SRM

INSTITUTE OF SCIENCE & TECHNOLOGY
(Deemed to be University u/s 3 of UGC Act, 1956)

SRM Institute of Science and Technology

Course Code & Title: 18CSC304J & COMPILER DESIGN **Duration: 2 periods**

Year & Sem: III Year /VI Sem **Max. Marks: 50**

	Part A	**Marks**	**BL**	**CO**	**PO**	**PI Code**
1	How many temporary variables are required to express the following statement in three address code? if(a+b*h>a*b +h) A=68 a) 3 b) 5 c) 6 d) 4 And: d	1	4	4	3	**3.6.1**
2	Consider the following translation scheme. S -> ER R -> * E{print(' * '); R | f E -> F + E{print(' + '); | F F -> (S) | id{print(id.value);} Here id is a taken that represents an integer and id. value represents the corresponding integer value. For an input '2 * 3 + 4', this translation scheme prints? a) 2 * 3 + 4 b) b) 2 * + 3 4 c) 2 3 * 4 + d) 2 3 4 + * And: d	1	3	4	3	**3.6.1**

3	How many basic blocks are there in the following code snippet?
	x=8;
	y=9;
	z=0;
	w=1;
	L1:
	if (x>6) {
	if(y>5) {
	z=x+y;
	z=37;
	w=x+y;
	x=0;
	Goto L1;
	} }
	
	a) 3
	b) 4
	c) 5
	d) 6
	
	Ans: B
		1	3	4	3	**3.6.1**

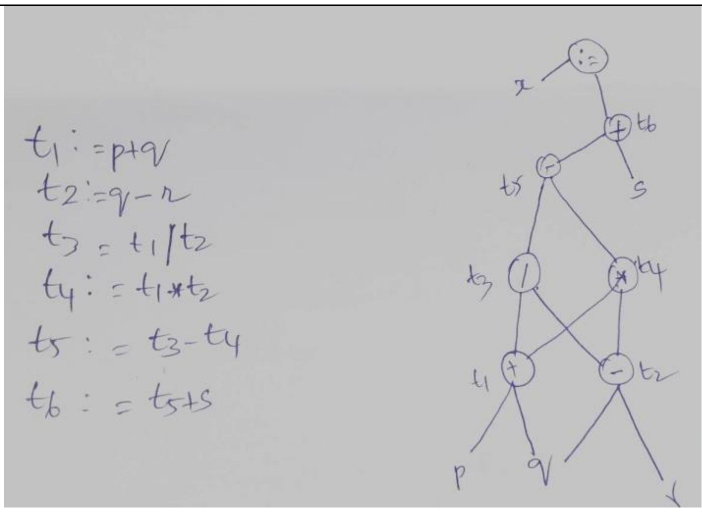
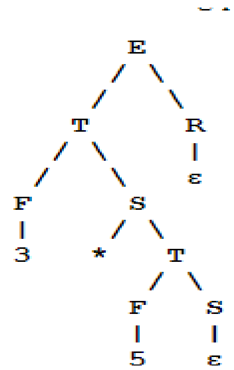
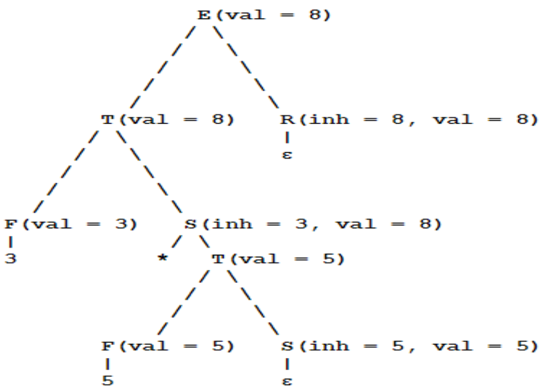
4	What does the following code print? Assume that the execution begins at Procedure Z. procedure A print x procedure B int x = 6 call A procedure C int x = 2	1	3	4	3	**3.6.1**

	print x call B procedure Z int x = 9; call A call B call C call A a) 9 6 2 6 9 b) 9 2 6 9 9 c) 9 6 6 2 9 d) 9 6 6 6 9 **Ans: a**					

