Register No.															

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

SRM Nagar, Kattankulathur – 603203, Chengalpattu District, Tamilnadu

SET- B

Academic Year: 2023-24 (EVEN)

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

Course Articulation Matrix: ***(to be placed)***

S.No.	Course Outcome	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2	PSO3
1	**CO1**	3	2	2	-	-	-	-	-	-	-	-	-	-	-	3
2	**CO2**	-	3	3	-	-	-	-	-	-	-	-	-	-	-	1
3	**CO3**	-	3	3	-	-	-	-	-	-	-	-	-	-	-	1
4	**CO4**	-	3	3	-	-	-	-	-	-	-	-	-	-	-	2
5	**CO5**	-	3	3	-	-	-	-	-	-	-	-	-	-	-	3

Part – A (10 x 1 = 10 Marks) Instructions: Answer all Questions						
Q. No	**Question**	**Marks**	**BL**	**CO**	**PO**	**PI Code**
1	A synthesized attribute is an attribute whose value at a parse tree node depends on _____ (A) Attributes at the siblings only (B) Attributes at parent node only **(C) Attributes at children nodes only** (D) Attributes at root nodes only **Ans: C**	**1**	**1**	**2**	**2**	**2.1.1**
2	The postfix representation of the following expression: (m + n) * (x – y) (A) + m n * - x y (B) m n + * x y - **(C) m n + x y - *** (D) m n + x y * - **Ans: C**	**1**	**1**	**2**	**2**	**2.1.1**

3	Which of the following is not a three-address code? (A) x = 50 (B) x = y (C) x = y + z **(D) x = y + z * n** **Ans: D**	**1**	**1**	**2**	**2**	**2.1.1**
4	The fields in quadruple presentation are _____ (A) operator, arg1, arg2 (B) pointer, arg1, arg2, result (C) position, arg1, arg2, result **(D) operator, arg1, arg2, result** **Ans: D**	**1**	**1**	**2**	**2**	**2.1.1**
5	Back-patching is useful for handling (A) conditional jumps (B) unconditional jumps (C) backward reference **(D) forward references** **Ans: D**	**1**	**1**	**2**	**2**	**2.1.1**
6	m * 2 can be replaced by m << 1 is an example of? (A) Algebraic expression simplification (B) Accessing machine instructions **(C) Strength reduction** (D) Code Generator **Ans: C**	**1**	**2**	**3**	**2**	**2.1.3**

7	The following code is an example of?
	`void power2(int n)`
	`{`
	`  return n * n;`
	`  printf("Power 2 of n is %d", (n*n));`
	`}`
	(A) Redundant instruction elimination
	(B) Unreachable code
	(C) Flow of control optimization
	(D) Reachable code
	
	Ans: B
		1	**1**	**3**	**2**	**2.1.1**

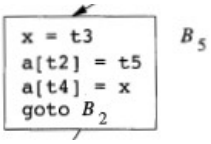
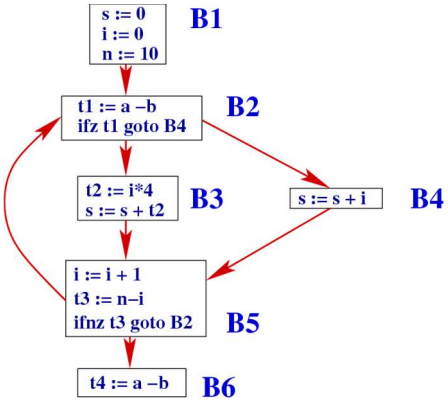
8	DAG stands for ____ (A) Data Acyclic Graph (B) Dynamic Acyclic Graph (C) Data Asynchronized Graph **(D) Directed Acyclic Graph** **Ans: D**	**1**	**1**	**3**	**2**	**2.1.1**

9	In loop optimization, ____ moves code outside a loop (A) Code run **(B) Code motion** (C) Code shift (D) Code push **Ans: B**	**1**	**1**	**3**	**2**	**2.1.1**
10	Peephole optimization is a _____ **(A) Local optimization** (B) Loop optimization (C) Global optimization (D) S-Attribute optimization **Ans: A**	**1**	**2**	**3**	**2**	**2.1.3**

