Date: __/__/__ Page No.: ____

Intermediate Code Representation

1) Postfix Notation

a + b

Postfix ⇒ ab +

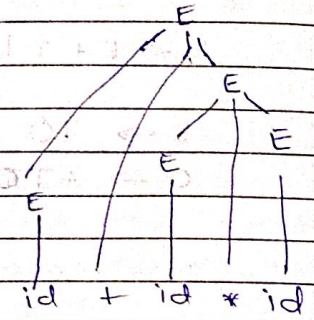
a + b * c abc * +

(This is limited to only arithmatic equations)

2) Syntax Tree

E → E+E
E → E*E
E → id

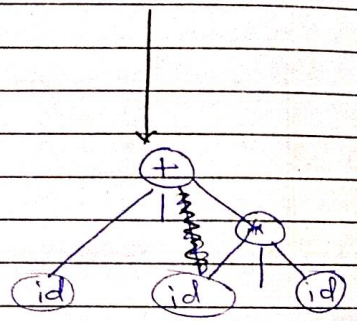
id + id * id



Syntax tree

RMD

E → E + E
E + E * E
E + E * id
E + id * id
id + id * id



(Terminal & Operator only)

Intermedete code

Date: __/__/__ Page No.: ____

Most popular way to Represent Intermediate code is

3) THREE ADDRESS CODE

- consider the following expression

x = a + b * c

⇒ t_1 = b * c
t_2 = a + t_1
x = t_2

Format
x = y OP z

OP → Operator

x y z

3 Addresses

Q Represent the following expression a + b * c + d by 3 Address statements

x = a + b * c + d	a + b * c + d
t_1 = c + d	t_1 = c + d
t_2 = b * t_1	t_2 = b * t_1
t_3 = a + t_2	t_3 = a + t_2
x = t_3	

→ * 3 Address statement for array

Format (array)
x = y [i]

Date: __/__/__ Page No.:

$x = y[i]$

The above statement is a 3 address statement with addresses x, y & i with the operator [,]

→ * 3 address statement for boolean expression

if $a < b$ goto L_1
goto L_2

Representation of 3 Address statements

1) Quadruple Representation
↳4

	Operand 1	OP	operand 2	Result
1				

In the above Representation 4 Fields are used to represent the 3 address statement

The Fields are OP, operand (1/2), Result

Date: __/__/__ Page No.:

Q Represent the following statement-
$x = ((a + b) * (-c)) / d$
by Quadruple Representation

~~Operand 1~~

(ondition (3 address)
→ only one operator
→ 3 variable = 2, 1

$t_1 = (a + b)$
$t_2 = t_1 * (-c)$
$t_3 = (t_2) / d$
$x = t_3$

$t_1 = (a + b)$
$t_2 = (t_1 - c)$
$t_3 = t_2 / d$
$x = t_3$

$t_1 = a + b$
$t_2 = -c$
$t_3 = t_1 * t_2$
$t_4 = t_3 / d$
$x = t_4$

	OP	Operand 1	Operand 2	Result
1	+	a	b	t_1
2	-	c	-	t_2
3	*	t_1	t_2	t_3
4	/	t_3	d	t_4
5	=	t_4	-	x

Date: __/__/__ Page No.: ____

2) Triple Representation

In the Quadruple representation operand 1 & operant 2 & result field are the pointer to the symbol table for the names represented of the fields.

In Triple Representation the entering of temporay ~~text~~ next into the symbol tables can be avoided by using the position of the statement to refer to a temporay value

Ques of last example

	op	op1	op2	Result
1	+	a	b	t_1
2	-	c	-	t_2
3	*	(1)	(2)	t_3
4	/	(3)	d	t_4
5	=	(4)	-	x

(Quadruple)

	op	op1	op2
1	+	a	b
2	-	c	-
3	*	(1)	(2)
4	/	(3)	d
5	=	(4)	-

(Triple)

The above table shows that the result field is not required, when it is removed the table consists of 3 fields hence the name Triple representable

Date: __/__/__ Page No.: ____

3) Indirect Triple Representation

What is the output of the following syntax directed ~~problem~~ translation scheme

