SRM Institute of Science and Technology
College of Engineering and Technology
SCHOOL OF COMPUTING

SRM Nagar, Kattankulathur – 603203, Chengalpattu District, Tamilnadu

Academic Year: 2023-24 (EVEN)

SET- A

Test: CLAT-3 **Date: 02.05.2024**
Course Code & Title: 18CSC304J -COMPILER DESIGN **Duration: 2 Periods**
Year & Sem: III & VI **Max. Marks: 50**

Part – A (10 x 1 = 10 Marks) Instructions: Answer all						
Q. No	**Question**	**Marks**	**BL**	**CO**	**PO**	**PI Code**
1	The sequence of procedure calls of a program corresponds to which traversal of the activation tree? A. In order traversal B. Pre order traversal C. Post order traversal D. Level order traversal Ans B	**1**	**1**	**4**	**1**	**1.6.1**
2	Consider the code following to apply the dead code elimination, If (condition) { a = y OP z; } else { ... } c = y OP z; y OP z should be computed as how many times in optimized code? A. one B. Two C. Three D. Four Ans A	**1**	**2**	**4**	**2**	**2.6.2**
3	Which of the following is not a three-address code? a). a = 5 b) b=a c) c=a+b d) d=a+b-c Ans D	**1**	**2**	**4**	**2**	**1.6.1**
4	On translating the expression given below into quadruple representation, how many operations are required? (i*j)+(e+f)*(a*b+c) a) 5 b)6 c)7 d)3 Ans: b	**1**	**3**	**4**	**3**	**3.6.1**
5	Which is not a permissible operation in static memory allocation? a) Call b) return c) error d)action Ans : c	**1**	**1**	**4**	**1**	**1.6.1**
6	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**	**1**	**1.6.1**
7	Ambiguous definitions for variables are more common during ______. a) Function calls b) Array declarations c) Register allocation d) Stack initialization	**1**	**1**	**5**	**1**	**1.6.1**

<table>
<tr><td></td><td>Ans: a</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>8</td><td>In algebraic expression simplification, a=a+1 can simply be replaced by?
A. a
B. INC a
C. DEC a
D. MUL a
Ans B</td><td>1</td><td>2</td><td>5</td><td>2</td><td>2.6.2</td></tr>
<tr><td>9</td><td>Which graph describes the basic block and successor relationship?
a) Control Graph
b) DAG
c) Flow graph
d) Hamiltonian graph
Ans C</td><td>1</td><td>1</td><td>5</td><td>1</td><td>1.6.1</td></tr>
<tr><td>10</td><td>The following code is an example of?
Void add_ten(int x)
{ return x + 10; printf(""value of x is %d"", x); }
(A) Redundant instruction elimination
B. Unreachable code
(C) Flow of control optimization
D. Reachable code
Ans B</td><td>1</td><td>2</td><td>5</td><td>2</td><td>2.6.2</td></tr>
<tr><td colspan="7">Part – B (4 x 4 = 16 Marks) Instructions: Answer Four</td></tr>
<tr><td>11</td><td>Explain the translation scheme to produce three address code for assignment statements.
<table>
<tr><th>Form</th><th>Intermediate Code</th></tr>
<tr><td>S→id :=E</td><td>{ p := lookup (id.name);
if p != nil then emit (p ': =' E.place)
else error }</td></tr>
<tr><td>E→E1+E2</td><td>{ E.place := newtmp;
emit(E.place ': = ' E 1.place '+'
E2.place) }</td></tr>
<tr><td>E→-E1</td><td>{ E.place := newtmp;
emit(E.place ': = 'uminus' E1.place) }</td></tr>
<tr><td>E → (E1)</td><td>{ E.place := E1.place; }</td></tr>
<tr><td>E → id</td><td>{ p := lookup (id.name);
if p!= nil then E.place := p
else error }</td></tr>
</table></td><td>4</td><td>2</td><td>4</td><td>2</td><td>2.6.2</td></tr>
<tr><td>12</td><td>Generate an intermediate code for the following code segment with the required syntax-directed translation scheme.
if (a <b)
y = a + b
else
y = a – b</td><td>4</td><td>3</td><td>4</td><td>3</td><td>3.6.1</td></tr>
</table>

