Unit. 5

- Directed Acyclic Graph (DAG)
- Graphical representation which finds the common sub expressions.

Algorithm.

Input: Contains a basic block

Output: It contains the following information

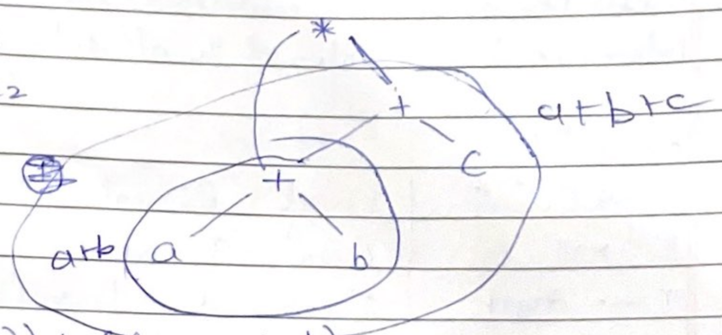
- Each node contains labels. For leaves, the label is an identifier
- Each node contains a list of attatched identifiers to hold the computed values

e.g. $(a+b) * (a+b+c)$

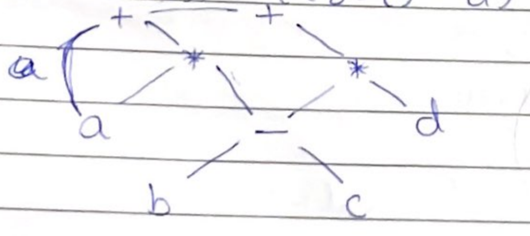
TAC : $t_1 = a+b$

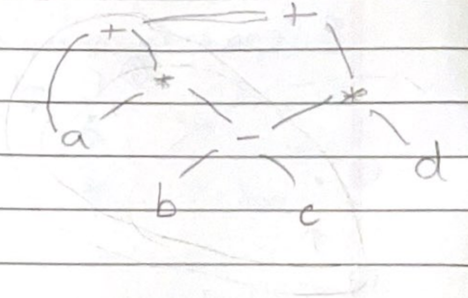
$t_2 = t_1 + c$

$t_3 = t_1 * t_2$



e.g. $(a+a*(b-c)) + ((b-c)*d)$





e.g. $a = b+c$

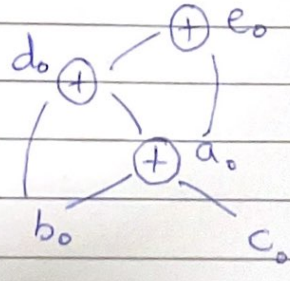
$d = b+a$

$e = d+a$

$a_0 = b_0 + c_0$

$d = b_0 + a_0$

$e_0 = d_0 + a_0$



e.g $a = b + c$

$b = a - d$

$c = b + c$

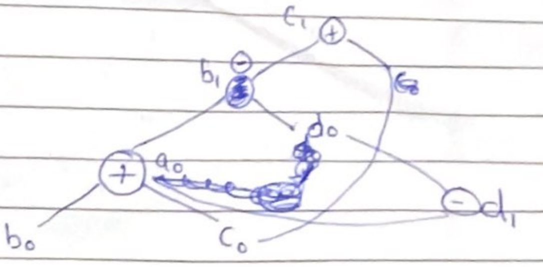
$d = a - d$

$a_0 = b_0 + c_0$

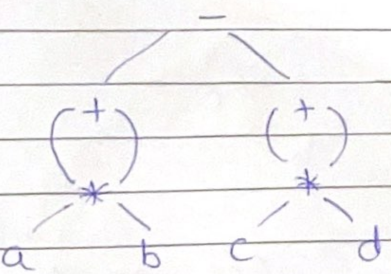
$b_1 = a_0 \overset{-}{\cancel{*}} d_0$

$c_1 = b_1 + c_0$

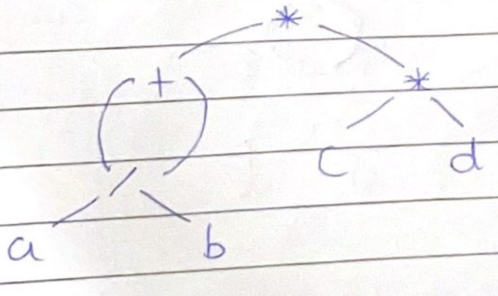
$d_1 = a_0 - d_0$



e.g. ((a*b) + (a*b)) − (((c*d) + (c*d))

