15/02/22

Tutorial - 06

SS

U19CS009

Q-1 Explain two pass assembler functions with example

Ans → Pass 2 assembler requires 2 scans of program to generate machine code

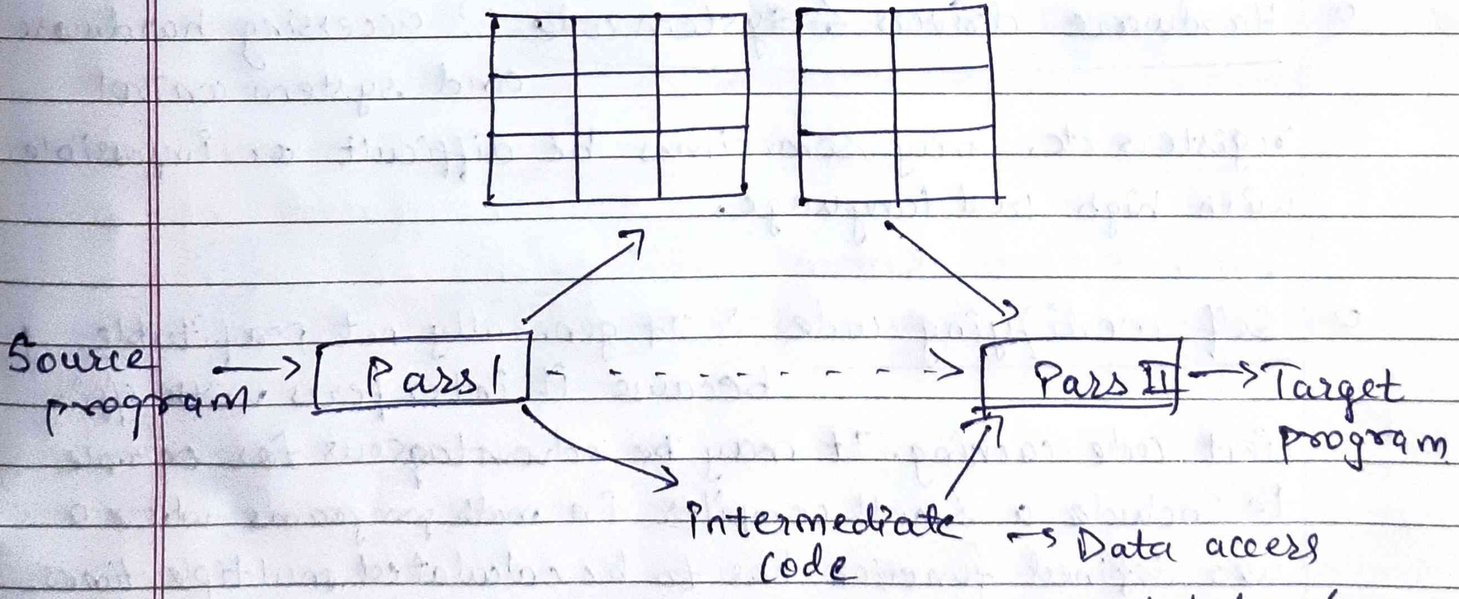
→ It uses data structure defined by pass1 like symbol table

Functioning of two pass assembler

→ Pass 1
- → separate symbol, mnemonic opcode, and operand fields
- → build symbol table
- → perform LC processing
- → Construct intermediate representation (IC)

→ Pass 2
- → synthesis targer program.