	Syntax directed translation scheme for if E then S1 else S2: E.true:= newlabel; E.false:=newlabel; S1.next:=S.next; S2.next:=S.next; S.code:=E.code || gen(E.true ":") || S1.code || gen('goto' S.next) gen(E.false ":") || S2.code **Intermediate code generated:** if a>b got L1 goto L2 L1: t1:=inttoreal(b) x:=a+t1 goto L3 L2: t2:=inttoreal(b) x:=a-t2 L3:					
13	Calculate the total cost of the following instruction. MOV R0,R1 - cost 1 MOV R1,M - cost 2 SUB 5(R0),*10(R1) -cost 3 **Total cost 6**	**4**	**2**	**4**	**2**	**2.6.2**
14	Explain the basic block and flow graph **Basic Block** is a straight line code sequence that has no branches in and out branches except to the entry and at the end respectively. Basic Block is a set of statements that always executes one after other, in a sequence. A flow graph is simply a directed graph. For the set of basic blocks, a flow graph shows the flow of control information. A control flow graph is used to depict how the program control is being parsed among the blocks. A flow graph is used to illustrate the flow of control between basic blocks once an intermediate code has been partitioned into basic blocks.	**4**	**2**	**5**	**2**	**2.6.2**
15	Explain the Copy propagation and Constant folding Copy propagation is used to replace the occurrence of target variables that are the direct assignments with their values. Copy propagation is related to the approach of a common subexpression. Constant folding is an optimization technique in which the expressions are calculated beforehand to save execution time. The expressions which generate a constant value are evaluated and during the compilation time, the expressions are calculated and stored in the designated variables. This method also reduces the code sizes as well.	**4**	**2**	**5**	**2**	**2.6.2**
	Part – C (2 x12 = 24 Marks)					
16	Write quadruples, triples and indirect triples for the expression: -(a*b)+(c+d)-(a+b+c+d) Corresponding three address code is :- t1 = a * b t2 = c + d t3 = a + b t4 = t3 + t2 t5 = t2 - t1 x = t5 - t4	**12**	**3**	**4**	**3**	**3.6.1**

Quadruples representation :

Operator	Operand1	Operand2	Result
*	a	b	t1
+	c	d	t2
+	a	b	t3
+	t3	t2	t4
-	t2	t1	t5
-	t5	t4	x

Triples representation :

	Operator	Operand1	Operand2
(0)	*	a	b
(1)	+	c	d
(2)	+	a	b
(3)	+	(2)	(1)
(4)	-	(1)	(0)
(5)	-	(4)	(3)

Indirect Triples :

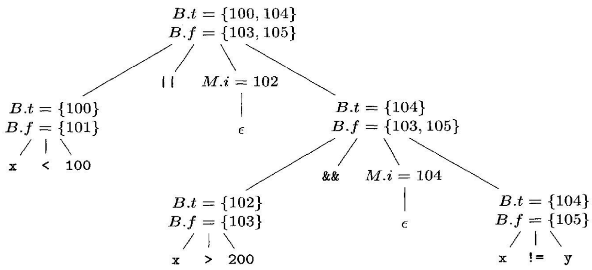
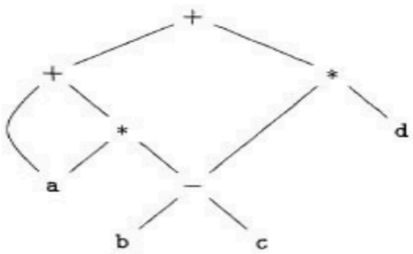
#	Operator	Operand1	Operand2
100	*	a	b
101	+	c	d
102	+	a	b
103	+	102	101
104	-	101	100
105	-	104	103