$E \rightarrow E_1 + T$ { E.code = concat (E.code, T.code, "+") }

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

$T \rightarrow T_1 * F$ { T.code = concat (T_1.code, F.code, "*") }

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

$F \rightarrow id$ { F.code = getname (id.place) ; }

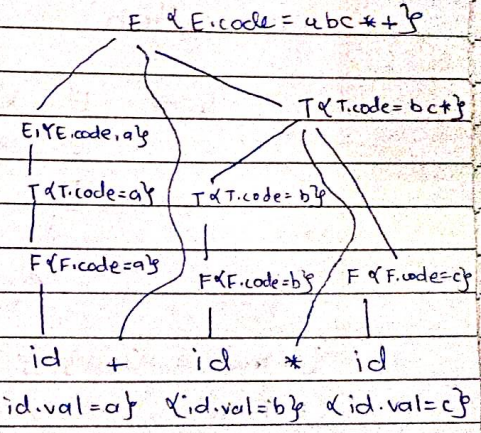
a + b * c
(id + id * id)

(a, b, c → of type id)

Ans Rightmost Derivation

$E = E_1 + T$
$= E_1 + T_1 * F$
$= E_1 + T_1 * id$
$= E_1 + F * id$
$= E_1 + id * id$
$= T + id * id$
$= F + id * id$
$= id + id * id$

Tree



Date: __/__/__ Page No.: ____

* Generation of 3 Address code for various programming construct

Example

Grammar → a < b

- $E \rightarrow E_1$ RELOP E_2 { E_3.true = makelist (nextquad) } → non-terminal

E (Extression) E.false = makelist (nextquad + 1)

predefined function the generates 3 address code

gencode (if E_1.place RELOP.val E_2.place goto _) ——(+)

gencode (goto __) }

- $E \rightarrow$ id { E.place = getname (id.place) }

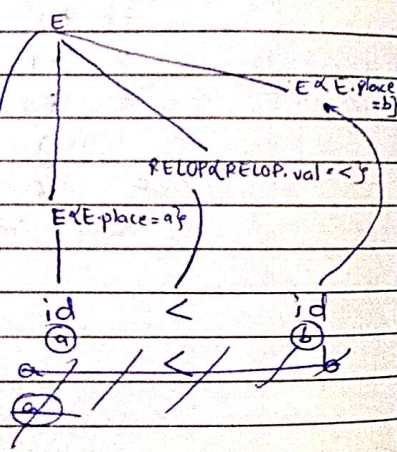
RELOP → < { RELOP.val = "<" }
| > { RELOP.val = ">" }
| <= { RELOP.val = "<=" }

Derivation

E → E RELOP E
E RELOP id
E < id
id < id
a < b

{ E.true = {100} }
{ E.false = {101} }

Trees



Initial next quad = 100
When this + is executed the nextquad value automatically increaments by 101.
Initial (nextquad = 101)

Assume that the initial value of next quad { static address of 3 address code } = 100
(Initial = 100)

100 : if a < b goto _
101 : goto _

Generate 3 address code for the following statement

a < b and c < d

Same grammar → (marker)

- $E \rightarrow E$ and M E (additional production)
- $M \rightarrow \epsilon$ { M.quad = nextquad }

