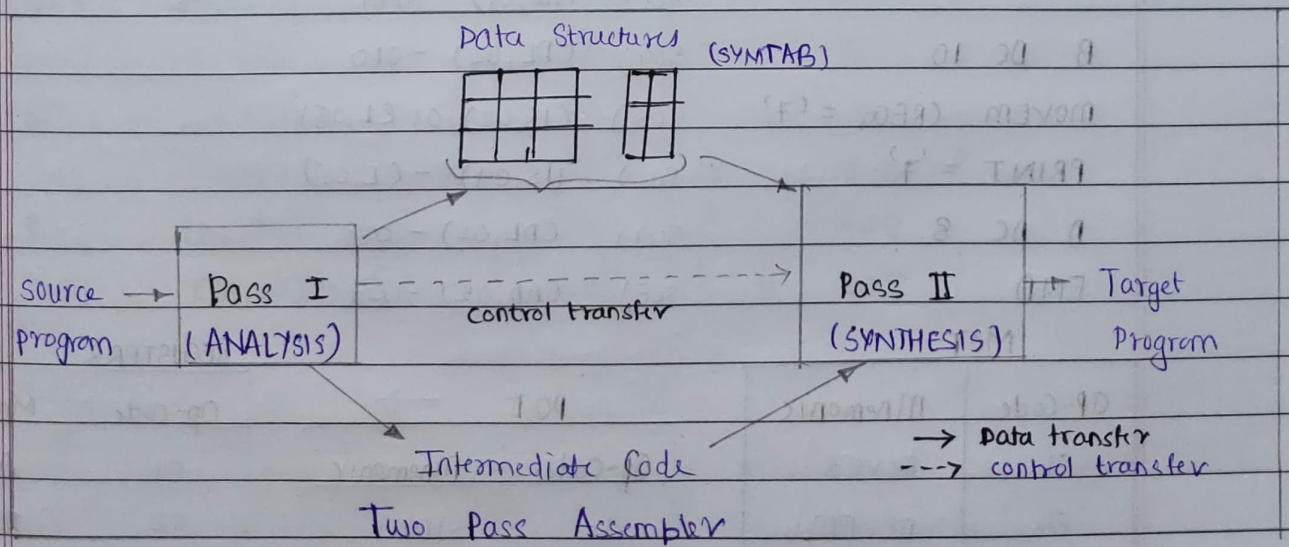
(ASL → IC) IC → machine code)

[ASL = Assembly Level Program; I.C. = Intermediate Code

Tasks performed by the passes of two pass assembler are as follows:

Pass 1: ① Seperate the symbol, mnemonic opcode and operand fields.
② Build the symbol table
③ Perform the LC (location counter) Processing
④ Construct Intermediate representation (or IC)

Pass 2: ① Synthesize the target program.



- It can handle forward refrances easily.

UI9CS012

Example of Two-Pass Assembler Program

Source Program

	LC	
START 100		(AD, 01) - (C, 100)
MOVER AREG, A	100)	(IS, 01) 01 (S, 01)
LOOP: PRINT B	101)	(IS, 09) - (S, 03)
ADD BREG, ='9'	102)	(IS, 03) 02 (L, 1)
SUB BREG, D	103)	(IS, 04) 02 (S, 04)
COMP CREG, ='23'	104)	(IS, 08) 03 (L, 02)
LTORG	105)	(AD, 05) - 009
	106)	(AD, 05) - 023
A DS 3	107)	(DL, 01) - 03
LABEL: EQU LOOP		
ORIGIN 500		
L1: MULT CREG, ='7'	500)	(IS, 05) 03 (L, 03)
SUB BREG, ='93'	501)	(IS, 04) 02 (L, 04)
LTORG	502)	(AD, 05) - 007
	503)	(AD, 05) - 093
B DC 10	504)	(DL, 02) - 010
MOVEM CREG, ='7'	505)	(IS, 02) 03 (L, 05)
PRINT ='7'	506)	(IS, 09) - (L, 05)
D DC 8	507)	(DL, 02) - 008
END	508)	(AD, 02) - 007

(Symbol table)

Sym-no	Symbol	Address
1	A	107
2	LOOP	101
3	B	504
4	D	507
5	LABEL	101
6	L1	500

(Literal table)

Lit-no	Literal	Address
1	='9'	105
2	='23'	106
3	='7'	502
4	='93'	503
5	='7'	508

MOT

Op-Code	Mnemonic
01	MOVER
02	MOVEM
03	ADD
04	SUB
05	MULT
06	DIV
07	BC
08	COMP

09 - PRINT
10 - READ

POT

Op-Code	Mnemonic
01	START
02	END
03	EQU
04	ORIGIN
05	LTORG
DL	01 DS
	02 DC

REGISTERS

Op-code	Mnemonics
01	AREG
02	BREG
03	CREG
04	DREG

U19CS012

Pass II →

	Intermediate Code	Target Code
	(AD, 01) - (C, 100)	01 - 100
100)	(IS, 01) 01 (S, 01)	100) 01 01 107
101)	(IS, 09) - (S, 03)	101) 09 - 504
102)	(IS, 03) 02 (L, 1)	102) 03 02 105
103)	(IS, 04) 02 (S, 04)	103) 04 02 507
104)	(IS, 08) 03 (L, 02)	104) 08 03 106
105)	(AD, 05) - 009	105) - - 009
106)	(AD, 05) - 023	106) - - 023
107)	(DL, 01) - 03	107) - - -
500)	(IS, 05) 03 (L, 03)	500) 05 03 502
501)	(IS, 04) 02 (L, 04)	501) 04 02 503
502)	(AD, 05) - 007	502) - - 007
503)	(AD, 05) - 093	503) - - 093
504)	(DL, 02) - 010	504) - - 010
505)	(IS, 02) 03 (L, 05)	505) 02 03 508
506)	(IS, 09) - (L, 05)	506) 09 - 508
507)	(DL, 02) - 008	507) - - 008
508)	(AD, 02) - 007	508) - - 007

output of pass-II
(Target Code / Object Code)

U19CS012

2.) What are some advantages of assembly languages over high level languages?

