B. Tech DEGREE EXAMINATION, JANUARY 2023

Fifth Semester

Computer Science and Engineering

LANGUAGE TRANSLATORS

(2013 – 14 Regulations)

ANSWER KEY

PART A

1. **Differentiate assembler and interpreter.(CT1,M)**

Assembler	Interpreter
The Assembler function is to translate source program into object program. The translation of source program to object code requires accomplishing the following function.	Interpreter is a set of programs which converts high level language program to machine language program line by line.

2. **List the types of addressing modes.(CT1, M)**
   - Base relative addressing:
   - PC relative addressing
   - Direct addressing modes
   - Indexed addressing modes

3. **Define linkage editors.**
   It produces a linked version of the program which is normally written to a file or library for later execution.

4. **What is meant by dynamic linking?**
   The process of loading the external shared libraries into the program and the bind those shared libraries dynamically to the program.

5. **Define compiler.(CT2)**
   Compiler is a set of program which converts the whole high level language program to machine language program.

6. **State input buffering(CT2, M)**
   - The lexical analyser scans the input string from left to right character at a time
   - It uses two pointer begin pointer and forward pointer.

7. **Define Grammar.**
   It is defined as four tuples – G=(V, T, P, S) G is a grammar , which consists of a set of production rules .

8. What is procedure calls?

A procedure call is frequently used programming construct for a compiler . it is used to generate good code for procedure calls and returns.

9. What is DAG?(M)

Directed acyclic graph (DAG) is a useful data structure for implementing transformations on basic blocks.

10. Write some code optimization techniques.

Local optimization
Loop optimization
Data flow analysis

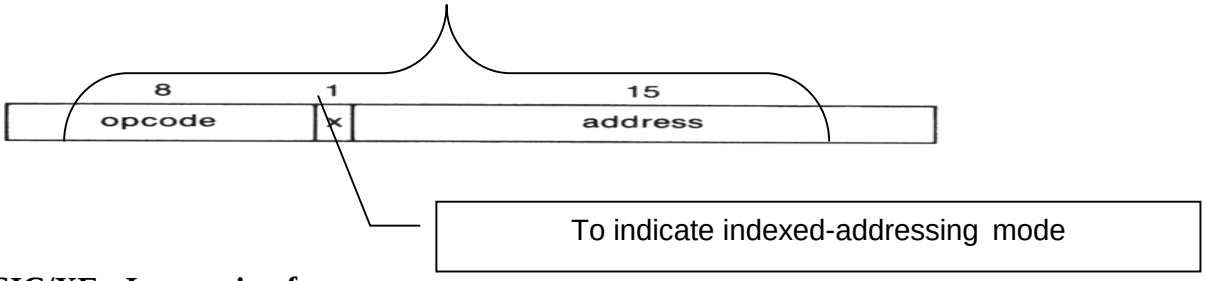
PART B

Unit - I

11. Illustrate various types of instruction set and addressing modes with an example.(CT1, M)

SIC - Instruction formats :

All machine instructions on the standard SIC have the following 24-bit format



SIC/XE - Instruction formats

Maximum memory in SIC/XE system is 1 megabyte(2^{20} bytes),address will no longer fit into a 15-bit field so, the instruction format used on the standard version of SIC is not suitable (SIC Instruction format uses 15-bit address, SIC/XE require 20-bit address)

Addressing modes:

Addressing Modes

There are two addressing modes available, which are as shown in the below table. Parentheses are used to indicate the contents of a register or a memory location.

Mode	Indication	Target address calculation
Direct	x = 0	TA = address
Indexed	x = 1	TA = address + (X)

( ): Contents of a register or a memory location

12. Compare one pass assembler and multi-pass assembler .

PASS1

- As each literal operand is recognized during pass1 the assembler reaches the LITTAB specifies literal name.
- If the literal is already present no action,otherwise it is added to the LITTAB.

Multi pass

- Multi-pass assembler can make as many passes as are needed to process the definition are symbols.
- It is not necessary for an assembler to make more than two passes over th entire program instead the portion of the program that has forward reference is stored using pass 1.
- Additional passes are made only through the stored definitions.
- This process is followed by a normal pass 2.

To solve forward reference problem

- Store symbols that involve forward reference in the symbol table.
- This symbol table also indicates which symbols are dependent on the value of others.

Eg: sequence of statements that involves forward reference.

```
1.  :      HALFSZ      EQU   MAXLEN/2
2.  :      MAXLEN      EQU   BUFFEND-BUFFER
    :
    :
    :
3.  1034   BUFFFER      RESB  4096
4.  2034   BUFFEND      EQU   *
```

Unit - II

13**.Discuss in detail about machine dependent and machine independent loader functions(CT1)**

MACHINE-INDEPENDENT LOADER FEATURES

- Loading and linking are often thought of as OS service functions. Therefore, most loaders include fewer different features than are found in a typical assembler.
- They include the use of an automatic library search process for handling external reference and some common options that can be selected at the time of loading and linking.