5	Which of the following has lower instruction costs?
	a) ADD R1, #5
	SUB 4(R1), *1(R0)
	b) MOV R0, #78
	ADD *R0, *R1
	c) MOV b, a
	ADD c, a
	d) All have equal costs
	
	Ans: b
		1	2	4	1	**1.7.1**

6	Which is not a NP complete problem? a) Allocation of registers b) Order of evaluation c) Instruction selection d) Both a and b Ans: c	1	2	5	1	**1.6.1**
7	Which optimization techniques is used to reduce multiple jumps? A.Latter optimization technique B.Peephole optimization technique C. Local optimization technique D. Code optimization technique Ans B	1	1	5	2	**2.7.1**
8	Contiguous memory allocation is possible only in a) Heap b) Heap and stack c) Static and stack d) Static and heap And: c	1	3	5	1	**1.7.1**
9	Consider the following three address code. Identify the CORRECT collection of different optimization can be performed? m = 3 j = n v = 2 * n limit = integer n / 2 L1: j = j – 1 t4 = 4 * j t5 = a[t4] if t5 > limit – v goto L1 A. Code Motion, Constant Folding, Induction Variable Elimination, Reduction in Strength B. Copy Propagation , Code Motion, Deadcode Elimination, Reduction in Strength C. Constant Folding, Copy Propagation, Deadcode Elimination, Reduction in Strength D. Code Motion, Constant Folding, Copy Propagation, Induction Variable Elimination	1	2	5	1	**1.6.1**

	Ans: A					
10	Consider the following statements S1: Static allocation bindings do not change at runtime S2: Heap allocation allocates and de-allocates storage at run time Which of the following statements is/are true? a) S1 is true and S2 is false b)S2 is true and S1 is false c) Both S1 and S2 are true d) Both S1 and S2 are false **Ans: C**	1	2	5	1	**1.6.1**
	Part B					
11	State various methods of implementing three address statements? Three Address Code (2) Three address code is a sort of intermediate code that is simple to create and convert to machine code. It can only define an expression with three addresses and one operator. Basically, the three address codes help in determining the sequence in which operations are action by the compiler. Pointers for Three Address Code • Three-address code is considered as an intermediate code and utilised by optimising compilers. • In the three-address code, the given expression is broken down into multiple guidelines. These instructions translate to assembly language with ease. • Three operands are required for each of the three address code instructions. It's a binary operator and an assignment combined. There are representations of three address codes, namely 1. Quadruple 2. Triples 3. Indirect Triples Explanation with Example (2)	4	2	4	1	**1.6.1**
12	Translate the conditional statement if a< b then 1 else 0 into three address code Three Address Code for the given expression is- (1) If (A < B) goto (4) (2) T1 = 0 (3) goto (5)	4	3	4	1	**1.6.1**
13	Illustrate Peephole optimization with suitable Examples Peephole Optimization Techniques **A.** Redundant load and store elimination: In this technique, redundancy is eliminated **B.** Constant folding: The code that can be simplified by the user itself, is simplified. **C.** Strength Reduction: The operators that consume higher execution time are replaced by the operators consuming less execution time. **D.** Combine operations: Several operations are replaced by a single equivalent operation. **E.** Dead code Elimination: A part of the code which can never be executed, eliminating it will improve processing time and reduces set of instruction.	4	3	4	1	**1.6.1**
14	Develop a DAG and optimal target code for the expression. x = ((p + q) / (q-r)) – (p + q) * (q-r) +s	4	3	5	2	**1.6.1**

						
15	Illustrate annotated parse tree with synchronized and inherited attribute for expression 3*5 for the given grammar. E -> TR T -> FS F -> n S -> *T|ε R -> ε  Here is the tree with attributes shown.  (2)	4	3	5	1	**1.6.1**

