Roll no. - 19115090
Name - Sonal Dubey

Compiler Design
Assignment - 2

(1.) Generate three address code for the following :-

(i) begin

```
add = 0
i = 1
j = 1
do
    begin
        add = add + a[i,j] * b[j,i]
        i = i+1
        j = j+1
    end
while i<=20 and j<=20
end
```

Soln: 1(i) Assuming 'a' and 'b' are arrays of size [20, 20] of 4 bytes per word, and the matrices follow row major order, the three address code for the given code can be written as:

1. add = 0
2. i = 1
3. j = 1
4. t1 = i * 20
5. t1 = t1 + j
6. t1 = t1 * 4
7. t2 = addr(a) - 84
8. t3 = t2 [t1]
9. t4 = j * 20
10. t4 = t4 + i
11. t4 = t4 * 4
12. t5 = addr(b) - 84
13. t6 = t5 [t4]
14. t7 = t3 * t6
15. t7 = add + t7
16. t8 = i + 1
17. i = t8
18. t9 = j + 1
19. j = t9
20. if i <= 20 goto (22)
21. goto NEXT
22. if j <= 20 goto (4)
23. goto NEXT

(ii) while (A<C and B>D) do
if A=1 then C=C+1
else
while A<=D do
A=A+3

Soln: 1 (ii) Three address code for the given code is:

1. if (A < C) goto (3)
2. goto (15)
3. if (B > D) goto (5)
4. goto (15)
5. if (A=1) goto (7)
6. goto (10)
7. T1 = C+1
8. C = T1
9. goto (1)
10. if (A<=D) goto (12)
11. goto (1)
12. T2 = A+B
13. A = T2
14. goto (10)
15. goto NEXT

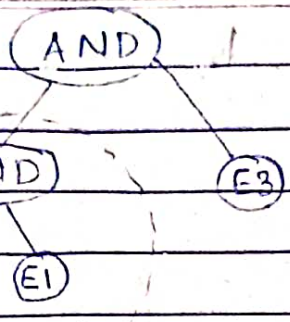
Q(2): For the code below what are the cases when we need backpatching:

(A) E1 = i >= j;
E2 = j != i;
E3 = i <= k;
if (E2 && E1 && E3)
{// True section}
else
{//False section}

Sol^n: 2(A). Three address code for the boolean expression E2 && E1 && E3, assuming initial line number to be 1001, can be written as:

101. if E2 goto ____
102 goto ____
103 if E1 goto ____
104 goto ____
105 if E3 goto ____
106 goto ____

Let (E2 && E1) be E4.
Binary tree to calculate the backpatching can be given as follows:



AAND

From address code:

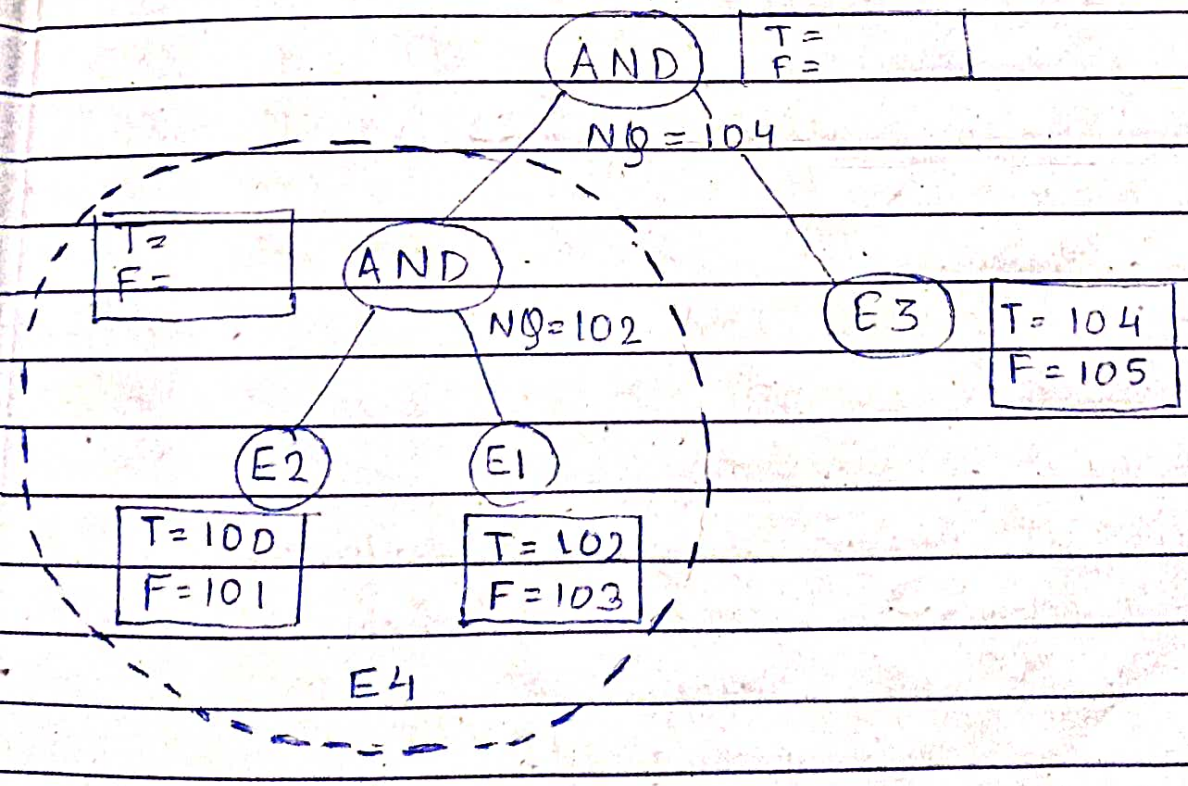
for E1 ⇒ T = 102, F = 103

for E2 ⇒ T = 100, F = 101

for E3 ⇒ T = 104, F = 105.

The next quadruple for E4 is 102.

The next quadruple for E4 && E3 is 104.



Truth table for E4 = E2 && E1 is:

E2	E1	AND
0	0	0
0	1	0
1	0	0
1	1	1

Therefore, backpatching is required when E2 = 1 (true). So, Backpatch statement for E4 will be : Backpatch(100, 102)

Truth list for E4 = 100, 102.

Since we backpatched (100, 102), truth list will be only 102.

False list for E4 = 101, 103.

Similarly, truth table for E4 && E3 is:

E4	E3	AND
0	0	0
0	1	0
1	0	0
1	1	1

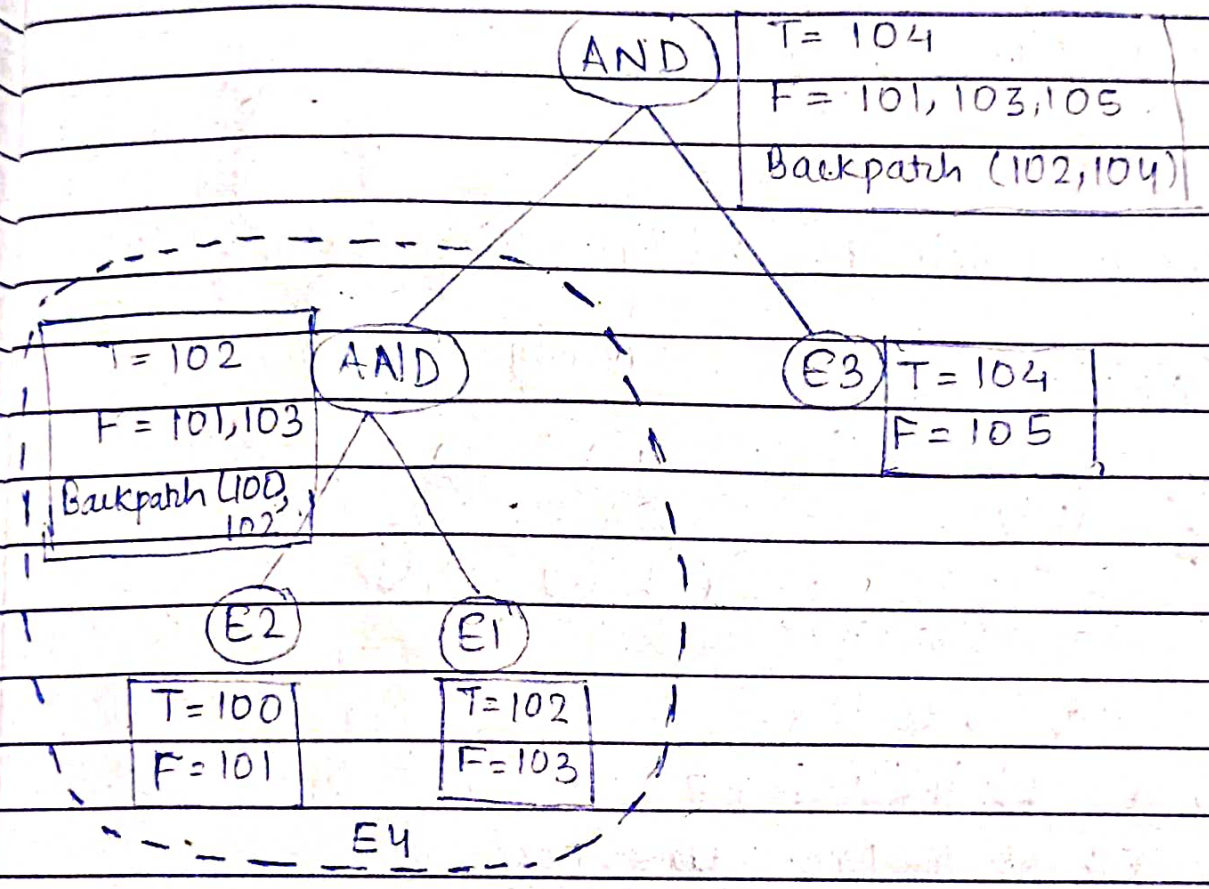
Therefore backpatching is required when E4 = 1 (true), so backpatch statement will be;