OR

17 Explain the Back patching for the expression
x < 100 || y > 200 && x != y

B → B1 || MB2	{ Backpatch(B1.fl,M.instr); B.tl = merge(B1.tl,B2.tl); B.fl = B2.fl }
B → B1 && MB2	{Backpatch(B1.tl,M.instr); B.tl = B2.tl; B.fl = merge(B1.fl,B2.fl);}
B → !B1	{B.tl = B1.fl ; B.fl = B1.tl;}
B → B1	{B.tl = B1.tl; B.fl = B1.fl;}
M→ ε	{m.instr = nextinstr; }

12 | 3 | **4** | **3** | **3.6.1**

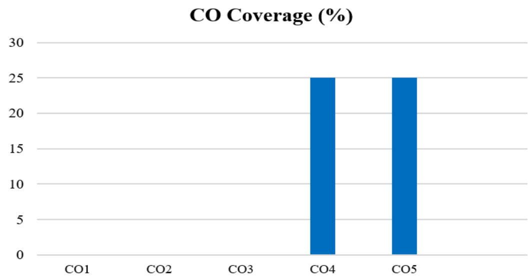
	 100: if x < 100 goto _ 101: goto 102 102: if y > 200 goto 104 103: goto _ 104: if x != y goto _ 105: goto _					
18	Generate the three-address code statement and construct the DAG representation for the expression I= a+ a*(b-c) + (b-c) *d. t1=b-c t2 =b-c t3=a*t1 t4=t2*d t5=a+t3 t6=t5+t4 	**12**	**3**	**5**	**3**	**3.6.1**
	OR					
19	Discuss in detail about storage allocation strategies. ● There are three different storage-allocation strategies: 1. Static Allocation 2. Stack Allocation 3. Heap Allocation ● Static allocation lays out storage for all data objects at compile time ● Stack allocation manages the run-time storage as a stack ● Heap allocation allocates and deallocates storage as needed at run time from a data area known as a heap Static Allocation	**12**	**2**	**5**	**2**	**2.6.2**

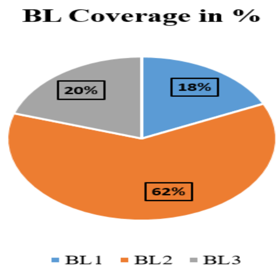
	• In static allocation, names are bound to storage as the program is **compiled** • So, there is no need for run-time support package • Every time a procedure is activated, its names are bound to the same storage locations • This property allows the values of local names to be retained across activations of a procedure • From the type of a name, the compiler determines the amount of storage to set aside for that name • The address of this storage consists of an offset from an end of the activation record for the procedure • The compiler must decide where the activation records go, relative to the target code and to one another • Once this decision is made the position of each activation record and hence the storage for each name in the record is fixed • At compile time we can fill in the addresses at which the target code can find the data it operates on • Similarly, the addresses at which information is to be saved when a procedure call occurs are also known at compile time **Limitations of Static Allocation** 1. The size of a data object and constraints on its position in memory must be known at compile time 2. Recursive procedures are restricted, because all activations of a procedure use the same bindings for local names 3. Data structures cannot be created dynamically, since there is no mechanism for storage allocation at run time 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 **Heap Allocation** • Heap allocation parcels out pieces of contiguous storage, as needed for activation records or other objects					

	● Pieces may be allocated in any order, so over time the heap will consist of alternate areas that are free and in use **Handling Activation records using Heap** ● There is generally some time and space overhead associated with using a heap manager					

*Performance Indicators are available separately for Computer Science and Engineering in AICTE examination reforms policy.

Course Outcome (CO) and Bloom's level (BL) Coverage in Questions





Approved by the Audit Professor/Course Coordinator