



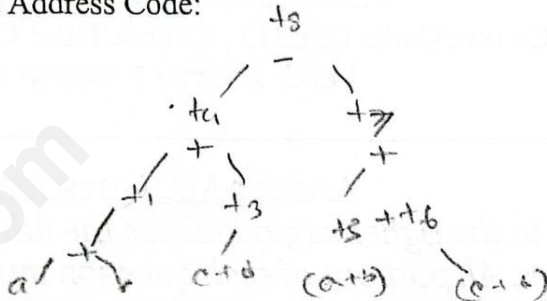
Daffodil International University
Faculty of Science & Information Technology
Department of Computer Science & Engineering
Final Examination, Spring 2026
Course Code: CSE313 , Course Title: Compiler Design
Level: 3 Term: 1 Batch: 66

Time: 02:00 Hrs
Marks: 40

Answer ALL Questions

[The figures in the right margin indicate the full marks and corresponding course outcomes. All portions of each question must be answered sequentially.]

1.	a)	Consider the following grammar to produce LR(0) parser and canonical table and identify any shift-reduce conflicts, if present. $S \rightarrow X + Y Z$ $Y \rightarrow W n \mid n \mid W$ $X \rightarrow p \mid q$ $W \rightarrow r \mid s \mid t$ $Z \rightarrow u \mid v$	[8]	CO2
	b)	Consider the following code segment: <pre>sum = 0; for (i = 0; i < n; i++) { x = a * b; y = i * 2; sum = sum + x + y; }</pre> Apply appropriate code optimization techniques to optimize the given code and rewrite the optimized code using suitable methods and justify the applied techniques.	[2]	
2.	a)	Given the grammar below, apply predictive parsing techniques to determine the FIRST and FOLLOW sets of each nonterminal. Then, use these sets to build the LL(1) parsing table. $P \rightarrow S Q$ $S \rightarrow S a \mid b T \mid \epsilon$ $T \rightarrow c T \mid d$ $Q \rightarrow f Q \mid g$	[7]	CO2
	b)	Given the grammar below, eliminate left factoring and produce an equivalent left-factored grammar: $S \rightarrow \text{stmt} \mid \text{stmt block} ; \text{stmt} \mid \text{stmt block} , \text{stmt}$ $\text{stmt} \rightarrow \text{while expr do stmt} \mid \text{while expr do stmt block}$	[3]	

