ass :- name info. by using whois

The website to get detailed info about domain name info. including its owner, its regis date of registrat^n, expiry, name of server, contact info. etc.

- Finding IP addr.

CC

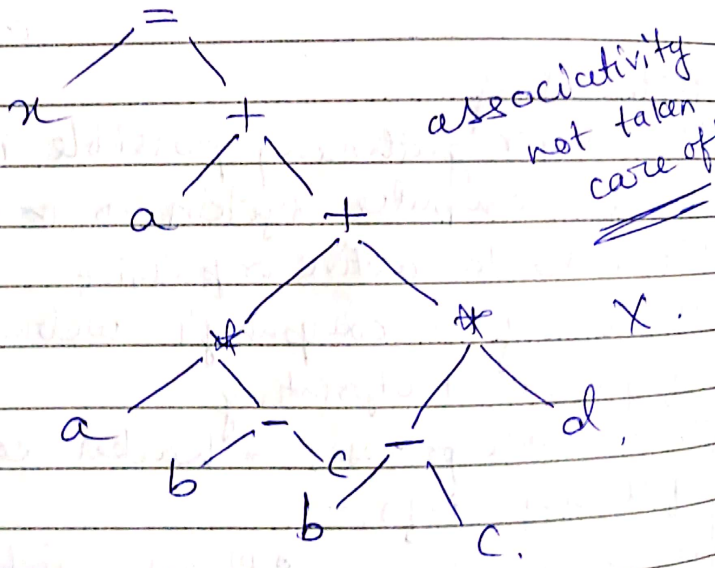
Q- tokens → [syntax] → syntax tree → [type checking / semantic analysis] → Intermediate Code Represent

- Intermediate Code Representat^n -

1) Syntax tree. 2). DAGs (Directed Acyclic Graph

ex :- $x = a + a * (b-c) + (b-c) * d$

AST

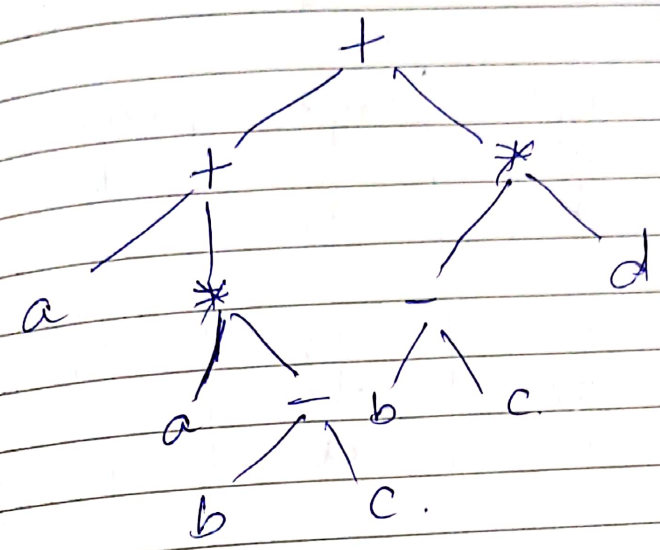


associativity not taken care off

X.

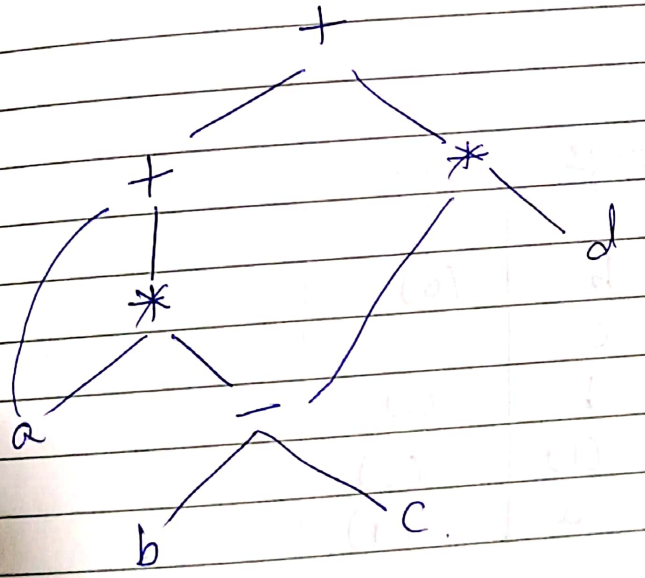
DAG

* '+' left associative.



AST →

DAG.



common subexpressions removed.

3AC (3 address code) {at most 3 addresses}
- 3 quadruples
- 3 triples.
- 3 indirect triples.

ex:- $a = y + x + z$.

$t_1 = y + x$
$t_2 = t_1 + z$.
$a = t_2$.

} 3AC for given expressions

if ($x < y$) then add1 else add2 } more than 3 addresses

C me IR nhi hota.
Java has ICG thing!!

→ quadriples – → 4 columns/4 cells.

	operator.	arg1	arg2.	result.
0	–	c		t1
1	*	b	t1	t2
2	–	c		t3
3	*	b	t3	t4
4	+	t2	t4	t5.
5	=	t5		a
6				

ex :- a = b * – c + b * – c

→ triples

	op	arg1	arg 2.
0	–	c	
1	*	b	(0)
2	–	c	
3	*	b	(2)
4	+	(1)	(3)
5	=	a	(4)

→ Indirect triples.