Automatic Library Search

- Many linking loaders can automatically incorporate routines from a subprogram library into the program being loaded.
- Linking loaders that support automatic library search must keep track of external that are referred to, but not defined, in the primary input to the loader.

MACHINE-DEPENDENT LOADER FEATURES

- The absolute loader has several potential disadvantages. One of the most obvious is the need for the programmer to specify the actual address at which it will be loaded into memory.
- On a simple computer with a small memory the actual address at which the program will be loaded can be specified easily.

14. Write notes on dynamic linking and bootstrap loaders.(M)
dynamic linking

A linking loader performs all linking and relocation operations, including automatic library search if specified, and loads the linked program directly into memory for execution.

bootstrap loaders.

With the machine empty and idle there is no need for program relocation. ¬ We can specify the absolute address for whatever program is first loaded and this will be the OS, which occupies a predefined location in memory.

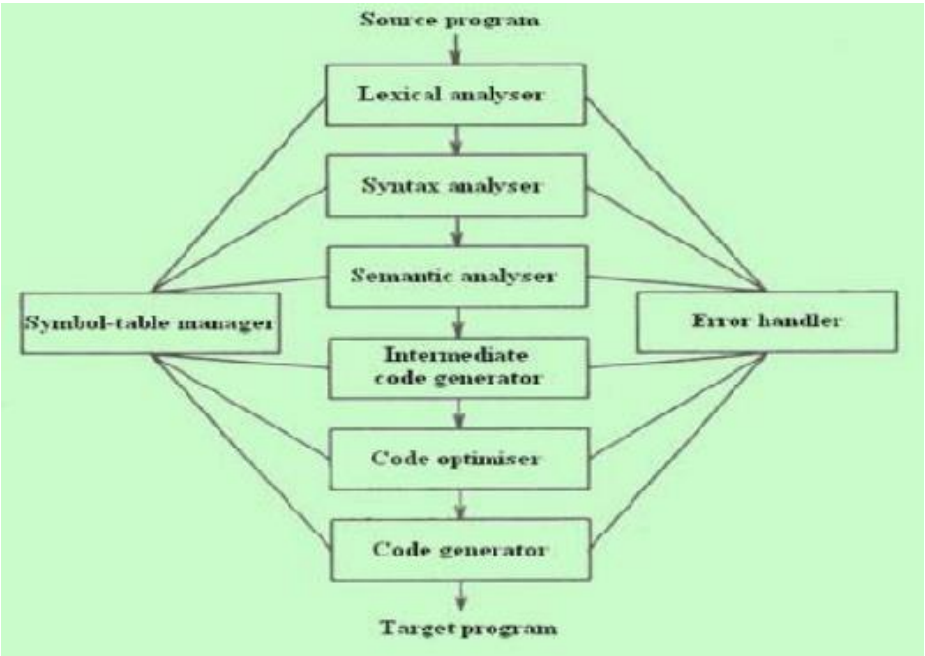
1. To have the operator enter into memory the object code for an absolute loader, using switches on the computer console.

2. To have the absolute loader program permanently resident in a ROM.

3. To have a built –in hardware function that reads a fixed –length record from some device into memory at a fixed location

Unit-III

15. What are the various phases of a compilers? Example each phase in detail by using the input "Position = initial + rate *60"(CT2, M)



Position: = initial + rate * 60

temp1:= inttoreal (60)
temp2:= id3 * temp1
temp3:= id2 + temp2
id1:= temp3

16. Describe in detail about specification of Tokens and recognition of Tokens.(CT2, M)

Specification of Tokens

Regular expressions are an important notation for specifying lexeme patterns.

RECOGNIZATION OF THE TOKENS

The tokens obtained during lexical analysis are recognized using a finite automaton

Finite automata

1. Finite automata are recognizers; they simply say "yes" or "no" about each possible input string.

2. Finite automata come in two flavors:

(a) Nondeterministic finite automata (NFA) have no restrictions on the labels of their edges. A symbol can label several edges out of the same state, and E, the empty string, is a possible label.