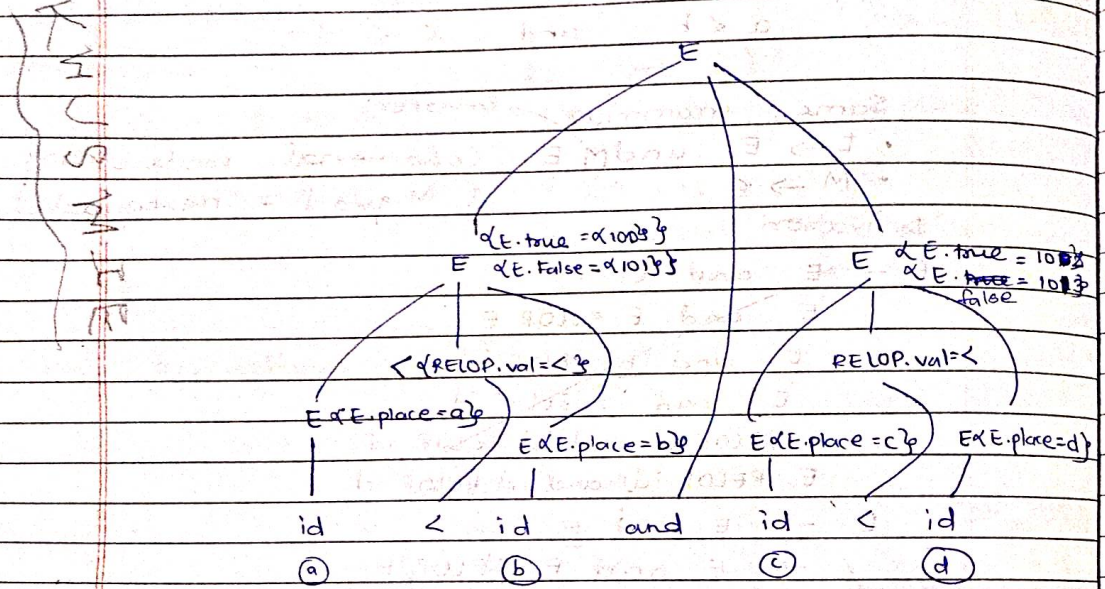
Derivation

E → E and E
E and E RELOP E
E and E RELOP id
E and id RELOP id
E RELOP E and id RELOP id
E RELOP id and id RELOP id

E → E and E
→ E and E RELOP E
→ E and E RELOP id
→ E and E < id
→ E and id < id
→ E RELOP E and id < id
→ E RELOP id and id < id
→ E < id and id < id
→ id < id and id < id

Date: __/__/__ Page No.:

E → E and E
→ E and E RELOP E
→ E and E RELOP id
→ E and E < id
→ E and id < id
→ E RELOP E and id < id
→ E RELOP id and id < id
→ E < id and id < id
→ id < id and id < id
ⓐ ⓑ ⓒ ⓓ



Date: __/__/__ Page No.

100 : if a < b goto __
101 : goto __
102 : if c < d goto __
103 : goto __

Final | nextquad : 104 |

Additional Production (same question)

E → E1 and M E2 { Backpatch (E1.true, M.quad)
E.true = E2.true
E.false = merge (E.false, E2.false) }

M → ϵ { M.quad = nextquad }

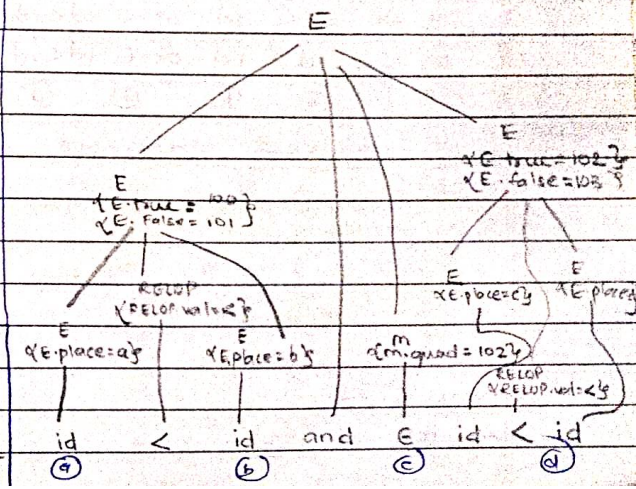
a < b and c < d

Rightmost derivation

E → E1 and M E2
→ E1 and M E RELOP E
→ E1 and M E RELOP id
→ E and M E < id
→ E and M id < id
→ E and ϵ id < id
→ E RELOP E and ϵ id < id
→ E RELOP id and ϵ id < id
→ E < id and ϵ id < id
→ id < id and ϵ id < id
ⓐ ⓑ ⓒ ⓓ

100 : if a < b goto 102
101 : goto
102 : if c < d goto __
103 : goto __
nextquad = 100 101 102 103 104

Tree



Date: __/__/__ Page No.:

Q Generate 3 Address statement for the following expression

$E \rightarrow E_1$ or M E_2 { Back patch(E_1.false, M.quad)

a < b or c < d E_1.false = E_2.false

E.true = Merge (E_1.true, E_2.true)