Backpatch (102, 104)

Truth list = 102, 104 = 104

False list = 101, 103, 105.

Therefore, final tree will be:

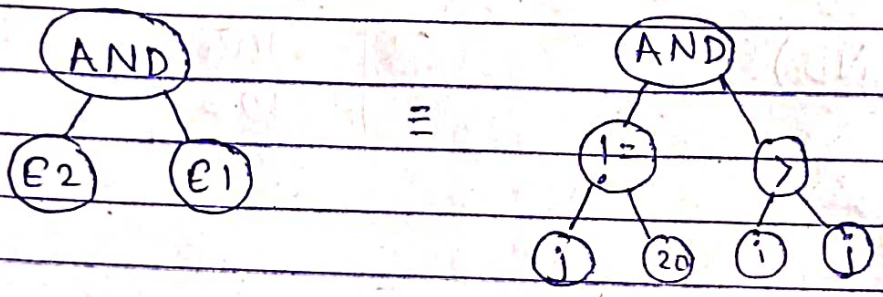


(B) E1 = i > j;
E2 = j != 20;
if (E2 && E1)
{ // True section}
else
{ // False Section}

Soln: 2(B) The three address for the boolean expression E2 && E1, assuming initial line no. to be 101, can be written as:

101. if (E2) then goto ____
102. goto ____
103. if (E1) then goto ____
104. goto ____

Binary tree to calculate backpatching can be given as:

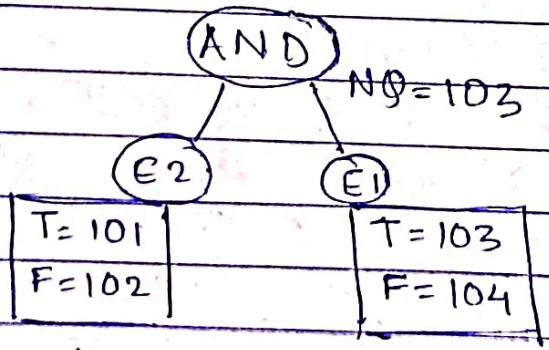


From three address code,

for E2 ⇒ T = 101 , F = 102

for E1 ⇒ T = 103 , F = 104.

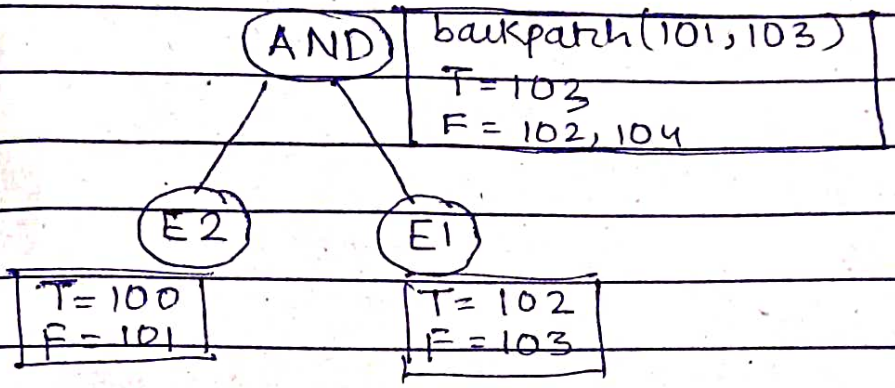
The next Quadruple is 103. Therefore



Truth table for E2 and E1 is:

E2	E1	AND
0	0	0
0	1	0
1	0	0
1	1	1

Therefore, backpatching is required when E2=1 (true)
So, backpatch statement will be: backpatch(101, 103)
Truth list = 101, 103 = 103
False list = 102, 104.
So, final tree will be:



(3.) For the given expression:
S+ R/3+q-t*5+

(3.) For the given expression: S+ R/3+q-t*5+U+V* W/U what is the minimum number of temporary variables needed to create a three-address code in static single assignment form? Show the complete solution.

Soln (3): The rules in static assignment form are:

(i) Each variable must be assigned exactly once.
(ii) Every variable must be defined before it is used.
(iii) All of the uses reached by the assignment must be renamed.

Given expression : $s + r/3 + q - t * 5 + u + v * w / u$

So,

$t1 = r/3$

$t2 = t * 5$

$t3 = v * w$

$t4 = t3 / u$

$t5 = s + t1$

$t6 = q + t5$

$t7 = t6 - t2$

$t8 = t7 + u$

$t9 = t8 + t4$

Total 9 temporary variables are required to create a three-address code in static single assignment form.

(4.) Consider a syntax directed translation

$E \rightarrow E * T$ { E.val = E.val * T.val }

$E \rightarrow T$ { E.val = T.val }

$T \rightarrow F - T$ { T.val = F.val - T.val }

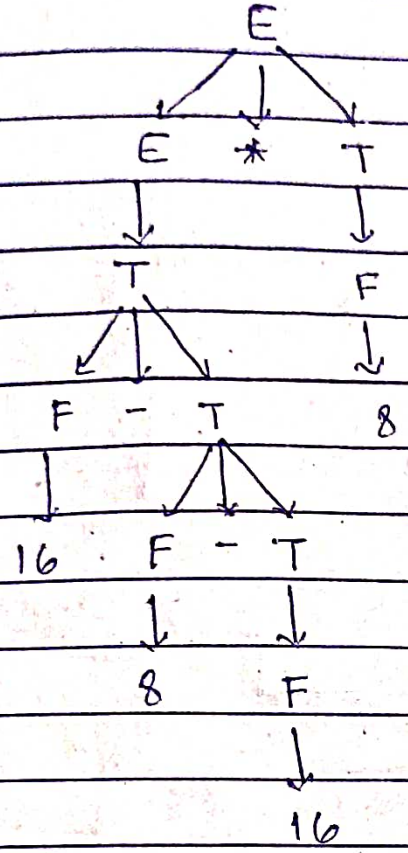
$T \rightarrow F$ { T.val = F.val }

$F \rightarrow 8$ { F.val = 8 }

$F \rightarrow 16$ { F.val = 16 }

Evaluate the input string 16-8-16*8 and find number of reduction. Draw the syntax directed translation tree.