35	(0)
36	(1)
37	(2)
101	(3)
108	(4)
109	(5)
⋮	⋮

ass-SLR parser

Low level virtual machine generator

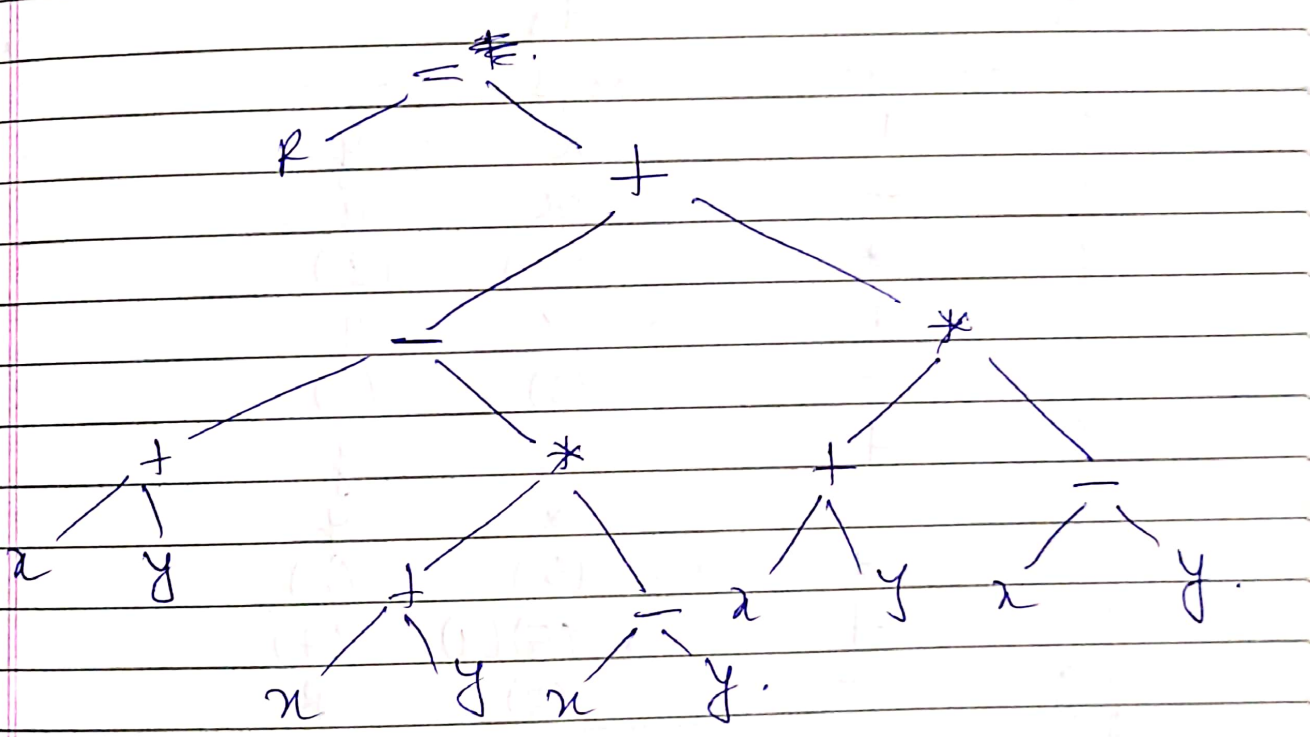
ass:- ICG # with quadruples & triples, take any expression.

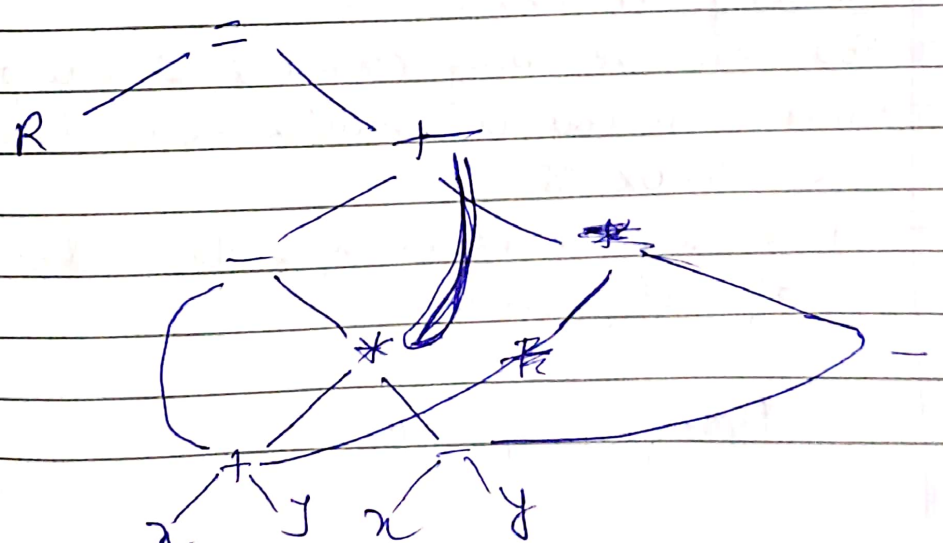
- SSA (Single Static Assignment) - [LLVM uses this]

$p_1 = a + b$
$q_1 = p_1 - c$
$p_2 = q_1 * d$
$p_3 = e - p_2$
$q_2 = p_3 + q_1$

useful for Dataflow Analysis at the time of optimization or CG.