}

$M \rightarrow \epsilon$ { M.quad = next quad }

* Rightmost Derivation

$E \rightarrow E_1$ or M E_2

$E \rightarrow E_1$ or M E RELOP E

$E \rightarrow E$ or M E RELOP id

$E \rightarrow E$ or M E < id

$E \rightarrow E$ or M id < id

$E \rightarrow E$ or ϵ id < id

$E \rightarrow$ E RELOP E or ϵ id < id

$E \rightarrow$ E RELOP id or ϵ id < id

$E \rightarrow E <$ id or ϵ id < id

$E \rightarrow$ id < id or ϵ id < id

(a) (b) (c) (d)

3 Address statement

100 : if a < b goto ___

101 : goto 102

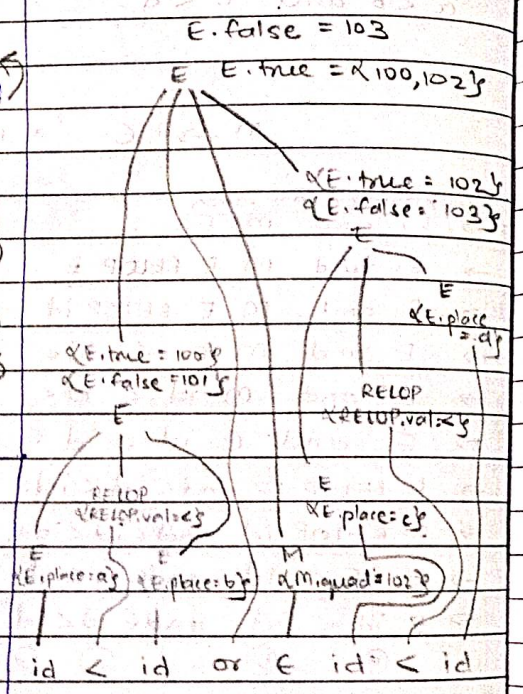
102 : if c < d goto ___

103 : goto ___

next quad

→ 100 101 102 103 104

Tree



Date: __/__/__ Page No.:

Q Write SDTS for arithmetic statement 3+5

$E \rightarrow E_1 + E_2$ { E.place = newtemp (), Gencode (" E.place = E_1.place + E_2.place) }

$E \rightarrow$ id { E.place = getname (id.place) }

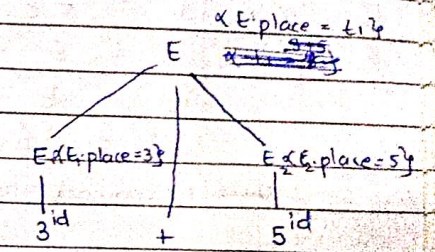
$E \rightarrow E + E$

$E \rightarrow E +$ id

$E \rightarrow$ id + id

3 Address Code

100 { t_1 = 3 + 5



$E \rightarrow E_1 * E_2$ { E.place = newtemp (), Gencode (" E.place = E_1.place * E_2.place) }