Part – B
Answer Any four Questions
(4 x 4 = 16 Marks)

11	Write three-address code for the following code segment:
	if (a > b) then
	c = a
	else
	c = b
	
	Answer:
	if a > b goto L2
	c = b
	goto L1
	L2: c = a
	L1:
		4	**3**	**2**	**2**	**2.1.3**

12	Write a note on register and address descriptors. **Register Descriptor: - 2 marks** Register descriptor is used to inform the code generator about the availability of registers. Register descriptor keeps track of values stored in each register. Whenever a new register is required during code generation, this descriptor is consulted for register availability. **Address Descriptor: - 2 marks** Values of the names (identifiers) used in the program might be stored at different locations while in execution. Address descriptors are used to keep track of memory locations where the values of identifiers are stored. These locations may include CPU registers, heaps, stacks, memory or a combination of the mentioned locations.	**4**	**2**	**2**	**2**	**2.1.1**

13	Brief about cross compiler A cross compiler is a compiler capable of creating executable code for a platform other than the one on which the compiler is run. Cross compiler tools are used to generate executables for embedded system or multiple platforms. It is used to compile for a platform upon which it is not feasible to do the compiling, like microcontrollers that don't support an operating system. **Uses of cross compilers:** The fundamental use of a cross compiler is to separate the build environment from target environment. This is useful in a number of situations: Embedded computers where a device has extremely limited resources. Since debugging and testing may also require more resources than are available on an embedded system, cross-compilation can be less involved and less prone to errors than native compilation.	**4**	**3**	**2**	**2**	**2.1.3**
14	Write about basic blocks and flow graphs in compiler design **Basic Blocks: - 2 marks** A basic block refers to a sequence of consecutive statements in a program's control flow graph that has a single entry point at the beginning and a single exit point at the end. These blocks are fundamental units for analysis and optimization during compilation. Example:  **Flow Graph: - 2 marks** Flow Graph is a graphical representation of the control flow within a program. It visually depicts how control flows from one basic block to another, including the possible conditional and unconditional transfers of control. Example:  	**4**	**3**	**3**	**2**	**2.1.3**

15	Write a note on common sub-expression elimination Common Subexpression Elimination is a compiler optimization technique used to eliminate redundant computations by identifying and removing common subexpressions within a program. B_5 t6 = 4*i x = a[t6] t7 = 4*i t8 = 4*j t9 = a[t8] a[t7] = t9 t10 = 4*j a[t10] = x goto B_2 (a) Before. B_5 t6 = 4*i x = a[t6] t8 = 4*j t9 = a[t8] a[t6] = t9 a[t8] = x goto B_2 (b) After.	**4**	**2**	**3**	**2**	**2.1.1**
	Part – C **Answer All Question** **(2 X 12 = 24 Marks)**					
16	What is three-address code? Write three-address code for the following expression and represent it in quadruple, triple and indirect triple. a * b + c / e ^ f - b * c **Three-address code: - 3 marks** t1 = a * b t2 = e ^ f t3 = c / t2 t4 = b * c t5 = t1 + t3 t6 = t5 – t4 **Quadruple representation: - 3 marks** **Operator | Argument 1 | Argument 2 | Result** * | a | b | t1 ^ | e | f | t2 / | c | t2 | t3 * | b | c | t4 + | t1 | t3 | t5 - | t5 | t4 | t6 **Triple Representation: - 3 marks** **# | Operator | Argument 1 | Argument 2** (0) | * | a | b (1) | ^ | e | f (2) | / | c | (1) (3) | * | b | c (4) | + | (0) | (2) (5) | - | (4) | (3)	**12**	**4**	**2**	**2**	**2.1.3**

Indirect Triple Representation: - 3 marks

#	Operator	Argument 1	Argument 2
(20)	*	a	b
(21)	^	e	f
(22)	/	c	(21)
(23)	*	b	c
(24)	+	(20)	(22)
(25)	-	(24)	(23)

#	statement
(0)	(20)
(1)	(21)
(2)	(22)
(3)	(23)
(4)	(24)
(5)	(25)

OR

17	How Boolean expressions are translated using backpatching technique? Explain with an example.	**12**	**4**	**2**	**2**	**2.1.3**

Backpatching is a technique used in compiler design and code generation to efficiently handle addresses or labels in intermediate code, particularly in situations where the target addresses are not known during the initial generation of the code. It is commonly used in contexts such as code generation for control flow constructs like if-else statements, loops, and switch statements.

