26 09.24

INTERMEDIATE CODE GENERATOR

↳ readily executed by the machine.

~~i) Syntax tree / Abstract syntax tree~~
~~ii) 3 address code~~

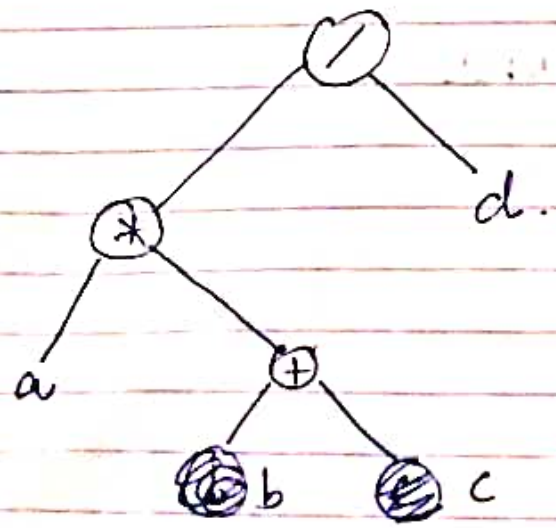
Form of I.C.

i) Syntax tree / Abstract syntax tree
ii) Postfix notation
iii) 3 address code.

Syntax tree:

- operator — parent node / internal node.
- operand — leaf node.

Q. a * (b + c) / d.



i) precedence:
ii) associativity.

26.09.24.

Postfix notation - operator after operand.

Q. $(a+b) * c$

$ab+c*$

Q. $a + (b * c)$

~~$bc+a+$~~

$abc*+$

Q. $(a-b) * (c/d)$.

$ab-cd/*$

3 address code:

- at most 3 addresses.
- only 1 operator on RHS.

→ Represented in 3 ways:

i) Quadruple
ii) Triple
iii) Indirect triple

26.09.24.

Quadruple
- (a) op
- (b) arg 1
- (c) arg 2
- (d) result.

$a = b * -c + b * -c$

```
t1 = -c
t2 = b * t1
t3 = b - c
t4 = b * t3
t5 = t2 + t4
```

(1) Quadruple.

Location	op	arg 1	arg 2	result
(0)	-	c		t1
(1)	*	b	t1(0)	t2
(2)	-	c		t3
(3)	*	b	t3(2)	t4
(4)	+	t2(1)	t4(3)	t5

Limitation.
- Stored in symbol table. (lot of memory).

26.09.24.

#1 Triple — (1) op
(2) arg1
(3) arg2

(3) Triple.

Location	op	arg1	arg2
(0)	−	c	
(1)	b*	b	(0)
(2)	−	c	
(3)	*	b	(2)
(4)	+	(1)	(3)

#1 Indirect table triple.
↳ extra pointers tables with contains pointers to the triple.

(3) Indirect triple

pointer	
1000	(0)
1002	(1)

triple.

Ptr	Triple
1001	(0)
1002	(1)
	(2)
	(3)
	(4)

..09.24.

Q. $a + b * c / e \uparrow f + b * a$.

i) Precedence
ii) Associativity

$t1 = e \uparrow f$.
$t2 = b * c$
$t3 = t2 / t1$
$t4 = b * a$
$t5 = a + t3$
$t6 = t5 + t4$.

Postfix.

$a + b * c / t1 + b * a$
$a + t2 / t1 + b * a$
$a + t3 + b * a$
$a + t3 + t4$
$t5 + t4$
$t6$.

Quadruple.

Location	op	arg1	arg2	result
(0)	↑	e	f	t1
(1)	*	b	c	t2
(2)	/	t2	t1	t3
(3)	*	b	a	t4
(4)	+	a	t3	t5
(5)	+	t5	t4	t6

Triple :

Location	op	arg1	arg2
(0)	↑	e	f
(1)	*	b	c
(2)	/	(1)	~~t1~~ (0)
(3)	*	b	a
(4)	+	a	(2)
(5)	+	(4)	(3)

26.09.24.

Indirect triple.

Pointer	Tuple
1000	(0)
1002	(1)
1004	(2)
10086	(3)
1008	(4)
1010	(5)

27.09.24

- Study from slides / Ma'am will share.
- different forms of 3 address codes.

Exceptions of 3 address codes:-

Exception 01

x = op y → arg1 field.

↓ ↓

result field operator field

arg2 field remains unused.

Exception 02.

param t1. → arg1 field.

↓

operator field.

Eg: call p, n → procedure, number of parameters

arg2 field & result field is not used.