$E \rightarrow$ id { E.place = getname (id.place) }

$E \rightarrow E * E$

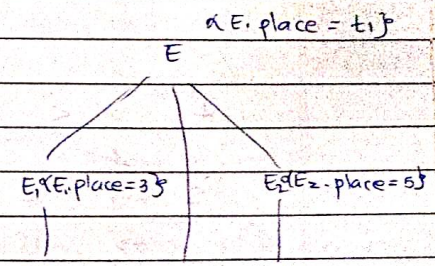
$\rightarrow E *$ id

$\rightarrow$ id * id

3 5

3 Address Code

100 : t_1 = 3 * 5



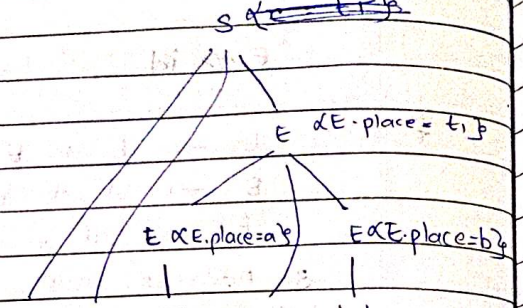
Date: __/__/__ Page No.: ____

Q SDTS for assignment statement. c = a + b

S → id = E { gencode (id.place) = E.place }

S → id = E
id = E + E
id = E + id
id = id + id

Next Nextquad = 100 101 102



100 t1 = a + b
101 c = t1

Q write SPTS for if statement

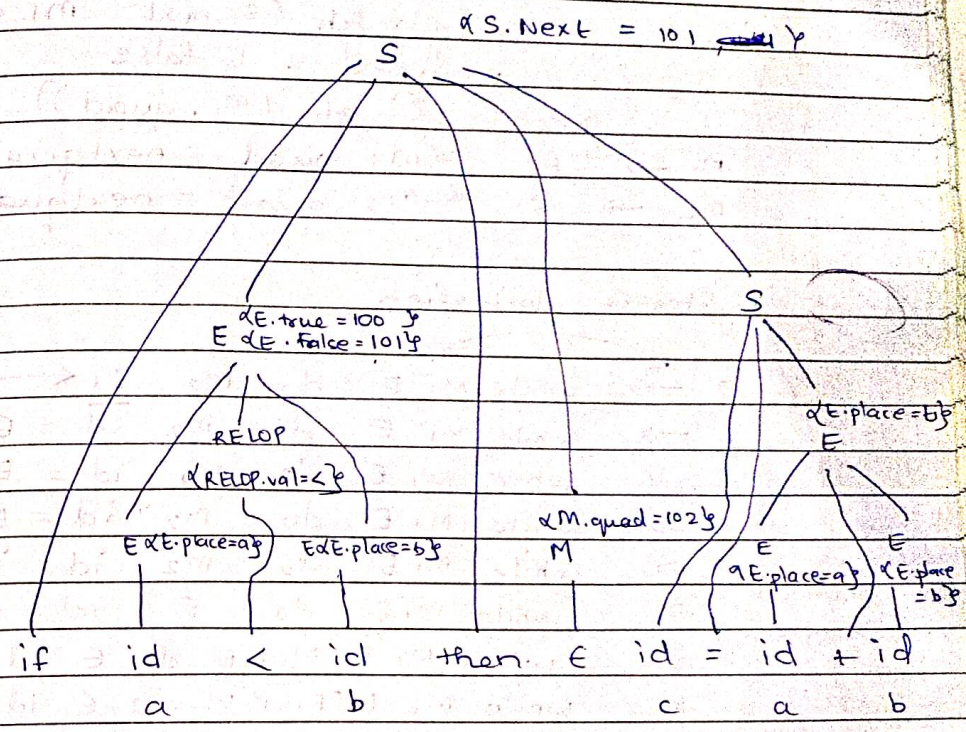
S → if E then M S
Backpatch (E.true , m.quad)
S.next = merge (E.false, S1.next)

if a < b the c = a + b

S → if E then M · S
→ if E then M id = E
→ if E then M id = E + E
→ if E then M id = E + id
→ if E then M id = id + id
→ if E then E id = id + id
→ if E RELOP E then E id = id + id
→ if E RELOP id then E id = id + id

Date: __/__/__ Page No.: ____

→ if E < id then E id = id + id
→ if id < id then E id = id + id
(if a < b then E c = a + b)



Q nextquad = 100 101 102 103 104

3 Address statement

100 : if a < b goto 102
101 : goto __
102 : t1 = a + b
103 : c = t1

5 Write SDTS scheme for while statement.

Input: while a < b do z = y + z

Grammar

S → while M_1 E do M_2 S_1 {
backpatch (E.true, M_2.quad)
backpatch (S_1.next, M_1.quad)
S.next = E.false
gencode ('goto (M_1.quad))