Data structure



→ Data access

→ control transfer

→ Eg:-

	Source code	LC	IC	By Pass 1
	START 100		(AD,01)	(C,100)
	MOVER AREG, X	100	(IS,04)	(RG,01) (~~G~~S,0)
L1	ADD BREG, ONE	101	(IS,01)	(RG,02) (S,1)

Pass 1

				IC		
	ADD	CREG, TEN	102	(IS,01)	(RG,03)	(S,2)
	STOP		103	(IS,00)		
X	DC	'5'	104	(DL,01)	(C,5)	
ONE	DC	'1'	105	(DL,01)	(C,1)	
TEN	DL	'10'	106	(DL,01)	(C,10)	
	END			(AD,02)		

SYMTAB =>

INDEX	SYMBOLNAME	Address
0	X	104
1	ONE	105
2	TEN	106

→ Machine code by Pass 2

LC	IC by Pass 1	machine code
~~LC~~	(AD,01) (C,100)	
100	(IS,04) (RG,01) (S,0)	04 01 104
101	(IS,01) (RG,02) (S,1)	01 02 105
102	(IS,01) (RG,03) (S,1)	01 03 106
103	(IS,00)	00 00 000
104	(DL,01) (C,5)	
105	(DL,01) (C,1)	
106	(DL,01) (C,10)	
	(AD,02)	

→ They don't change in forming machine code

S → stands for symbol.
C → stands for constant.

Q-2 What are some advantages of ~~machine~~ assembly languages over high level language?

Ans

- Debugging and verifying → looking at compiler generated assembly code or the disassembly window in a debugger is useful for finding errors and for checking how well a compiler optimizes a particular piece of code.

- Making compilers → understanding assembly coding is must for making compilers, debuggers and other development tools.

- Embedded System → small embedded system has fewer resources. Assemby programming is necessary for optimizing code for speed or size.

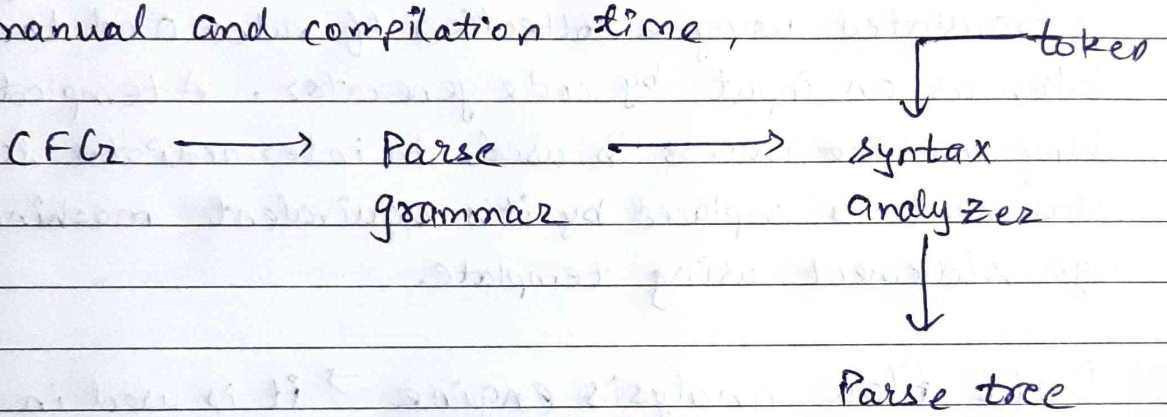
- Hardware drivers 4 system code → accessing hardware and system control registers etc. may sometimes be difficult or impossible with high level language.

- Self-modifying code → If generally not profitable because it interferes with efficient code caching. It may be advantageous for example to include a small compiler in math programs where a user defined function has to be calculated multiple times.