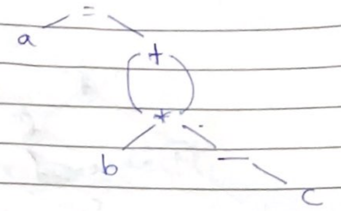


e.g. (a/b) + (a/b) * (c*d)

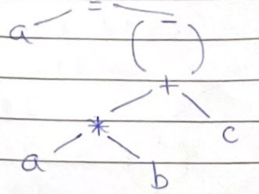


- DAG represents the structure of basic block
Leaf nodes represent identifiers, constants
Internal nodes represent result of expression

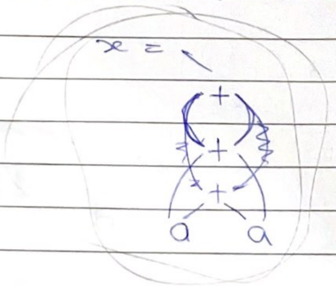
e.g. a = b* - c + b * - c

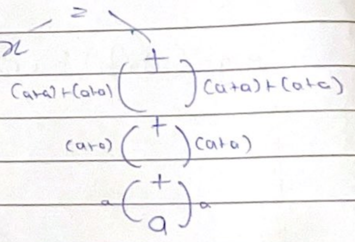


e.g. a = (a*b+c) - (a*b+c)

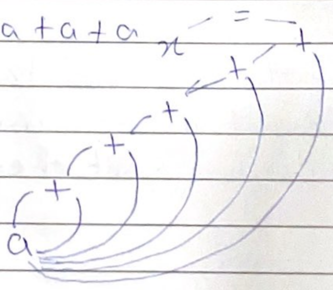


x =
e.g. (((a+a) + (a+a)) + ((a+a) + (a+a)))



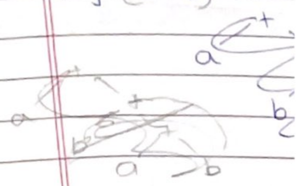


e.g. x = a+a+a+a+a+a



- Value-Number Me

e.g. (a+b)+(a+



1	id
2	id
3	+
4	+
5	+

P1 = mkleaf (

P2 = mkleaf

P3 = ~~mkleaf~~ mknode

P4 = mkleaf (

P5 = mkleaf

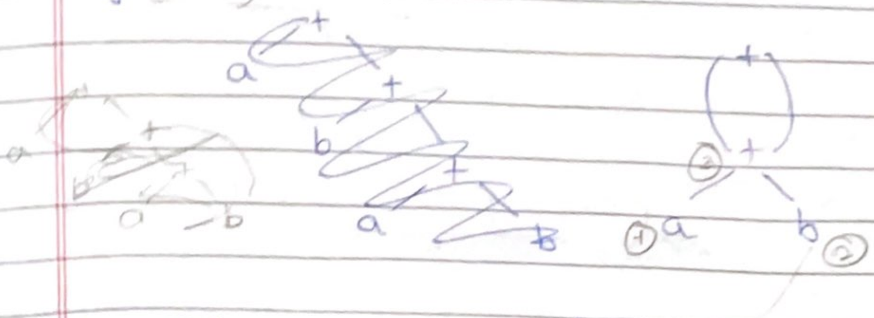
P6 = mknod

P7 = mknod

e.g. a + a

- Value-Number Method:

e.g. (a+b)+(a+b)