Production	Semantic rule
$E \rightarrow T\ R$	$E.\text{val} = R.\text{val}$ $R.\text{inh} = T.\text{val}$
$R \rightarrow \varepsilon$	$R.\text{val} = R.\text{inh}$
$R \rightarrow +\ E$	$R.\text{val} = R.\text{inh} + E.\text{val}$
$T \rightarrow F\ S$	$T.\text{val} = S.\text{val}$ $S.\text{inh} = F.\text{val}$
$S \rightarrow \varepsilon$	$S.\text{val} = S.\text{inh}$
$S \rightarrow *\ T$	$S.\text{val} = S.\text{inh} * T.\text{val}$
$F \rightarrow \mathbf{n}$	$F.\text{val} = \mathbf{n}.\text{val}$
$F \rightarrow (\ E\ )$	$F.\text{val} = E.\text{val}$

(2)

Part B
(2*12=20)

16 Write quadruples, triples and indirect triples for the expression: -(a*b)+(c+d)-(a+b+c+d) and explain the sequences of code generation algorithm.

a. Write quadruples, triples and indirect triples for the expression: -(a*b)+(c+d)-(a+b+c+d) (9)

Sol:

First of all this statement will be converted into Three Address Code as−

t1 = a + b

t2 = −t1

t3 = c + d

t4 = t2 ∗ t3

t5 = t1 + c

t6 = t4 − t5

Quadruple

Location	Operator	arg 1	arg 2	Result
(0)	+	a	b	t1
(1)	−	t1		t2
(2)	+	c	d	t3
(3)	∗	t2	t3	t4
(4)	+	t1	c	t5
(5)	−	t4	t5	t6

12 2 4 2 **1.5.1**

Triple

Location	Operator	arg 1	arg 2
(0)	+	a	b
(1)	–	(0)	
(2)	+	c	d
(3)	*	(1)	(2)
(4)	+	(0)	c
(5)	–	(3)	(4)

Indirect Triple

Statement	
(0)	(11)
(1)	(12)
(2)	(13)
(3)	(14)
(4)	(15)
(5)	(16)

Location	Operator	arg 1	arg 2
(11)	+	a	b
(12)	-	(11)	
(13)	+	c	d
(14)	*	(12)	(13)
(15)	+	(11)	c
(16)	-	(14)	(15)

The sequences of code generation algorithm. (3)

The algorithm takes a sequence of three-address statements as input. For each three address statement of the form a:= b op c perform the various actions. These are as follows:

1. Invoke a function getreg to find out the location L where the result of computation b op c should be stored.
2. Consult the address description for y to determine y'. If the value of y currently in memory and register both then prefer the register y' . If the value of y is not already in L then generate the instruction **MOV y' , L** to place a copy of y in L.
3. Generate the instruction **OP z' , L** where z' is used to show the current location of z. if z is in both then prefer a register to a memory location. Update the address descriptor of x to indicate that x is in location L. If x is in L then update its descriptor and remove x from all other descriptor.
4. If the current value of y or z have no next uses or not live on exit from the block or in register then alter the register descriptor to indicate that after execution of x : = y op z those register will no longer contain y or z.

OR

17 (a)	State the syntax directed translation? How it is different from translation schemes? Explain with an example. syntax directed translation: A technique of compiler execution, where the source code translation is totally conducted by the parser, is known as syntax-directed translation. The parser primarily uses a Context-free-Grammar to check the input sequence and deliver output for the compiler's next stage. It is a kind of notation in which each production of Context-Free Grammar is related with a set of semantic rules or actions, and each grammar symbol is related to a set of Attributes. Thus, the grammar and the group of semantic Actions combine to make **syntax-directed definitions**. The translation may be the generation of intermediate code, object code, or adding the information in symbol table about constructs type. **Semantic Actions** – It is an action that is executed whenever the Parser will recognize the input string generated by context-free grammar. For Example, $A \rightarrow BC$ {Semantic Action} Semantic Action is written in curly braces Attached with a production. **In Top-Down Parser**, semantic action will be taken when A will be expanded to derive BC which will further derive string w. **In Bottom-Up Parser**, Semantic Action is generated when BC is reduced to A. **Semantic Action can perform –** • **Computation of value of variables** $S \rightarrow S^{(1)} + S^{(2)}$ $\{S.\ VAL = S^{(1)}.\ VAL + S^{(2)}.\ VAL\}$ Here S. VAL will compute the sum of $S^{(1)}$ and $S^{(2)}$values. • **Printing of Error Messages** **Example** – $A \rightarrow BC$ {error ( ); } Whenever A will be expanded to BC, an error function will be called to print an error message. The syntax-directed translation scheme is beneficial because it allows the compiler designer to define the generation of intermediate code directly in terms of the syntactic structure of the source language. It is division into two subsets known as synthesized and inherited attributes of grammar. Attributes are related to the grammar symbol that are the labels of the parse tree node. In other terms, attributes are associated information with language construct by attaching them to grammar symbols representing that construct. An attribute can describe anything (reasonable) that it can select a string, a number, a type, a memory location, a code fragment, etc. For example, an attribute for an identifier can include name, scope, type, actual arguments (number of parameters), and type of parameters, return type, etc. The value of an attribute at the parse tree node is represented by a semantic rule related with the production applied at that node. A TRANSLATION SCHEME is a context-free grammar in which semantic rules are embedded within the right sides of the productions. So a translation scheme is like a syntax-directed definition, except that the order of evaluation of the semantic rules is explicitly shown.	6	1	4	2	**1.6.1**
(b)	Express the semantic rule for productions of Boolean expression. Write three-address code for	6	3	4	2	**1.6.1**