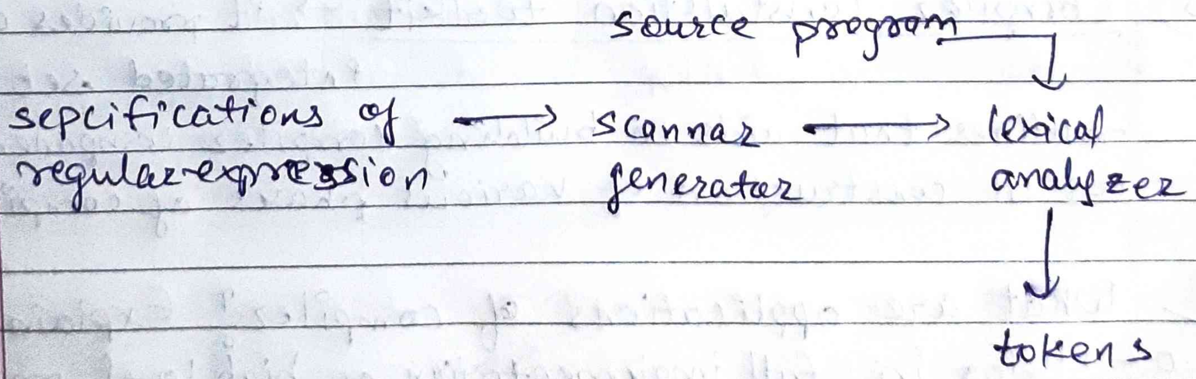
Page No.:
Date : / /

Q3 What tools are used for compiler construction?

Ans ① **Parser Generator** -> it provides syntax analyzers from the input that is based on a grammatical description of programming language or on a context-free grammar. It is useful as the syntax analyzer phase is highly complex and consumes more manual and compilation time,



② **Scanner Generator** -> it generates lexical analyzer from input that consists of regular expression description based on tokens of a language. It generates a finite automation to recognize regular expression.



③ **Syntax directed translation engines** -> it generates intermediate code with three address formats from the input that consists of a parse tree. These engines have routines to traverse the parse tree and then produces the intermediate code. In this, each node of parse tree is associated with one or more translations.

(4) Automatic code generator → It generates machine language for a target machine. Each operation of intermediate language is translated using a collection of rules and then is taken as an input by code generator. A template matching ~~machine~~ process is used. An intermediate language statement is replaced by it's equivalent machine language statement using template.

(5) Data flow analysis engine → It is used in code optimization. Data flow analysis is key part of the code optimization that gather the Information that is the value that flow from one part of a program to another.

(6) Compiler Construction toolkit → It provides an Integrated set of routines that aids in building compiler components or in construction of various phases of compilers.

Qs-4 What are applications of compiler? Explain

Ans
- → Helps in full implementation of high level programming languages
- → Supports optimization for computer architecture parallelism.
- → Design of new memory - hierarchy of machines
- → Translating programs
- → used with other software productivity tools.

Q-5 D/B a macro and subroutine. And explain macro definition and expansion using an example

Ans
- Both permit a group of instructions to be defined as a single entity with a unique given label or name called up when needed.
- A subroutine is called by BSR or JSR instruction, while a macro is called by simply using it's name.
- Macro are simpler to write and use
- Macro are faster
- Macro are not substitute of subroutine → as macro is substituted with the code and additional code is generated every time a macro is called. Very long macro that are used many times in a program will result in an enormous expansion of code size.
- Macro can be called only in program it is defined, subroutine can be called from other program also.

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

→ macro definitions are typically located at the start of the program.

→ Macro definition consists of

- macro prototype → declares name of macro & kind of parameter
- one or more model → from which assembly language is ~~genrate~~ generated
- preprocessor statements
  ↳ used to perform auxillary function during macro expansion.

- macro expansion
  <macro name> [<formal parameter space>[....]]

- <formal parameter space>'s of form

<parameter name> [<parameter kind>]

→ ## Macro Expansion

→ a macro call leads to macro expansion, in which macro call statement is replaced by a sequence of assembly statement.

→ (g)

```
INCR        A, B, AREG
MACRO CALL
```

```
MACRO
INCR     &MEM_VAL    &INC_VAL  ,  &REG

MOVER    &REG        &MEM_VAL
ADD      &REG        &INC_VAL
MOVEM    &REG        &MEM_VAL
MEND
```

MACRO

```
+     MOVER    AREG    A
+     ADD      AREG    B
+     MOVEM    AREG    A
```