ex:- $R = ((x+y) - ((x+y) * (x-y))) + ((x+y) * (x-y))$





$((x+y)-((x+y)*(x-y)))+((x+y)*(x-y))$

	operator	org1	org2	result
0	+	x	y	t_1
1	−	x	y	t_2
2	*	t_1	t_2	t_3
3	+	x	y	t_4
4	−	t_4	t_3	t_5
5	+	x	y	t_6
6	−	x	y	t_7
7	*	t_6	t_7	t_8
8	+	t_5	t_8	t_9
9	=	t_9		R

	operator	org1	org2
0	+	x	y
1	−	x	y
2	*	(0)	(1)
3	+	x	y
4	−	(3)	(2)
5	+	x	y
6	−	x	y
7	*	(5)	(6)
8	+	(4)	(7)
9	=	(8)	R

CS

Finding IP address

→ You can use ping command at your prompt.

→ This command is available on windows as well as on Linux OS.

→ following is the example to find out the IP add of coep.org.in

$ping coep.org.in

platforms

5) Hybrid cloud Migration -

techniques -

1) Lift & shift.
2) Cloud bursting.

Process:

Pros: cost optimization, flexibility in managing workloads

Cons: Network Latency.

CE

3AC.

Arrays 1D X = a[i]

base + i x w

t1 = i x w
t2 = a[t1]

{base+t1}

3AC.

2D Arrays

00	01	02
10	11	12
20	21	22

3×3

$x = a[i][j]$

a[row][col]

row major

00	01	02	10	11	12	20	21	22

100

example

100 + (2×3+1)×2 = 128

$L = a[i][j] = base + (i \times c + j) * w$

c → col. w → width of each element.

3AC

$t_1 = i \times c$

$t_2 = t_1 + j$

$t_3 = t_2 \times w$

$t_4 = a[t_3]$

$x = t_4$

col major

00	10	20	01	11	21	02	12	22

$base + (j \times r + i) * w$

r → row

3D Arrays

a[i][j][k] a[r][c][b]

$Loc\ a[i][j][k] = base + (i \times c \times b + j \times b + k) * w$

c → col. b → last dim. w → width

$t_1 = i \times c$

$t_2 = t_1 \times b$

$t_3 = j \times b$

$t_4 = t_2 + t_3$

$t_5 = t_4 + k$

$t_6 = t_5 * w$

$t_7 = a[t_6]$

$x = t_7$

Q. $t_0 = i \times 1024$
$t_1 = j \times 32$
$t_2 = K \times 4$
$t_3 = t_1 + t_0$
$t_4 = t_3 + t_2$
$t_5 = X[t_4]$
what is c & b?

$j \times 32 + i \times 1024$
$j \times 32 + i \times 1024 + K \times 4$
$t_5 = a[j \times 32 + \ldots]$

Ans $t_5 = X[i \times 1024 + j \times 32 + K \times 4]$
$= X[4(i \times 256 + j \times 8 + K)]$
b, integer array, c → 1024

$b = 8$,
$(i \times 8 \times 32 + j \times 8 + k) \times 4$
$c =$

$c = 32, b = 8$.

- **Boolean expression-**
operator - &, |, !
precedence - 'not' > 'and' > 'or'.

ex:- a and b or not c
$t_1 = \text{not } c$
$t_2 = a \text{ and } b$
$t_3 = t_2 \text{ or } t_1$
} 3AC

- **Conditional statements -**
→ if, while, for etc.
→ Jump or goto statements using addresses.

if(a<b) then 1 else 0

2 ways of doing it

way (1) 3AC

100 : if (a < b) goto 103
101 : $t_1 = 0$
102 : goto 104
103 : $t_1 = 1$
104 :

way (2) 3AC

L1 : if (a < b) goto L_t
$t_1 = 0$
goto exit
L_t : $t_1 = 1$.
exit:

ex:2. if (a < b) then X = Y + 1 else X = Y - 1

3AC

100 : if (a < b) goto 104
101 : ~~X = Y - 1~~ $t_1 = Y - 1$
102 : ~~goto 105~~ $X = t_1$.
103 : ~~X = Y + 1~~ goto 106
104 : ~~[exit or blank]~~ $t_1 = Y + 1$
105 : $X = t_1$
106 :

ex3 - while (c < d) do
if (a < b) then X = y + 1 else X = Y - 1.

100 : if (c < d) goto 102
101 : goto 109
102 : if (a < b) goto 106
103 : $t_1 = Y - 1$
104 : $X = t_1$
105 : goto ~~108~~ 100
106 : $t_1 = Y + 1$
107 : $X = t_1$
108 : goto 100
109 : exit

sniffing tools → BetterCAP, Ettercap

used to perform various MITM attacks against networks.

some more tools - Wireshark, Tcpdump, WinDump, OmniPeek.

Threat, vulnerability, risk, attack.

slide no. 62

Risk = Threat * vulnerabilities * impact

a = b + c Add b, c, reg. r = b + c
mov reg, a. a = r.

a) if ((a==b && c==d) || (e==f)) x==1;

100: if (a==b) goto 102
101: goto 104
102: if (c==d) goto 106
103: goto 108
104: if (e==f) goto 106
105: goto 108
106 ~~x==1~~ t = 1.
107 ~~to7~~ x = t
108

$\overline{a==b}$ && c==d || e==f

~~a==b~~
$\overline{a==b}$, x==d / e==f.