~~1~~	~~id~~	~~a~~	
~~2~~	~~id~~	~~b~~	
~~3~~	~~+~~	~~1~~	~~2~~
~~4~~	~~+~~	~~2~~	~~3~~
~~5~~	~~+~~	~~1~~	~~4~~

1	id	a	
2	id	b	
3	+	1	2
4	+	3	3

P_1 = mkleaf (id, entry - a)

P_2 = mkleaf (id, entry - b)

P3 = ~~mkleaf~~ mknode ("+", P1, P2)

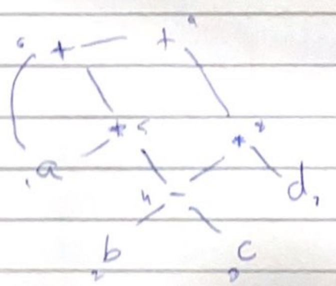
P4 = mkleaf (id, entry - a) = P1

P5 = mkleaf (id, entry - b) = P2

P6 = mknode("+", P4, P5) = P3

P7 = mknode ("+", P3, P6)

e.g. a + a * (b - c) + (b - c) * d

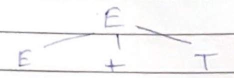


1	id	a	
2	id	b	
3	id	c	
4	~~id~~ -	~~d~~ 2	3
5	*	1	4
6	+	1	5
7	id	d	
8	*	4	7
9	+	6	8

- SDD to generate 3-address code:

$S \rightarrow id = \{E\}$	{gen(idname = E.place)}
$E \rightarrow E_1 + T$	{E.place = new temp(), gen (E.place = E_1.place + T.place
$E \rightarrow T$	{E.place = T.place}
$T \rightarrow T_1 * F$	{T.place = new temp(), gen(T place = T_1.place + F.place)}
~~[illegible]~~ $T \rightarrow F$	{T.place = F.place}
~~[illegible]~~	~~[illegible]~~
$F \rightarrow F_1 - G$	{F.place = new temp(), gen (F.place = F_1.place + G.place)}
$F \rightarrow G$	{F.place = G.place}
$G \rightarrow G_1 / H$	{G.place = new temp(), gen (G.place = G_1 place + H.place)}
$G \rightarrow H$	{G.place = H.place}
$H \rightarrow id$	{H.place = id.name}

* $(a+b) * (c+d) - (a/b/c)$



- 3-address code

e.g. $-(a*b) + (c+d)$

$t_1 = a * b$

$t_2 =$ uminus t_1

$t_3 = c + d$

$t_4 = t_2 + t_3$

$t_5 = a + b$

$t_6 =$ ~~[illegible]~~ t_5

$t_7 = t_4 - t_6$

e.g. if $A < B$

then 1

else 0

(1) if $(A < B$

(2) $T1 = 0$

(3) goto(5

(4) $T1 = 1$

(5)

e.g. $a = b + c +$

$t_1 = b +$

$t_2 = t_1$

$a =$ ~~[illegible]~~

e.g. if $a <$

(1)

(2)

(3)

(4)

(5

(

e)}

gen (E.place = E_1.place + T.place)}

n(T.place = T_1.place + F.place)}

(F.place = F_1.place + G.place)}

G.place = G_1.place + H.place)}

- 3 - address code

e.g. - (a * b) + (c + d) - (a + b + c + d)

$t_1 = a * b$

t_2 = uminus t_1

$t_3 = c + d$

$t_4 = t_2 + t_3$

$t_5 = a + b$

t_6 = $t_5 + t_3$

$t_7 = t_4 - t_6$

e.g. if A < B
    then 1
    else 0

(1) if (A < B) goto (4)

(2) T1 = 0

(3) goto (5)

(4) T1 = 1

(5)

e.g. a = b + c + d

$t_1 = b + c$

$t_2 = t_1 + d$

a = ~~t_3~~ t_2

e.g. if a<b and c<d then t = 1 else t = 0

(1) if (a < b) goto (2)

(2) goto (4)

(3) if (c < d) goto (6)

(4) ~~goto (8)~~ t = 0

(5) goto (7)