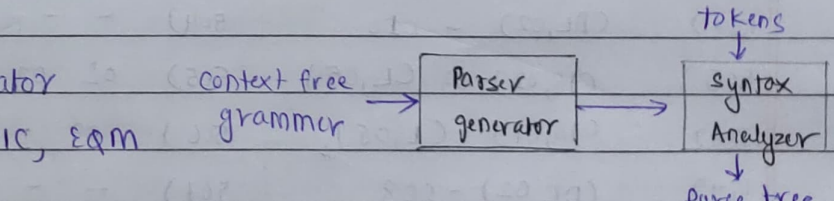
2.) Advantages of Assembly Languages (ARM, MIPS, x86) over High Level Languages (C, C++, Fortran, Prolog)

① Extremely Fast.
② Second closest programming language to hardware components
③ minamalistic syntax, just few mnemonic keywords (except binary)
④ It is memory effecient, as it requires less memory.
⑤ Supports low-level operation & access machine-dependent registers & I/O
⑥ Helps to build compilers
⑦ You can get access to unusual programming modes of processor (16 bit mode to interface startup, firmware or legacy code in Intel PC)
⑧ You can break the conventions of usual compiler, which might allow some optimization (like temporarily breaking rules about memory allocation, threading, calling convention)

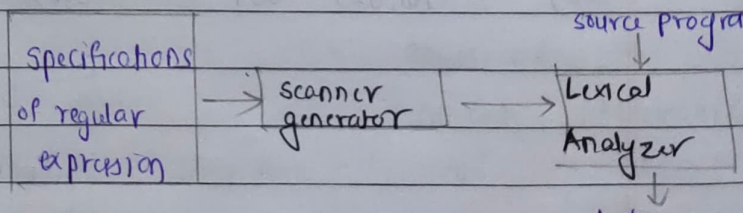
3.) What tools are used for compiler construction?

3.)

① Parser Generator
Eg: YACC, PIC, EQN



② Scanner Generator



③ Syntax directed translation engine
④ Automatic code generators
⑤ Data-Flow Analysis Engine
⑥ Compiler Construction Toolkits

4.> What are Applications of Compiler? Explain.

4.> Applications of Compiler are:

① Full Implementation of High Level Programming Languages
② Support optimization for computer Architecture Parallelism.
③ Design a New memory Hierarchies of Machine.
④ Widely used for Translating Programe.
⑤ Used with other Software Productivity Tools.

5.> Differentiate between a macro and a subroutine. And explain macro definition and macro Expansion using an example.

5.>

MACRO	SUB ROUTINE
① Macro can called only in the program it is defined	① Sub-routine can be called from other programs also.
② Macro can have max 9 parameter	② can have any no. of parameters
③ macro can be called only after its definition.	③ This is not true for sub-routine.
④ Macro is used when same thing is to be done in a program a number of times.	④ Sub-routines is used for modularization.
⑤ Called by its name	⑤ A subroutine is called by BSR or JSR instruction.
⑥ Simple to write and use	⑥ subroutines are more complex (stacks are used)
⑦ Macros are faster than subroutines (no overheads, no saving of return addresses	⑦ Slower

U19CS012

Macro Definition: A macro definition is enclosed between a macro header statement and a macro end statement.

① Macro definations are typically at start of program.

② A macro definition consist of

ⓐ Macro Prototype Statement - declares name of macro, name & kind of params

ⓑ one or more model statement - Assembly Level statement generated during macro expansion

ⓒ Macro preprocessor statement - Perform Auxillary function

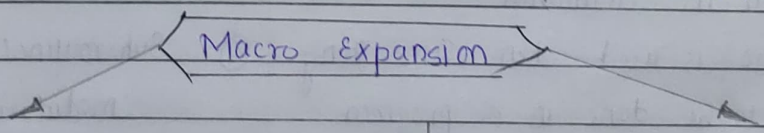
Example of definition of macro INCR

```
macro header → MACRO
               INCR    &MEM_VAL, &INCR_VAL, &REG
               MOVER   &REG, &MEM_VAL
               ADD     &REG, &INCR_VAL
               MOVEM   &REG, &MEM_VAL
macro end   → MEND
```

Macro Expansion

① A macro call leads to macro expansion

② During macro expansion, Macro call stmt replaced sequence of assembly stmts



Lexical Expansion

① replacement of character string by another character string during program execution.

② typically employed to replace occurance of formal parameters by corresponding actual parameters

Semantic Expansion

① generation of instruction tailored to requirement of specific usage

② Eg: generation of type specific instruction for manipulation of byte and word operands.