Page No.
Date

Backpatching

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

b) if (a==b || c==d || e==f) x==1;
c) if (a==b && c==d && e==f) x==1;

Ans (b)

100 : if (a==b) goto 102.
101 : goto 103
102 : t = 1.
103 : x = t.
104 : goto 107.
105 : if (c==d) goto 102.
106 : if (e==f) goto 102.
107 : exit.

x = a+b.
t = a+b
x = t
If

Ans (c)

100 : if (a==b) goto 102.
101 : goto 107.
102 : if (c==d) goto 104
103 : goto 107
104 : if (e==f) goto 105
105 : t = 1
106 : x = t
107 : exit

CBD
2nd lec after MSE
8/10/24

Considerat"s for VM migrat"

1) Compatibility
2) Downtime & performance
3) Data Integrity
4) Network Requirements
5) Security
6) Compliance
7) Testing & validat"

→ Isolat^ & Restrictions

→ Replicat^

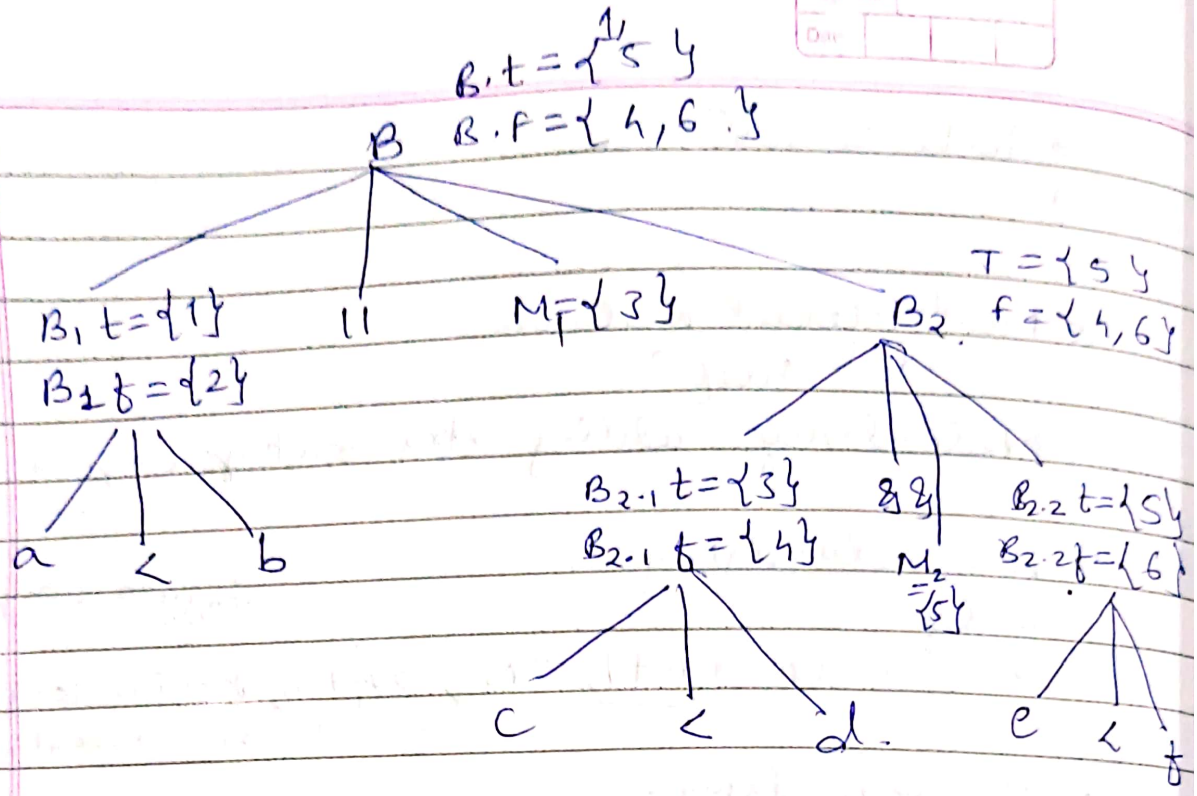
→ Dynamic reconfiguration.

* Backpatching

M = storing addr. of next instr.

ex :- if (a<b || c<d && e<f) x=0

1. if (a<b) goto 7
2. goto 3
3. if (c<d) goto 5
4. goto 9
5. if (e<f) goto 7
6. goto 9
7. t=0
8. x=t
9.



$B \rightarrow B_1 \; || \; M \; B_2 \mid B_1 \; \&\& \; M \; B_2 \mid !B_1 \mid (B_1)$