(6) t = 1

(7)

- Quadruples, Triples & Indirect triple:

e.g. $a+b*c/e \wedge f)+b*a$

$t_1 = e \wedge f$

$t_2 =$ ~~a~~ $b*c$

$t_3 = t_2 / t_1$

$t_4 = b*a$

$t_5 = a + t_3$

$t_6 = t_5 + t_4$

Quadruple:

Position Location	Op	Arg1	Arg2	Result
(0)	^	e	f	t_1
(1)	*	b	c	t_2
(2)	/	t_2	t_1	t_3
(3)	*	b	a	t_4
(4)	+	a	t_3	t_5
(5)	+	t_5	t_4	t_6

Triples:

Position Location	OP	Arg1	Arg2
(0)	^	e	f
(1)	*	b	c
(2)	/	(1)	(0)
(3)	*	b	a
(4)	+	a	(2)
(5)	+	(4)	(3)

Indirect triples:

	Statement
35	(0)
36	(1)
37	(2)
38	(3)
39	(4)
40	(5)

e.g. $-(a*b) + (c*d+e)$

$t_1 = a*b$

$t_2 = \text{uminus } t_1$

$t_3 = c*d$

$t_4 = t_0 + e$

$t_5 = t_2 + t_4$

Quadruple:

Position	Op	Arg1	Arg1	Result
(0)	*	a	b	t_1
(1)	uminus	t_1		t_2
(2)	*	c	d	t_3
(3)	+	t_3	e	t_4
(4)	+	t_2	t_4	t_5

Triple:

Position	Op	Arg1	Arg2
(0)	*	a	b
(1)	uminus	(0)	
(2)	*	c	d
(3)	+	(2)	e
(4)	+	(1)	(3)

Indirect triple:

	Statement
100	(0)
101	(1)
102	(2)
103	(3)
104	(4)

- Code Generation: $d = (a+b) - (a-c) + (a-c)$

e.g. $t = a+b$
$u = a-c$
$v = t-u$
$d = v+u$

Statement	Target Code	Register descriptor	Address descriptor
$t = a+b$	Mov a, R_0 Add b, R_0	R_0 contains t	t is present in R_0
$u = a-c$	Mov a, R_1 Sub c, R_1	R_1 contains u	u is present in R_1
$v = t-u$	Sub R_1, R_0	R_0 contains ~~R~~ v	v is present in R_0
$d = v+u$	Add R_0, R_1 Mov d, R_1	R_0 contains v R_1 contains d	v is present in R_0 d is present in R_1 & also in memory

e.g. $t = a-b$, $u = a-c$, $v = t+u$, $a = d$, $d = v+u$

Statement	Target Code	Register descriptor	Address descriptor
$t = a-b$	Mov a, R_0 Sub b, R_0	R_0 contains t	t is present in R_0
$u = a-c$	Mov a, R_1 Sub c, R_1	R_0 contains t R_1 contains u	t is present in R_0 u is present in R_1
$v = t+u$	Add R_1, R_0	R_0 contains v	v is present in R_0
$a = d$			

Code generation

t = a - b
u = a - c
v = t + u
a = d
d = v + u

R_1	R_2	R_3

a	b	c	d	t	u	v
a	b	c	d			

t = a b
Mov a, R_1
Mov b, R_2
Sub R_2, R_1, R_2

a	t	

a, R_1	b	c	d	R_2		

u = a - c
Mov c, R_3
Sub R_1, R_1, R_3

u	t	c

a	b	c, R_3	d	R_2	R_1	

v = t + u
Add R_3, R_2, R_1

u	t	v

a	b	c	d	R_2	R_1	R_3

a = d
Mov R_2, d

u	a, d	v

a, R_2	b	c	a, R_2		R_1	R_3

d = v + u
Add R_1, R_1, R_3

d	a	v

R_2	b	c	R_1			R_3

exit
ST a, R2
d, R_1

d	a	v

a, R_2	b	c	d, R_1			R_3