~~sum(a,b)~~

call sum, 2.

09.24

Exception 03.

To represent unconditional & conditional jump statements, we place label of the target in the result field.

Characteristics of 3 address code's.

i) max^m of 3 addresses to represent any statement
ii) implemented as a record with the address field.

- general form:

a = b op c. ← operator

operands
(constants, names, compiler generated temporaries).

3 Address Instruction Forms:

1) Assignment Statement

x = y op z
x = op y

- assigns result obtained after solving RHS expr to left side operand.
- ~~at least~~ 1 op. on RHS.

ii) Copy Statement

x = y.

Copies/assigns value of operand y to x.

27.09.24.

iii) Conditional jump.

x relop y goto X

- x, y → operands
- relop → relational operator
- X → tag / label of target statement

→ If x relop y: gets true satisfied -
- control is sent directly to location specified by label X.
- statements inbetween is skipped.

→ If x relop y: fails -
- control not transferred.
- usual flow of execution

iv) Unconditional jump.

- X - tag/label label of target st.
- on executing, control is directly sent to loc. specified by X.
- all st b/w are skipped.

v) Procedure call.

param x call p return y.

- x → as a parameter
- p → function

* - Learn how to change while / if else to 3 address code (Imp. Eval term)

8.09.24.

Basic Blocks & Flow graph.

- Basic Blocks – is a set of statements that always executes in a sequence one after the other.

- 1 entry-point - 1 exit-point
- it should not contain any conditional / unconditional control statement in the middle of the block.

Eg: a = b + c + d. // is a basic block.

t1 = b + c
t2 = t1 + d.
& a = t2.

Eg: if A < B then 1 else 0 // Not a basic block

(1) if A < B goto (4)
(2) T1 = 0
(3) goto (5)
(4) T1 = 1
(5)

28.09.24

Partitioning intermediate code into basic blocks.

Rule 01: Determining leaders.

- 1st st. of code
- st. that is target of cond./uncond. goto st.
- st. immediately after a goto st.

Rule 02: Determining basic block.

- All st. that follows a leader till next leader is basic block.
- 1st block after 1st leader is initial block.

Q.

1. PROD = 0
2. I = 1
3. T2 = addr (A) − 4
4. T4 = addr (B) − 4
5. T1 = 4 * i
6. T3 = T2 [T1]
7. PROD = PROD + T3.
8. I = I + 1
9. if I <= 20 goto (5).
10. j = j + 1
11. K = k + 1
12. If j <= 5 goto (7)
13. i = i + j

07.24.

(1) Determining leader

1
5
7
10
13.

(2) Determining basic block

B1

```
PROD = 0
I = 1
T2 = addr(A) - 4
T4 = addr(B) - 4
```

B2

```
T1 = 4 * i
T3 = T2 [T1]
```

B3

```
PROD = PROD + T3
I = I + 1
If I <= 20 goto B2
```

B4

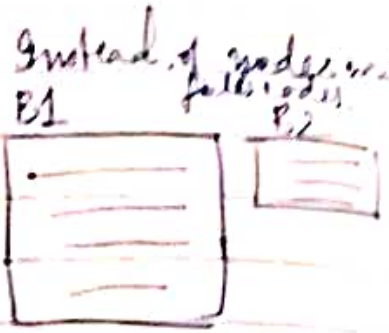
```
j = j + 1
k = k + 1
If j <= 5 goto B3
```

B5

```
i = i + j
```

28 · 09 24.

- directed graphs (Basic block + flow info).



Q.

```
1  a = 1
2  c = 1
3  t1 = 10 * a
4  t2 = t1 + c
5  t3 = 8 * t2
6  t4 = t3 - 88
7  a[t4] = 0
8  c = c + 1
9  if c <= 10 goto (3)
10 a = a + 1
11 if a <= 10 goto (2)
12 a = 1
13 t5 = c - 1
```

```
14  t6 = 88 * t5
15  a[t6] = 1.0
16  n = n + 1
17  if n <= 10 goto (13)
```

(1) Leaders.

1	1
3	2
10	3
2.	10
12.	12
13	13.

(2) B.B.