$\mid E_1 \; rel \; E_2 \mid true$

$M \rightarrow \varepsilon$

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; }

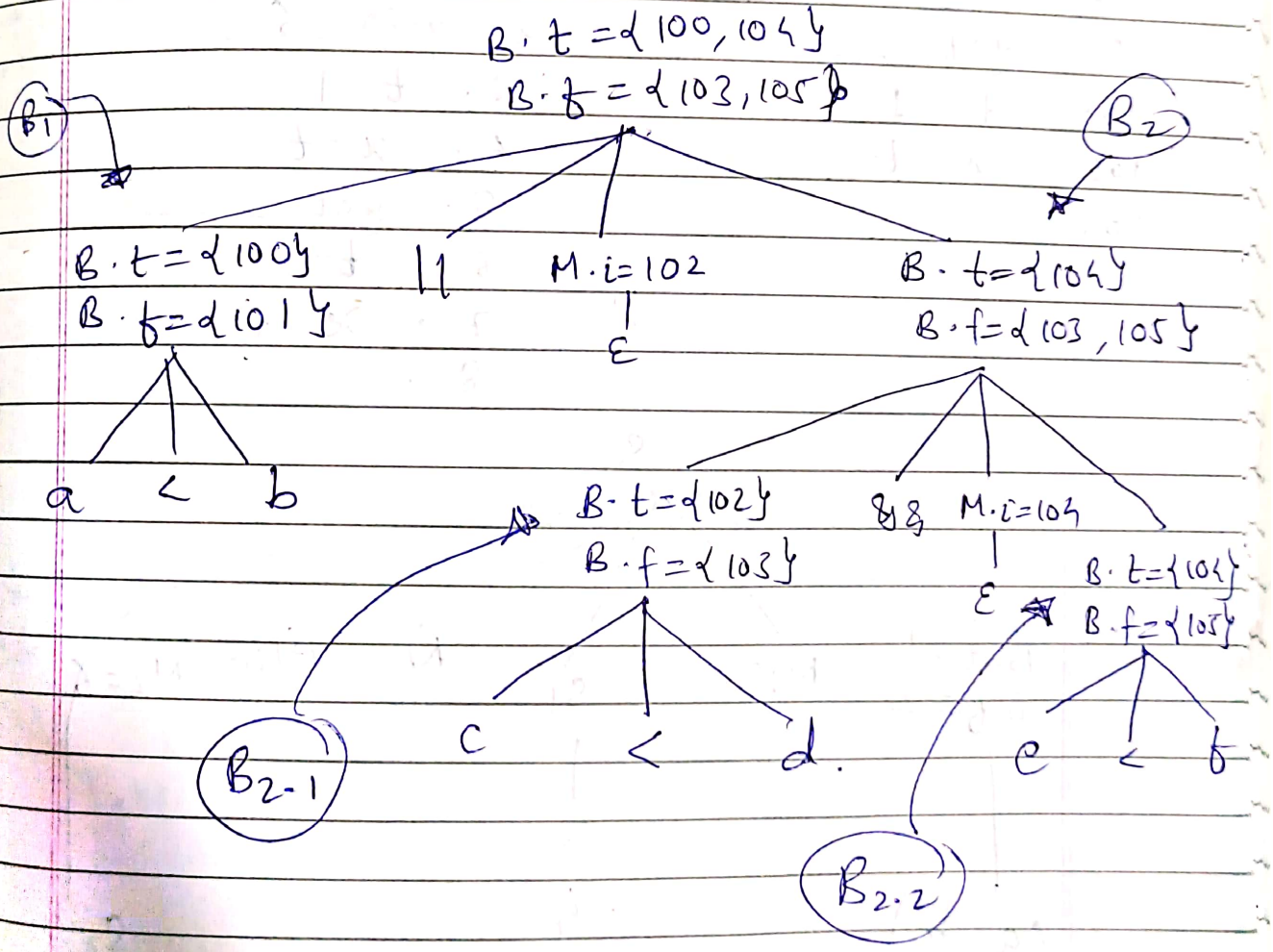
4)
5)
6) 7)

$B \rightarrow B_1 \;||\; M\, B_2$

backpatch (B_1.falselist, M.instr.)
(2, 3)

100 if (a < b) goto ____
101 goto 102
102 if (c < d) goto 104
103 goto ____
104 if (e < f) goto ____
105 goto ____
106 t = 0
107 x = t
108

$B \rightarrow B_1 \;\&\&\; M\, B_2$
backpatch (B_1.truelist, M_2.instr.
backp (3, 5)
B.truelist = B_2.truelist
B.falselist = merge
(B_1.falselist,



1) $S \rightarrow$ if $(B)\ M\ S_1$ { ba

2) $S \rightarrow$ if $(B)\ M_1\ S_1\ N$ else $M_2\ S_2$

{ backpatch $(B.truelist, M_1.instr)$

backpatch $(B.falselist, M_2.instr)$.

temp = merge $(S_1.nextlist, N.nextlist)$;

$S.nextlist$ = merge (temp, $S_2.nextlist$); }

100 if $(a=b)$ goto ____

101 ~~goto~~ $t=2$

102 $x=t$

103 goto ____

104 $t=1$

105 $x=t$

106.