3.	a)	Consider the following arithmetic expression: $(a + b) * (c - d) + (a + b) / (-e) + (f - d)$ Apply intermediate code generation techniques to evaluate the expression. - Apply your knowledge to write the Three Address Code (TAC) for the expression. - Generate the corresponding Quadruples representation. - Analyze the structure and produce the Indirect Triples representation where you have 25, 27, 29, 31, 33, 35 and 41 available memory in STACK.	[6]	CO3																																										
	b)	Consider the following Three Address Code: $t1 = a + b$ $t3 = c + d$ $t4 = t1 + t3$ $t5 = a + b$ $t6 = c + d$ $t7 = t5 + t6$ $t8 = t4 - t7$  - Construct the Syntax Tree for the Three Address Code. - Construct a Directed Acyclic Graph (DAG) for the Three Address Code.	[4]																																											
4.	a)	Consider the following Three Address Code: <table border="1"> <tr> <td>1. $i = 0$</td> <td>15. $t6 = 4 * i$</td> <td>29. $t12 = 4 * j$</td> </tr> <tr> <td>2. $j = n$</td> <td>16. $y = a[t6]$</td> <td>30. $p = a[t12]$</td> </tr> <tr> <td>3. $k = m$</td> <td>17. $t7 = 4 * j$</td> <td>31. If $p == x$ goto (34)</td> </tr> <tr> <td>4. $t1 = 4 * i$</td> <td>18. $t8 = a[t7]$</td> <td>32. $q = p + x$</td> </tr> <tr> <td>5. $x = a[t1]$</td> <td>19. $a[t6] = t8$</td> <td>33. goto (36)</td> </tr> <tr> <td>6. $i = i + 1$</td> <td>20. $a[t7] = y$</td> <td>34. $q = p - x$</td> </tr> <tr> <td>7. $t2 = 4 * i$</td> <td>21. $k = k - 1$</td> <td>35. $r = q * 2$</td> </tr> <tr> <td>8. $t3 = a[t2]$</td> <td>22. If $k > 0$ goto (6)</td> <td>36. $s = r + 1$</td> </tr> <tr> <td>9. If $t3 < x$ goto (6)</td> <td>23. $t9 = 4 * i$</td> <td>37. If $s < 100$ goto (21)</td> </tr> <tr> <td>10. $j = j - 1$</td> <td>24. $z = a[t9]$</td> <td>38. $t13 = 4 * k$</td> </tr> <tr> <td>11. $t4 = 4 * j$</td> <td>25. $t10 = 4 * n$</td> <td>39. $a[t13] = s$</td> </tr> <tr> <td>12. $t5 = a[t4]$</td> <td>26. $t11 = a[t10]$</td> <td>40. end</td> </tr> <tr> <td>13. If $t5 > x$ goto (10)</td> <td>27. $a[t9] = t11$</td> <td></td> </tr> <tr> <td>14. If $i >= j$ goto (25)</td> <td>28. $a[t10] = z$</td> <td></td> </tr> </table> Answer the following question: - Which lines in the given three-address code are identified as leaders? - Is / Are there any instruction(s) designated as leaders using both Rule 2 and Rule 3? - What is the total number of basic blocks identified in the code? - Draw the flow graph for the above-mentioned instructions.	1. $i = 0$	15. $t6 = 4 * i$	29. $t12 = 4 * j$	2. $j = n$	16. $y = a[t6]$	30. $p = a[t12]$	3. $k = m$	17. $t7 = 4 * j$	31. If $p == x$ goto (34)	4. $t1 = 4 * i$	18. $t8 = a[t7]$	32. $q = p + x$	5. $x = a[t1]$	19. $a[t6] = t8$	33. goto (36)	6. $i = i + 1$	20. $a[t7] = y$	34. $q = p - x$	7. $t2 = 4 * i$	21. $k = k - 1$	35. $r = q * 2$	8. $t3 = a[t2]$	22. If $k > 0$ goto (6)	36. $s = r + 1$	9. If $t3 < x$ goto (6)	23. $t9 = 4 * i$	37. If $s < 100$ goto (21)	10. $j = j - 1$	24. $z = a[t9]$	38. $t13 = 4 * k$	11. $t4 = 4 * j$	25. $t10 = 4 * n$	39. $a[t13] = s$	12. $t5 = a[t4]$	26. $t11 = a[t10]$	40. end	13. If $t5 > x$ goto (10)	27. $a[t9] = t11$		14. If $i >= j$ goto (25)	28. $a[t10] = z$		[7]	CO3
1. $i = 0$	15. $t6 = 4 * i$	29. $t12 = 4 * j$																																												
2. $j = n$	16. $y = a[t6]$	30. $p = a[t12]$																																												
3. $k = m$	17. $t7 = 4 * j$	31. If $p == x$ goto (34)																																												
4. $t1 = 4 * i$	18. $t8 = a[t7]$	32. $q = p + x$																																												
5. $x = a[t1]$	19. $a[t6] = t8$	33. goto (36)																																												
6. $i = i + 1$	20. $a[t7] = y$	34. $q = p - x$																																												
7. $t2 = 4 * i$	21. $k = k - 1$	35. $r = q * 2$																																												
8. $t3 = a[t2]$	22. If $k > 0$ goto (6)	36. $s = r + 1$																																												
9. If $t3 < x$ goto (6)	23. $t9 = 4 * i$	37. If $s < 100$ goto (21)																																												
10. $j = j - 1$	24. $z = a[t9]$	38. $t13 = 4 * k$																																												
11. $t4 = 4 * j$	25. $t10 = 4 * n$	39. $a[t13] = s$																																												
12. $t5 = a[t4]$	26. $t11 = a[t10]$	40. end																																												
13. If $t5 > x$ goto (10)	27. $a[t9] = t11$																																													
14. If $i >= j$ goto (25)	28. $a[t10] = z$																																													
	b)	Explain Dead Code Elimination and Loop Invariants. Provide examples to show how each optimization works.	[3]																																											