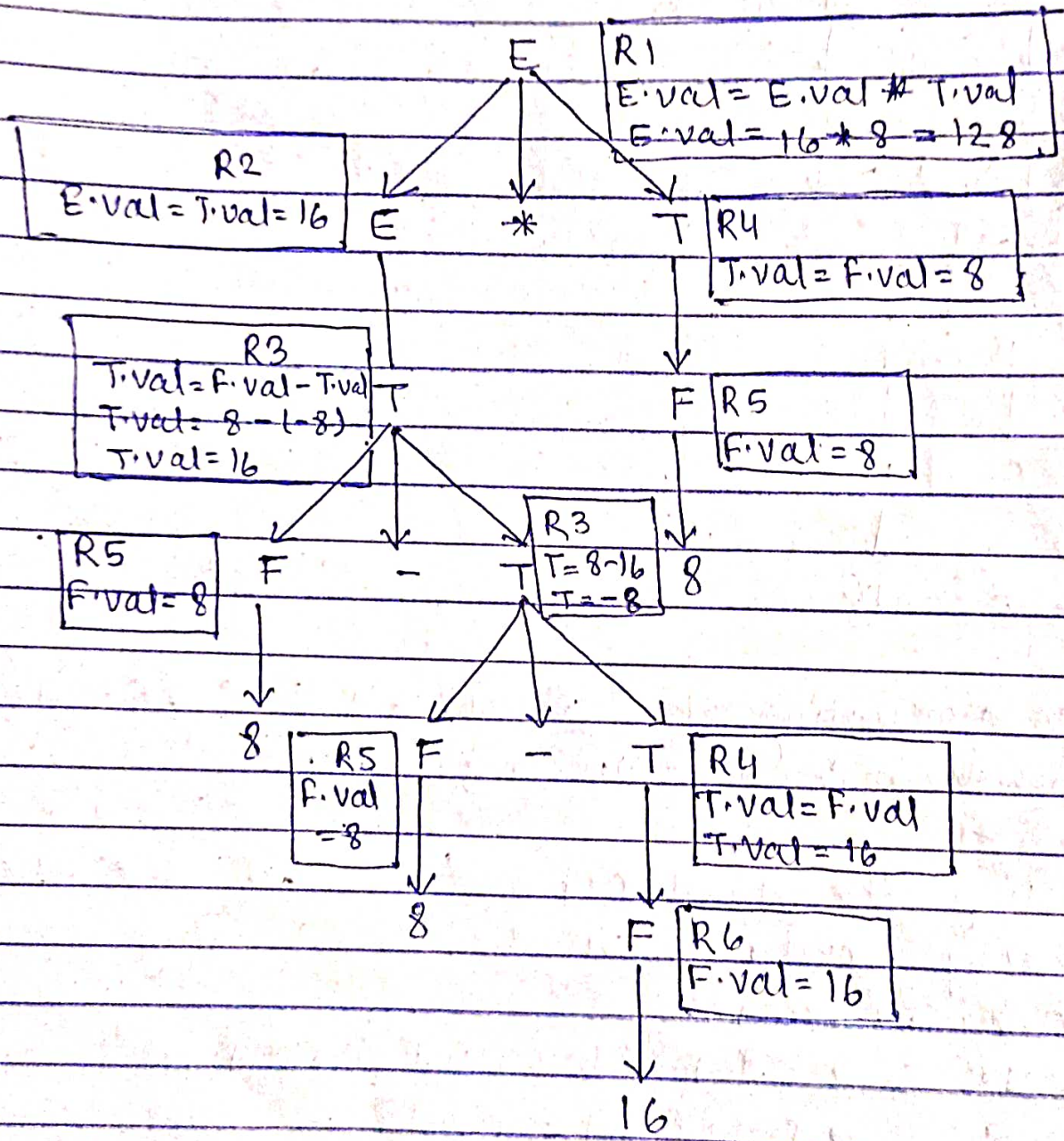
Sol"(4): The SDT tree for the given expression will be:



~~For~~ For evaluating the given expression, we traverse the tree ~~in postorder~~ inorderly.

For evaluating the given expression, we traverse the tree inorderly, and perform the corresponding reduction while moving upwards.

Given,
1. $E \rightarrow E * T$ { E.val = E.val * T.val }
2. $E \rightarrow T$ { E.val = T.val }
3. $T \rightarrow F - T$ { T.val = F.val − T.val }
4. $T \rightarrow F$ { T.val = F.val }
5. $F \rightarrow 8$ { F.val = 8 }
6. $F \rightarrow 16$ { F.val = 16 }



Thus, $16 - 8 - 16 * 8 = 128$

and total number of reductions are 10.

15.) Explain the significance of intermediate code generation in terms of compiles design.

Ans(5.): In the analysis-synthesis model of a compiler, the front end of a compiler translates a source program into an independent intermediate code, then the back end of the compiler uses this intermediate code to generate the target code (which can be understood by the machine). The benefits of using machine independent intermediate code are:

1) Because of the machine independent intermediate code, portability will be enhanced.
2) Retargeting is facilitated.
3) It is easier to apply source code modification to improve the performance of source code by optimising the intermediate code.
4) If we generate machine code directly from source code then for n target machines we will have n optimisers and n code generators but if we have a machine independent intermediate code, we will have only one optimiser.

Intermediate code can be either language specific (eg.-Bytecode) or language independent (eg.-Three address code).