1. if $(a=b)$ goto 3 ~~$x=t$~~

2. goto 6

3. $t=1$

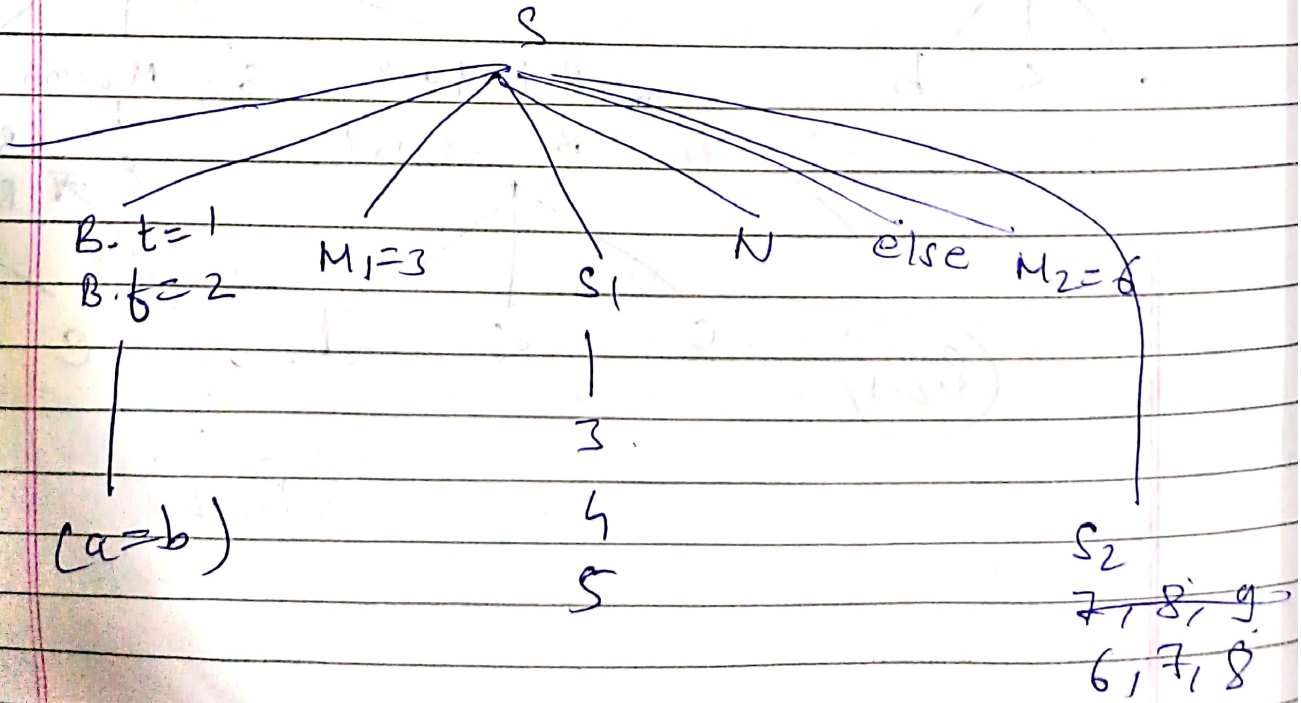
4. $x=t$

5. goto ____

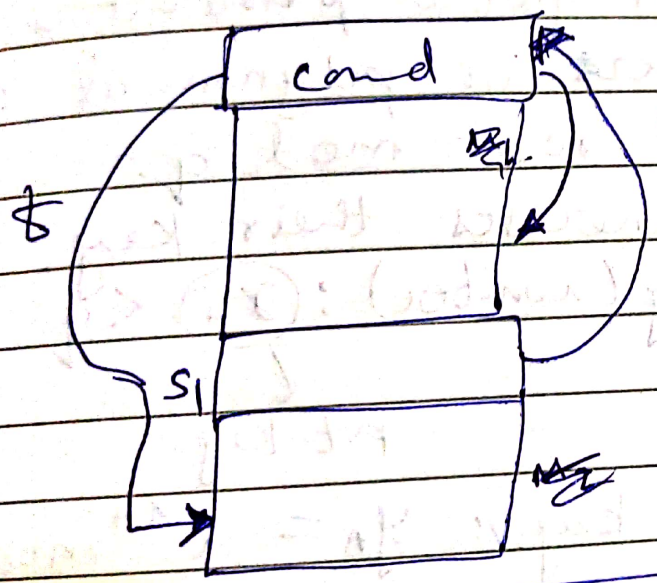
6. $t=2$

7. $x=t$.

8.



S → while M_1 (B_1) M_2 S_1
backpatch (S_1.next, M_1)
backpatch (B.truelist, M_2)



CNS
8/10/24

• Public-Key Distribut^n of secret keys -
↳ (4 ways)

→ Simple secret key Distribut^n
proposed by (Merkle)
$[N_1 || ID_A]$

ex:- $q = 353$, $\alpha = 3$

$x_A = 97$, $x_B = 233$

$y_A = 3^{97} \bmod 353$, $y_B = 3^{233} \bmod 353 = 248$

$y_A = 40$, $y_B = 248$

* Introduction to Elliptical Curves -

General form → $y^2 = x^3 + ax + b$.

nature of curve varies with a & b.

To fill target addresses → 2 pass
→ 1 pass + backpatching.