B1: `n = 1`

B2: `c = 1`

B3:
```
t1 = 10 * n
t2 = t1 + c
t3 = 8 * t2
t4 = t3 - 88
a[t4] = 0.0
c = c + 1
if c <= 10 goto B3
```

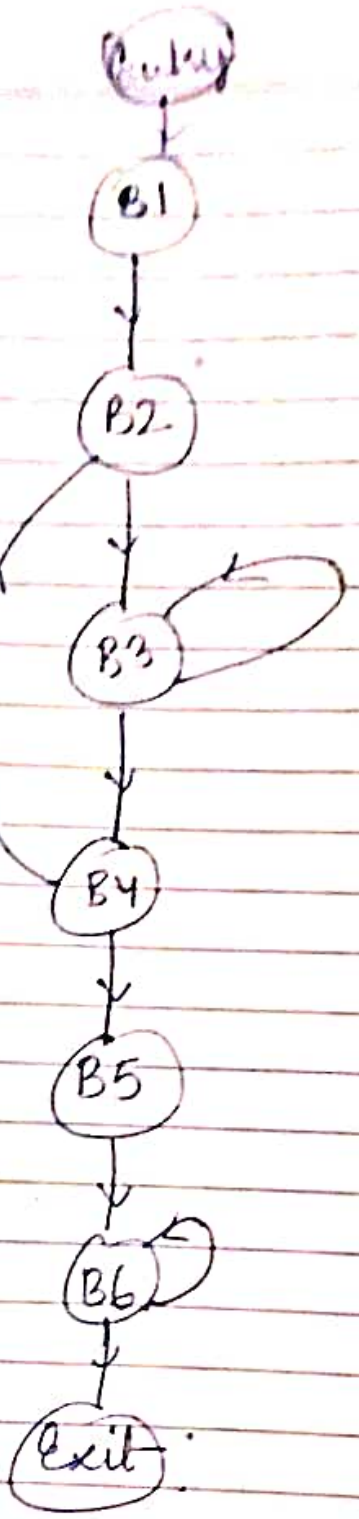
B4:
```
n = n + 1
if n <= 10 goto B2
```

B5: `n = 1`

B6:
```
t5 = c - 1
t6 = 88 * t5
a[t6] = 1.0
n = n + 1
if n <= 10 goto B6
```

Q. 07 24

Solution:



Date 09.04.24.

DAG representation

- Directed acyclic graph is a special kind of Abstract Syntax Tree.
- Each node of it contains a unique value.

- for optimising basic block
- uses 3 Address code
- dead code / sub-exp elimination

Construction of DAG.

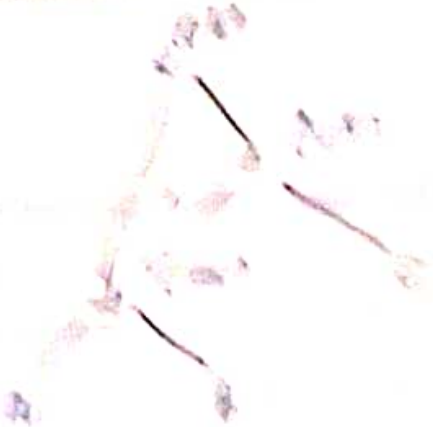
Rule 01: Interior node always represent the operators (also stores result of exp.)
Exterior node always represent identifiers/constants.

Rule 02:

Rule 03:

$x = y$. are not performed unless necessary.