<table>
<tr>
<td></td>
<td>
if (x < 100 || x > 200 && x != y)
x=0;

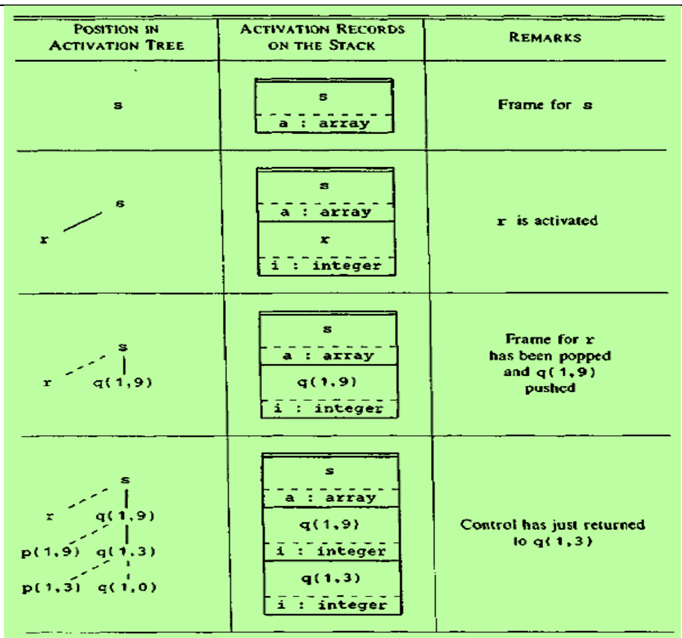
Ans:
<table>
<tr><th>PRODUCTION</th><th>SEMANTIC R RULES</th></tr>
<tr><td>B => B1 | | B2</td><td>B1.true = B.true B1.false = newlabel () B2.true = B.true B2.false = B.false B.code = B1.code || label(B1.false) || B2.code</td></tr>
<tr><td>B => B1 && B2</td><td>B1.true = newlabel () B1.false = B.false B2.true = B.true B2.false = B.false B.code = B1.code | | label(B1.true) | | B2.code</td></tr>
<tr><td>B => !B1</td><td>B1.true = B.false B1.false = B.true B.code = B1.code</td></tr>
<tr><td>B => E1 rel E2</td><td>B.code = E1.code | | E2.code | | gen('if' E1.addr rel.op E2.addr 'goto' B.true) | | gen('goto' B.false)</td></tr>
<tr><td>B => true</td><td>B.code = gen('goto' B.true)</td></tr>
<tr><td>B => false</td><td>B.code = gen('goto' B.false)</td></tr>
</table>
(3)

if (x < 100 || x > 200 && x ! = y) x = 0;
if x < 100 goto L2
goto L3
L3: if x > 200 goto L4
goto L1
L4: if x != y goto L 2
goto L1
L2: x = 0
L1: (3)
</td>
<td></td><td></td><td></td><td></td><td></td>
</tr>
<tr><td colspan="7" align="center">OR</td></tr>
<tr>
<td>18
(a)</td>
<td>
Explain the sequence of stack allocation process for a function call.