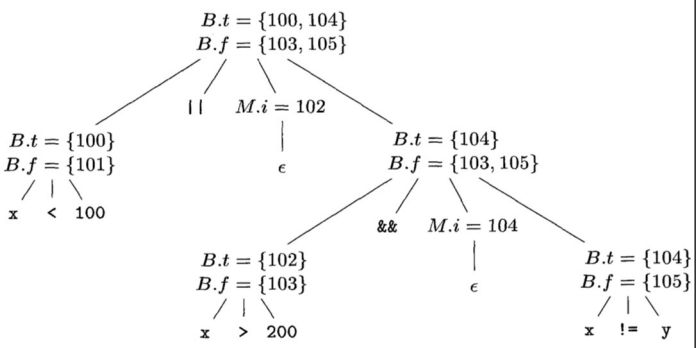
To manipulate list of labels, three functions are used:

- makelist(i)
- merge(p1,p2) and
- backpatch(p,i)

Let the grammar be:

B → B1 or MB2 | B1 and MB2 | not B1 | (B1) | id1 relop id2 | false | true

M → ε

	Translation scheme for Boolean expression: 1) $B \rightarrow B_1$ `||` M B_2 { $backpatch(B_1.falselist, M.instr)$; $B.truelist = merge(B_1.truelist, B_2.truelist)$; $B.falselist = B_2.falselist$; } 2) $B \rightarrow B_1$ `&&` M B_2 { $backpatch(B_1.truelist, M.instr)$; $B.truelist = B_2.truelist$; $B.falselist = merge(B_1.falselist, B_2.falselist)$; } 3) $B \rightarrow$ `!` B_1 { $B.truelist = B_1.falselist$; $B.falselist = B_1.truelist$; } 4) $B \rightarrow$ `(` B_1 `)` { $B.truelist = B_1.truelist$; $B.falselist = B_1.falselist$; } 5) $B \rightarrow E_1$ **rel** E_2 { $B.truelist = makelist(nextinstr)$; $B.falselist = makelist(nextinstr + 1)$; $gen('\texttt{if}'\ E_1.addr$ **rel**$.op\ E_2.addr\ '\texttt{goto \_}')$; $gen('\texttt{goto \_}')$; } 6) $B \rightarrow$ **true** { $B.truelist = makelist(nextinstr)$; $gen('\texttt{goto \_}')$; } 7) $B \rightarrow$ **false** { $B.falselist = makelist(nextinstr)$; $gen('\texttt{goto \_}')$; } 8) $M \rightarrow \epsilon$ { $M.instr = nextinstr$; } Example: Annotated parse tree for `if( x < 100 || x > 200 && x != y ) x = 0;` 					
18	Consider the following source code for dot product of two vectors a and b of length 10: **1. begin** **2. prod :=0;** **3. i:=1;** **4. do begin** **5. prod :=prod+ a[i] * b[i];** **6. i :=i+1;** **7. end** **8. while i <= 10** **9. end** (a) Construct the basic blocks. (6 Marks) (b) Draw the flow graphs (6 Marks)	**12**	**4**	**3**	**2**	**2.1.3**

Answer:

a) Basic Blocks: **- 6 Marks**

B1:

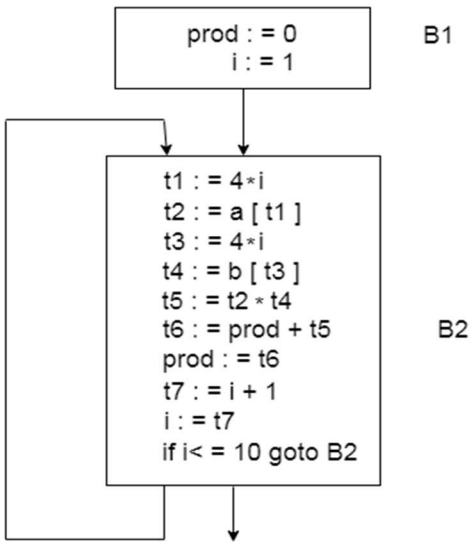
(1) prod := 0
(2) i := 1

B2

(3) t1 := 4* i
(4) t2 := a[t1]
(5) t3 := 4* i
(6) t4 := b[t3]
(7) t5 := t2*t4
(8) t6 := prod+t5
(9) prod := t6
(10) t7 := i+1
(11) i := t7
(12) if i<=10 goto (3)

b) Flow Graph: **- 6 Marks**

Flow graph for the vector dot product is given as follows:



OR

19	i) Discuss in detail about peephole optimization with suitable example. ii) Write about parameter passing	**12**	**4**	**3**	**2**	**2.1.3**

Answer:

i) Peephole Optimization: - 8 marks

Peephole optimization is a local and iterative optimization technique used in compiler design to improve the efficiency of generated code by analyzing and transforming small, contiguous sections of assembly or machine code, known as "peepholes." These peepholes typically consist of a fixed number of adjacent instructions.

Characteristic of peephole optimizations:

- Redundant-instruction elimination
- Flow-of-control optimizations
- Algebraic simplifications
- Use of machine idioms

Redundant-instruction elimination:

```
int add_ten(int x)
   {
   int y, z;
   y = 10;
   z = x + y;
   return z;
   }
```

```
int add_ten(int x)
   {
   int y;
   y = 10;
   y = x + y;
   return y;
   }
```

```
int add_ten(int x)
   {
   int y = 10;
   return x + y;
   }
```

```
int add_ten(int x)
   {
   return x + 10;
   }
```

Flow-of-control optimizations:

```
...
MOV R1, R2
GOTO L1
...
L1 :   GOTO L2
L2 :   INC R1
```

```
...
MOV R1, R2
GOTO L2
...
L2 :   INC R1
```

Algebraic simplifications:

- There are occasions where algebraic expressions can be made simple.
- For example, the expression **a = a + 0** can be replaced by **a** itself
- The expression a = a + 1 can simply be replaced by INC a.

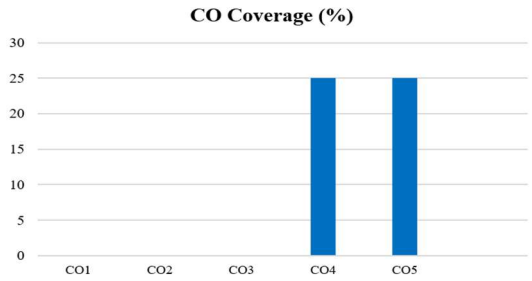
Use of machine idioms:

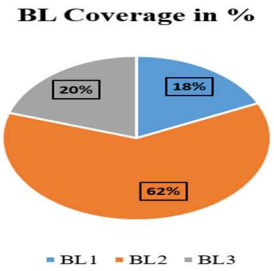
- The target machine may have hardware instructions to implement certain specific operations efficiently.
- Detecting situations that permit the use of these instructions can reduce execution time significantly.
- For example,
  - ➢ some machines have auto-increment and auto-decrement addressing modes.
  - ➢ These add or subtract one from an operand before or after using its value.
  - ➢ The use of the modes greatly improves the quality of code when pushing or popping a stack, as in parameter passing.

	ii) Parameter passing - 4 marks **Call by value** Actual parameters are evaluated and their r-values are passed to the called procedure caller evaluates the actual parameters and places r-value in the storage for formals call has no effect on the activation record of caller **Call by reference (call by address)** The caller passes a pointer to each location of actual parameters if actual parameter is a name then l-value is passed if actual parameter is an expression then it is evaluated in a new location and the address of that location is passed **Copy Restore** A hybrid between call by value and call by reference Also called as copy-in copy-out/ call by value result actual parameters are evaluated, rvalues are passed by call by value, lvalues are determined before the call when control returns, the current rvalues of the formals are copied into lvalues of the locals **Call by name** The procedure is treated as if it were a macro, its body is substituted for the call in the caller with the actual parameters The local names of the called procedure are kept distinct from the names of the calling procedure. The actual parametes are surrounded by parentheses if necessary to preserve their integrity					

*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