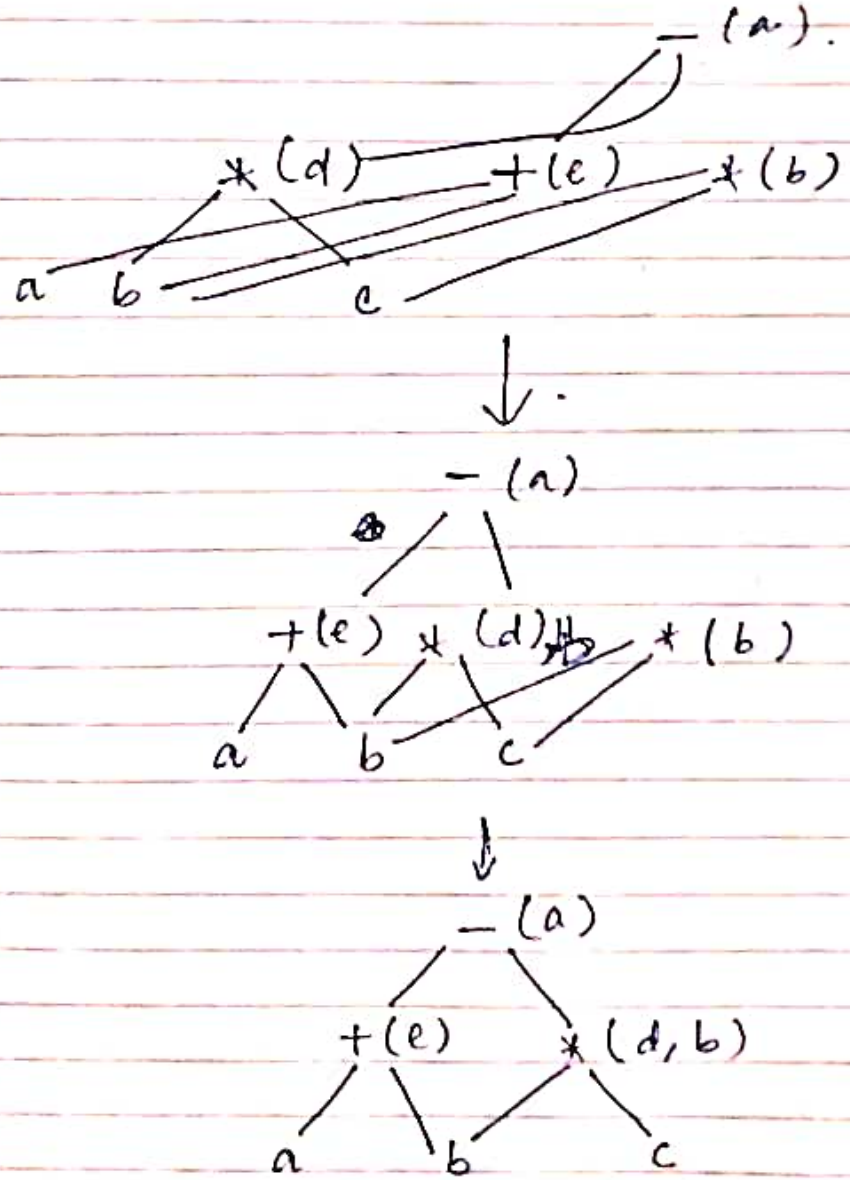
Q. $t1 = a + b$
$t2 = t1 + c$
$t3 = t1 * t2$.



9.54

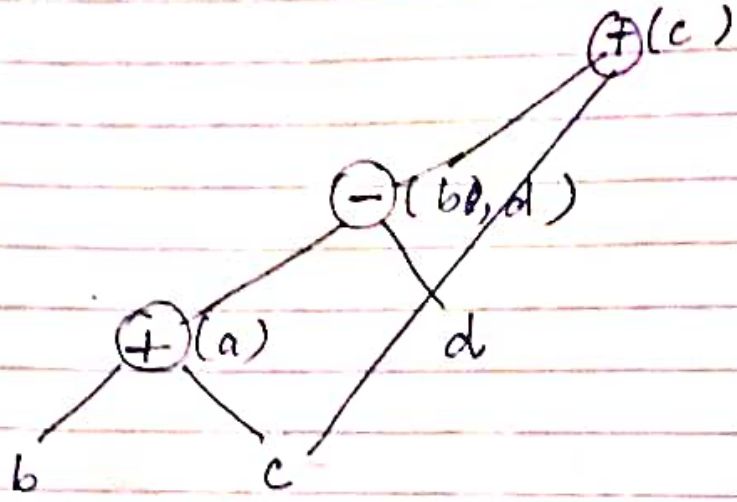
d = b * c
e = a + b
b = b * c
a = e - d.