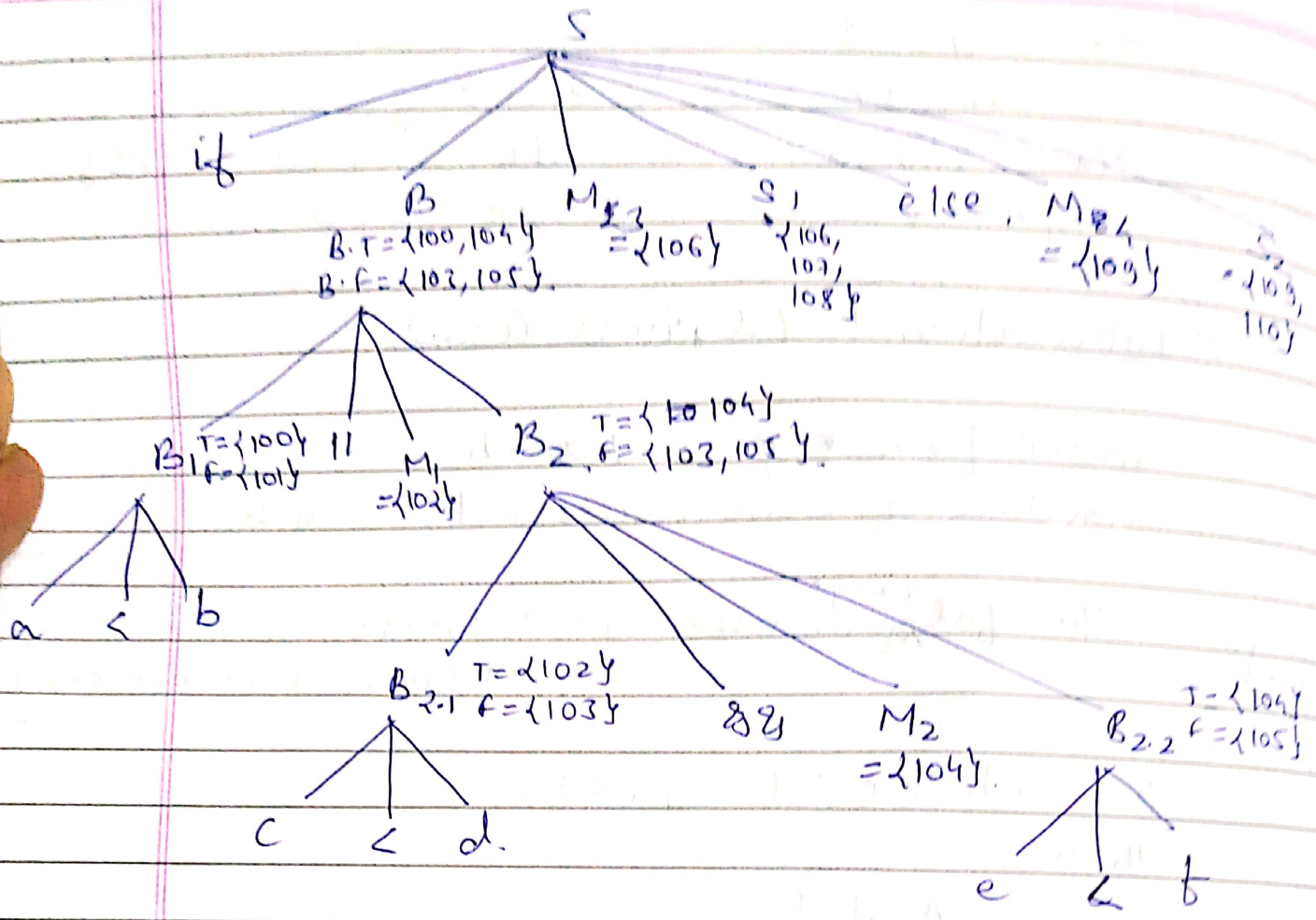
```
if (a<b || c<d && e<f)
then
        x = y - 1.
else
        x = y + 1.
```

bp(b.false, M).

```
100.  if (a<b) goto 106          [bp(B.T, M3)]
101.  goto 102.                  [bp(B1.f, M1)]
102   if (c<d) goto 104          bp(b2.true/list, M2)
103.  goto 109                   → [bp(b.f, M4)]
104.  if (e<f) goto 106          →
105.  goto 109                   → [bp(b.f, M4)]
106.  t = y - 1
107.  x = t.
108.  goto ____
109.  t = y + 1
110.  x = t
111.  goto ____
```

S1

not necessarily.



① if (c<d)
then $x = y + z$
else
$x = y - z$

② while (a<b) do
if (c<d)
$x = 1$
else
$x = 0$.

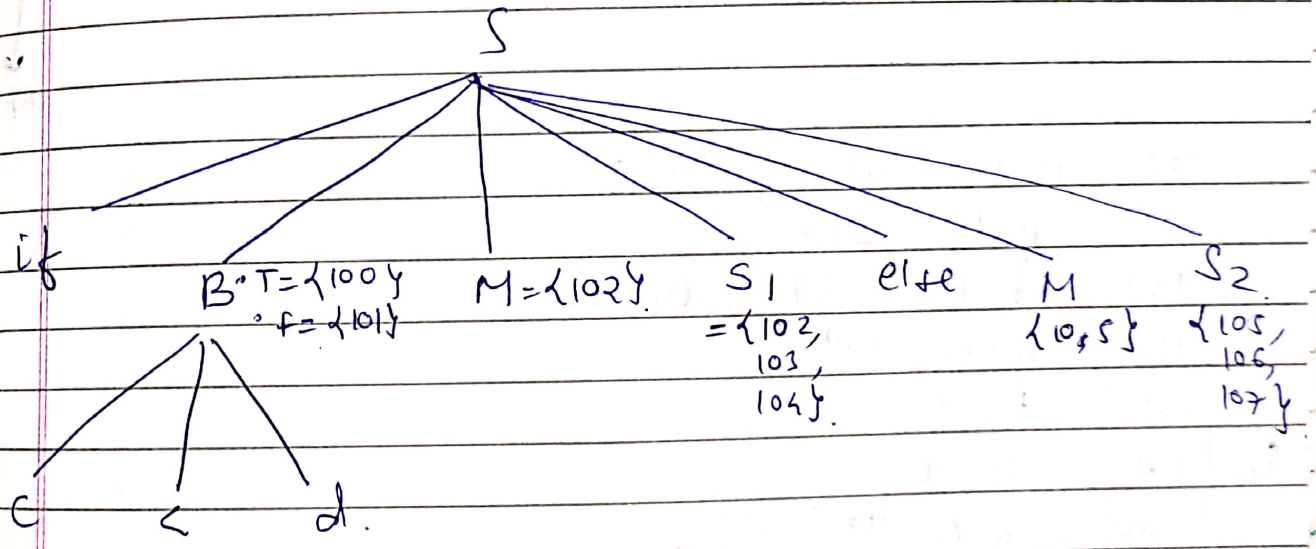
S → while M_1 E do M_2 S_1
Bp (S_1.next, M_1)
Bp (B.true, M_2).