Stack Allocation

Stack allocation is based on the idea of a control stack
Storage is organized as a stack, and activation records are pushed and popped as activations begin and end respectively
Storage for the locals in each call of a procedure is contained in the activation record for that call
Thus locals are bound to fresh storage in each activation, because a new activation record is pushed onto the stack when a call is made
The values of locals are deleted when the activation ends, because the storage for locals disappears when the activation is popped
Suppose that register top marks the top of the stack
At runtime an activation record can be pushed and popped by incrementing and decrementing top by the size of the record

</td>
<td>6</td><td>2</td><td>5</td><td>1</td><td>1.6.1</td>
</tr>
</table>

<table>
<tr>
<td></td>
<td>

Calling Sequences

Procedure calls are implemented by generating calling sequences in the target code
A call sequence allocates an activation record and enters information into its fields
A return sequence restores the state of the machine so that the calling procedure can continue exec
</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(b)</td>
<td>What is an Activation Record? Explain how it is relevant to the intermediate code generation phase with respect to procedure declarations

Activation records: (3)

• Procedure calls and returns are usually managed by a run time stack called the control stack.

• Each live activation has an activation record on the control stack, with the root of the activation tree at the bottom, the latter activation has its record at the top of the stack.

• The contents of the activation record vary with the language being implemented. The diagram below shows the contents of activation record. Temporaries Local Data Machine Status Control Link Access Link Actual Parameters Return Value

• Temporary values such as those arising from the evaluation of expressions.

• Local data belonging to the procedure whose activation record this is.

• A saved machine status, with information about the state of the machine just before the call to procedures.

• An access link may be needed to locate data needed by the called procedure but found elsewhere.

• A control link pointing to the activation record of the caller.

• Space for the return value of the called functions, if any. Again, not all</td>
<td>6</td>
<td>2</td>
<td>5</td>
<td>1</td>
<td>1.6.1</td>
</tr>
</table>

<table>
<tr><td></td><td>called procedures return a value, and if one does, we may prefer to place that value in a register for efficiency.

• The actual parameters used by the calling procedure. These are not placed in activation record but rather in registers, when possible, for greater efficiency.

Intermediate Code for Procedures (3)
Let there be a function f(a1, a2, a3, a4), a function f with four parameters a1,a2,a3,a4.
Three address code for the above procedure call(f(a1, a2, a3, a4)).
param a1
param a2
param a3
param a4
call f, n

‘call’ is a calling function with f and n, here f represents name of the procedure and n represents number of parameters
example program to understand function definition and a function call.
main()
{
swap(x,y); //calling function
}

void swap(int a, int b) // called function
{
// set of statements
}</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td colspan="7">OR</td></tr>
<tr><td>19</td><td>Perform all possible optimization on the given code and explain the same.
t0=2
t1=a
t2=12
t3=t1+t2
t4=m[t3]
t5=t0*t4
t6=-16
t7=r+t6
t8=m[t7]
t9=m[t8]
t10=t9-t5
t11=4
t12=t10+t11
m[t12]=t10
Ans: 3 marks for each.

Copy propagation
t3=2+12
t4=m[t3]
t5=2*t4
t7=r+(-16)
t8=m[t7]
t9=m[t8]
t10=t9-t5</td><td>14</td><td>3</td><td>5</td><td>5</td><td>1.6.1</td></tr>
</table>

	t12=t10+4 m[t12]=t10 **Constant folding** t3=14 t4=m[t3] t5=2*t4 t7=r+(-16) t8=m[t7] t9=m[t8] t10=t9-t5 t12=t10+4 m[t12]=t10 **Copy propagation** t4=m[14] t5=2*t4 t7=r+(-16) t8=m[t7] t9=m[t8] t10=t9-t5 t12=t10+4 m[t12]=t10 **Reduction in strength** t4=m[14] t5=t4+t4 t7=r+(-16) t8=m[t7] t9=m[t8] t10=t9-t5 t12=t10+4 m[t12]=t10					