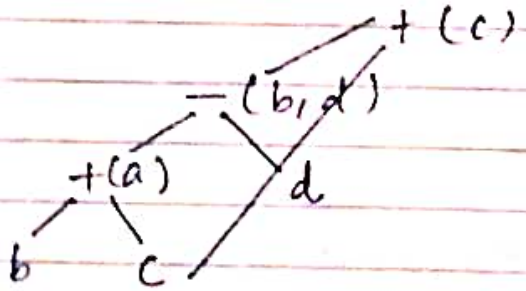
b * c



30.09.24

8. $a = b + c$
$b = a - d$
$c = b + c$
$d = a - d$





30.9.24

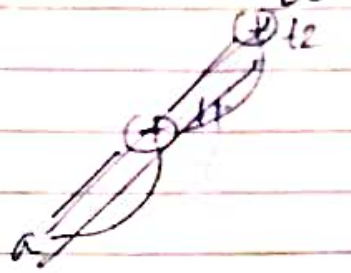
8 $(((a+a)+(a+a))+((a+a)+(a+a)))$

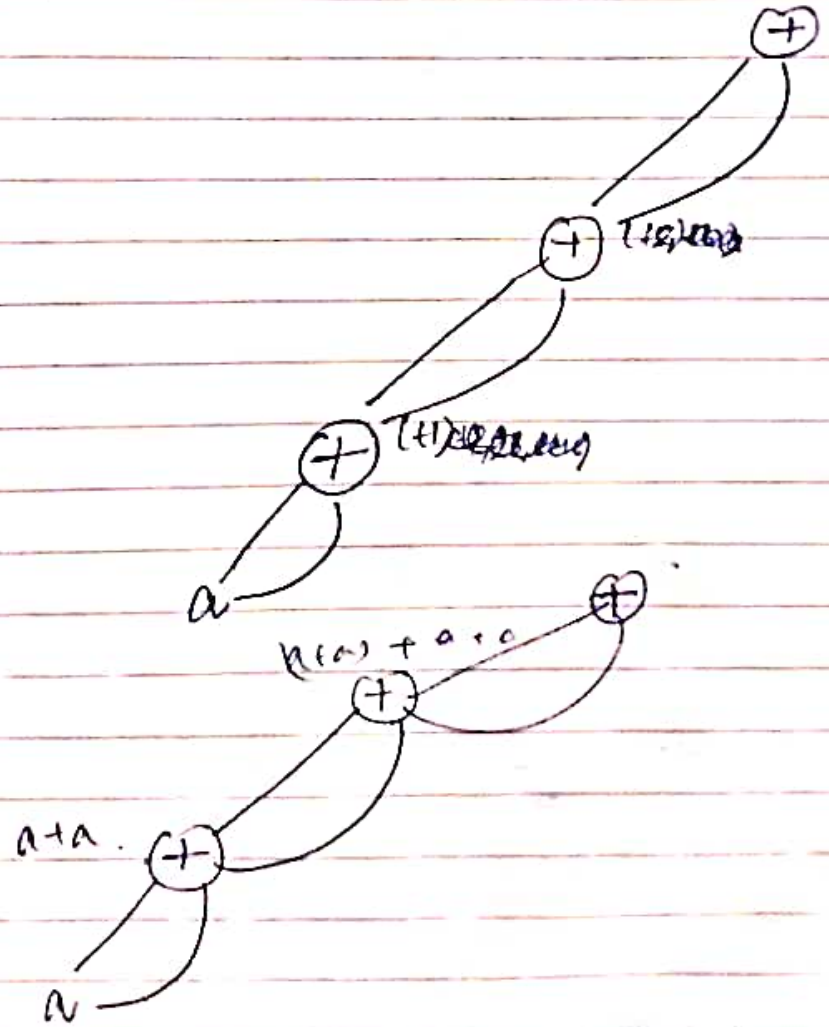
$t1$ $t2$ $t3$ $t4$ $t5$ $t6$

$t1 = a+a$
$t2 = a+a$
$t3 = a+a$
$t4 = a+a$
$t5 = t1+t2$
$t6 = t3+t4$
$t7 = t5+t6.$

optimised →

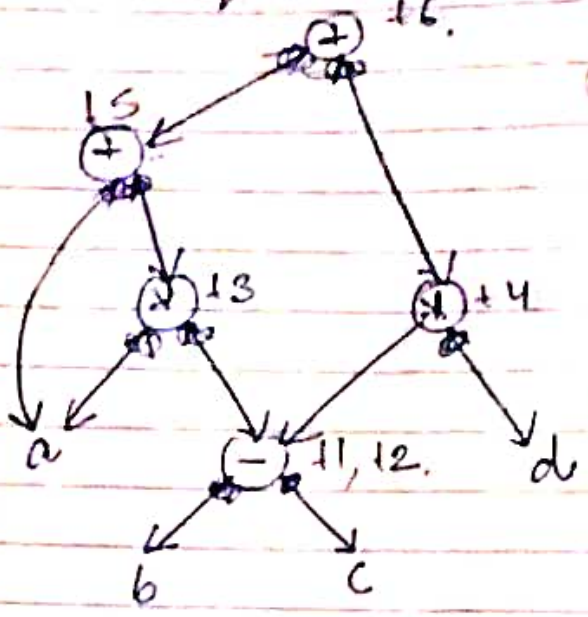
$t1 = a+a$
$t2 = t1+t1$
$t3 = t2+t2.$





30 09 24

8. a + a * (b-c) + (b-c) * d



t1 = b-c
t2 = b-c
t3 = a * t1
t4 = t2 * d
t5 = a + t3
t6 = t5 + t4

↓optimized

t1 = b-c
t2 = a * t1
t3 = t1 * d
t4 = a + t2
t5 = t4 + t3