(1). 100. if (c < d) goto 102

101. goto 105.

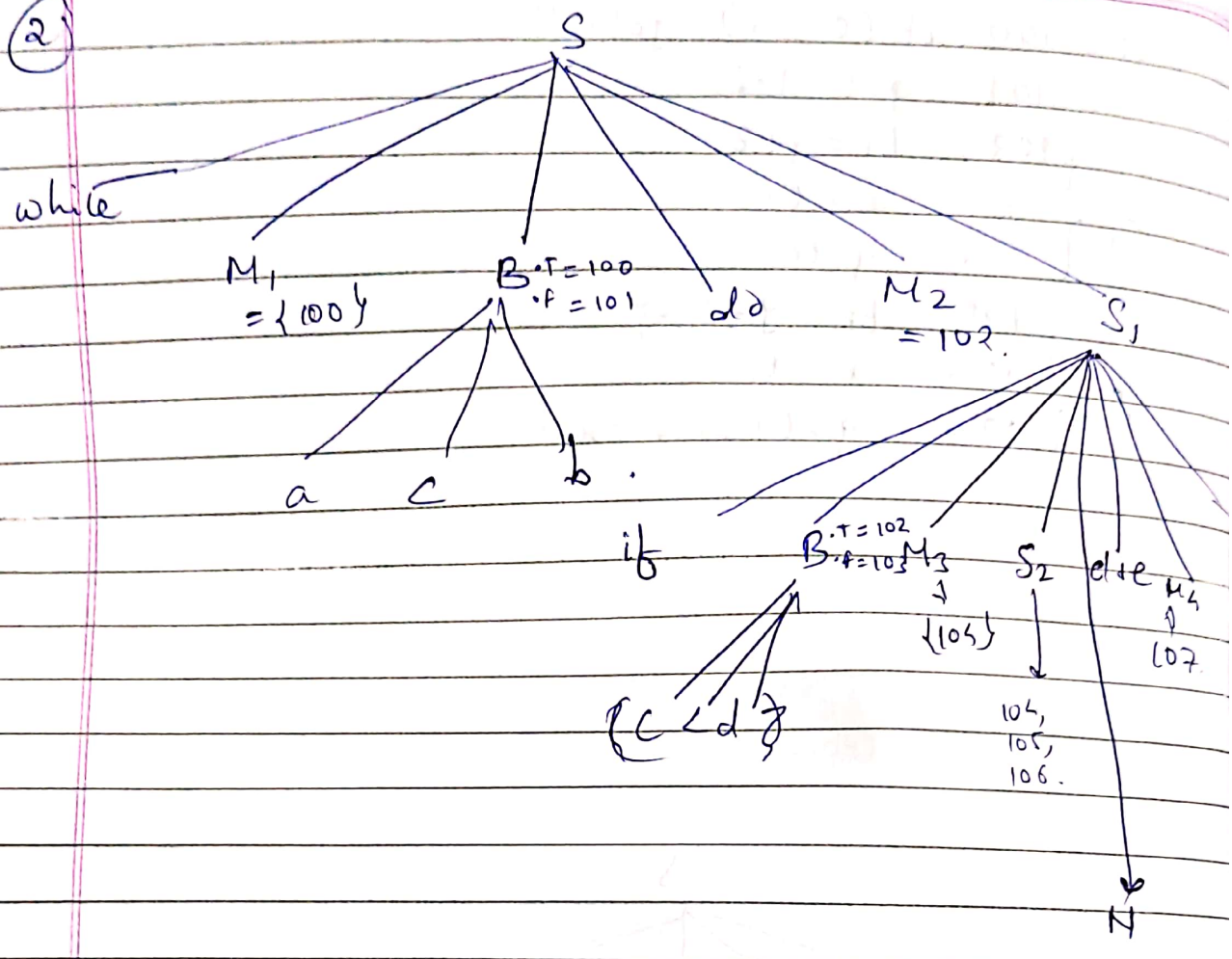
S_1 { 102. $t_1 = y + z$
103. $x = t_1$
104. goto ____

S_2 { 105. $t_1 = y - z$
106. $x = t_1$
107. goto ____ exit



(2).

(2)



100. if (a < b) goto 102

101. goto 102

102 if (c < d) goto 104

103 goto 107.

S_2 { 104 t = 1
105 x = t
106 goto ~~100~~ 100

S_3 { 107. t = 0
108 x = t
109 goto. 100