(b) Deterministic finite automata (DFA) have, for each state, and for each symbol of its input alphabet exactly one edge with that symbol leaving that state

Unit-IV

17. Construct Predictive parsing table for the following grammar and check the input string (CT2)

W=id + id * id
E->E+T / T ; T->*F;
F->(E) /id

FIRST(E) = FIRST(T) = FIRST(F) = {(, id}

FIRST(E') = {+, ϵ}

FIRST(T') = {*, ϵ}

FOLLOW(E) = FOLLOW(E') = {), $}

FOLLOW(T) = FOLLOW(T') = {+,), $}

FOLLOW(F) = {+, *,), $}

Parsing table M for grammar

NONTER-	INPUT SYMBOL					
MINAL	id	+	*	(	)	$
E	$E \to TE'$			$E \to TE'$		
E'		$E' \to +TE'$			$E' \to \epsilon$	$E' \to \epsilon$
T	$T \to FT'$			$T \to FT'$		
T'		$T' \to \epsilon$	$T' \to *FT'$		$T' \to \epsilon$	$T' \to \epsilon$
F	$F \to$ id			$F \to (E)$		

Moves made by predictive parser on input id+id*id.

STACK	INPUT	OUTPUT
$$E$	id + id * id$	
$$E'T$	id + id * id$	$E \to TE'$
$$E'T'F$	id + id * id$	$T \to FT'$
$$E'T'$id	id + id * id$	$F \to$ id
$$E'T'$	+ id * id$	
$$E'$	+ id * id$	$T' \to \epsilon$
$$E'T+$	+ id * id$	$E' \to +TE'$
$$E'T$	id * id$	
$$E'T'F$	id * id$	$T \to FT'$
$$E'T'$id	id * id$	$F \to$ id
$$E'T'$	* id$	
$$E'T'F*$	* id$	$T' \to *FT'$
$$E'T'F$	id$	
$$E'T'$id	id$	$F \to$ id
$$E'T'$	$	
$$E'$	$	$T' \to \epsilon$
$	$	$E' \to \epsilon$

18. Write notes on back patching and procedure calls.

Back Patching

- To implementing syntax-directed definitions, compute the translations given in the definition.
- To generating three address codes in a single pass for Boolean expressions and flow of control statements is that we may not know the labels that control must go to at the time jump statements are generated.

➢ To manipulate list if labels ,we use three functions:

❖ **makelist(i) --** creates a new list containing only i, an index into the array of quadruples and returns pointer to the list it has made.

❖ **merge(i,j) –** concatenates the lists pointed to by i and j ,and returns a pointer to the concatenated list.

❖ **backpatch(p,i)** – inserts i as the target label for each of the statements on the list pointed to by p.

Procedure calls

Simple procedure call statement

S→ call id (elist)

elist→ elist, E

elist → E

Translation includes

- Calling sequence → actions taken on entry to and exit from each procedure.
- Arguments are evaluated and put in a known places(return address) location to which the called routine must transfer after it is finished.
- Static allocation→ return address is placed after code sequence itself.

Unit- V

19. Explain the issues in the design of code generator.

Issues in the design of code generator are:

- Input to the code generator
- Target programs
- Memory management
- Instruction selection
- Register allocation
- Choice of evaluation order
- Approaches to code generation.

INPUT TO THE CODE GENERATOR

The input to the code generator consists of the intermediate representation of the source program produced by the front end, together with information in the symbol table that is used to determine the run time addresses of the data objects denoted by the names in the intermediate representation.

TARGET PROGRAMS

The output of the code generator is the target program. The output may take on a variety of forms: absolute machine language, relocatable machine language, or assembly language.

MEMORY MANAGEMENT

Mapping names in the source program to addresses of data objects in run time memory is done cooperatively by the front end and the code generator.

INSTRUCTION SELECTION

The nature of the instruction set of the target machine determines the difficulty of instruction selection. The uniformity and completeness of the instruction set are important factors.

THE TARGET MACHINE

Familiarity with the target machine and its instruction set is prerequisite for designing a good code generator.

REGISTER ALLOCATION

During which we select the set of variable that will resides in register at a point in the program.

20. Discuss in detail about DAG representation of basic blocks with an example.

We construct a DAG for a basic block as follows:

1. There is a node in the DAG for each of the initial values of the variables appearing in the basic block.
2. There is a node N associated with each statement s within the block. The children of N are those nodes corresponding to statements that are the last definitions, prior to s, of the operands used by s.
3. Node N is labeled by the operator applied at s, and also attached to N is the list of variables for which it is the last definition within the block.
4. Certain nodes are designated output nodes. These are the nodes whose variables are live on exit from the block; that is, their values may be used later, in another block of the flow graph. Calculation of these "live variables" is a matter for global flow analysis.

The DAG representation of a basic block lets us perform several code improving transformations on the code represented by the block.

a) We can eliminate local common sub expressions, that is , instructions that compute a value that has already been computed.
b) b) We can eliminate dead code, that is , instructions that compute a value that is never used.
c) c) We can reorder statements that do not depend on one another; such reordering may reduce the time a temporary value needs to be preserved in a register.

d) d) We can apply algebraic laws to reorder operands of three-address instructions, and sometimes thereby simplify the computation.

Example : x= y+ z

a = y + z