M_1 → ∈ {M_1.quad = nextquad}

M_2 → ∈ {M_2.quad = nextquad}

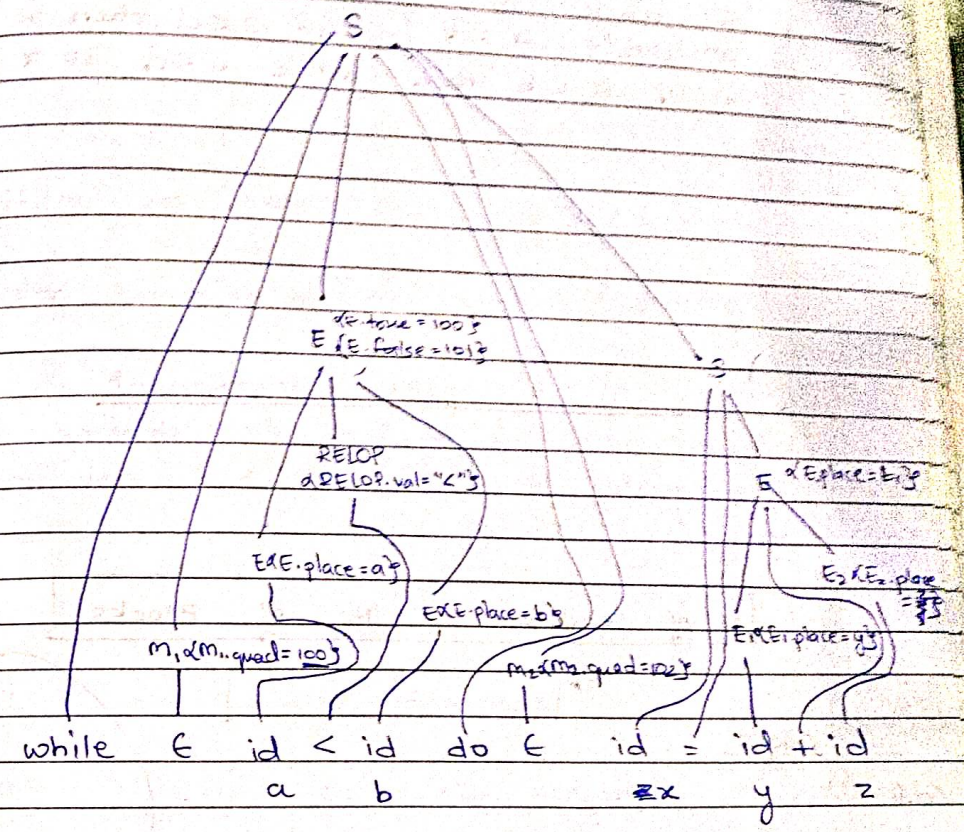
Rightmost derivation

S → while M_1 E do M_2 S_1
→ while M_1 E do M_2 id = E
→ while M_1 E do M_2 id = E_1 + E_2
→ while M_1 E do M_2 id = E_1 + id
→ while M_1 E do M_2 id = id + id
→ while M_1 E do ∈ id = id + id
→ while M_1 E RELOP E do ∈ id = id + id
→ while M_1 E RELOP id do ∈ id = id + id
→ while M_1 E < id do ∈ id = id + id
→ while M_1 id < id do ∈ id = id + id
→ while ∈ id < id do ∈ id = id + id

S	Address	Statement
	100 :	if a < b goto 102
	101 :	goto ___
	102 :	t_1 = y + z
	103 :	x = t_1
	104 :	goto 100

nextquad : 100, 101, 102, 103, 104

Derivation Tree



Date: __/__/__ Page No.: ____

Generate 3 Address Code for the following statements.

① if $a < b$ and $c < d$ then $x = y + z$
② while $a < b$ and $c < d$ do $x = y + z$
③ while $a < b$ do if $c < d$ then $x = y + z$
④ if $a < b$ then while $c < d$ do $x = y + z$

—x—

if $x < 3$ goto (100) → target ⇒ Conditional goto

goto (102) → target ⇒ unconditional goto.

* Blocks of 3 Address statement

Include all the statements starting from the leader & up to, but not including the next leader

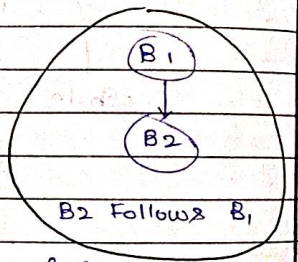
No of Leaders = No of Blocks

⇓

Blocks ≃ Nodes

Edges

Condition ① In 3 address code $i = 2$ ke bad if $i <= x$ goto 8 is comming i First statement of B_2 comes from last statement of B_1 and one more condition B_2 will follow B_1 if and only if last statement of B_1 should not be unconditional goto



29/03/23

Leader

Continue...!

Date: __/__/__ Page No.: ____

→ Flow Graph!

Q Identify the hidden statement in the 3 Address code

3 Address code for program

(1) f = 1
(2) i = 2
(3) if i <= x goto (8)
(4) f = f * i
(5) t1 = i + 1
(6) i = t1
(7) goto (3)
(8) goto calling program

(1) (3) (8) (4) → Leader

Any statement followed by the conditional goto (3) is leader the statement following is (4) ∴ (4) is also leader

- (1) → first statement
- (3) → Target goto
- (4) → immediately follows a conditional statement
- (8) → because it is a target of conditional goto statement

Block B1 :

```
f = 1
i = 2
```

Block B2 :

```
if i <= x goto(8)
```

Block B3 :

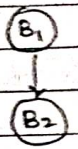
```
f = f * i
t1 = i + 1
i = t1
goto (3)
```

Block B4 :

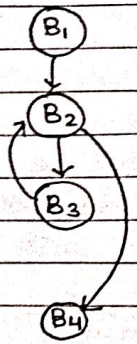
```
goto calling program
```

Date: __/__/__ Page No.:

Till Now we got



Condition ② The last statement of Block B1 is either a conditional or unconditional goto statement and first statement of Block B2 is the target of the last statement of block B1



Block 2 consists of conditional goto, True condition goes to B3 and if the condition is false then goto B4

Date: __/__/__ Page No.:

* Construct Directed Acyclic Graph (DAG) for

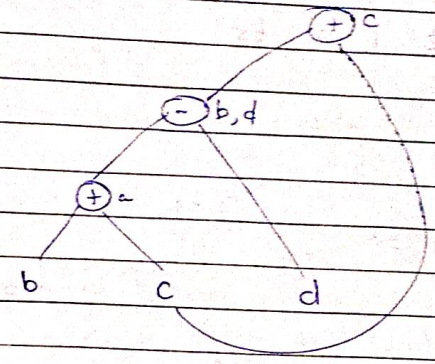
$x = y \text{ OP } z$

a = b + c

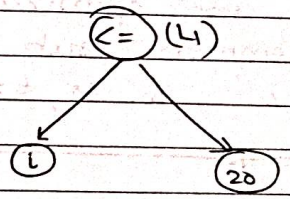
b = a - d

c = b + c

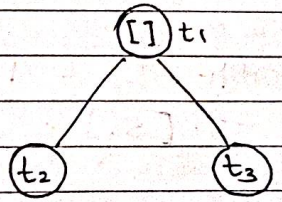
d = a - d



if (i <= 20) goto L1



$t_1 = t_2 [t_3]$



Date: __/__/__ Page No.:

$t_1 = 4 * i$

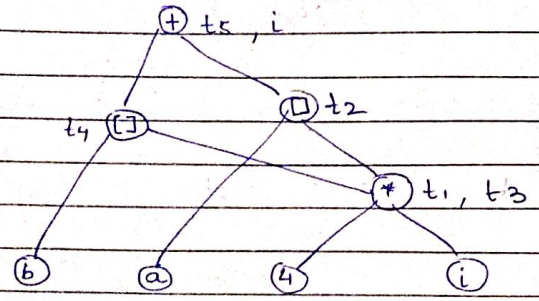
$t_2 = a\ [t_1]$

$t_3 = 4 * i$

$t_4 = b\ [t_3]$

$t_5 = t_2 + t_4$

$t\ i = t_5$



Its used for code optimization to rectify the (local) common sub expression.

i.e t_5, i & t_1, t_3

$S_1 = 4 * I$ -

$S_2 = addr(A) - 4$ -

$S_3 = S_2\ [S_1]$ - RROD = S_8

$S_4 = 4 * I$ -

$S_5 = addr(B) - 4$ -

$S_6 = S_5 - [S_4]$ -

$S_7 = S_3 * S_6$ -

$S_8 = PROD + S_7$ -

$S_9 = I + 1$

$I = S_9$

IF I < [illegible] goto (1)

Date: __/__/__ Page No.:

L_1 : if a < b goto L_2

L_3 :

may be replaced by the sequence

L_3 :

if a < b goto L_2

goto L_3