Three Address Code

Intermediate Code Generation

Assumption: we have a validated Abstract Syntax tree (AST)

↳ The AST itself is a intermediate form, however we do this stage to facilitate machine dependent/independent optimization more.

Traverse the AST and generate three address codes for all statements and expressions. And the ordering of the three address code lines will follow the ordering of nodes in the post order traversal of the AST.

type of instruction in RISC:

① R-type

② I-type

③ J-type

(b) else is not used since the circuit complexity will slow down the CPU.

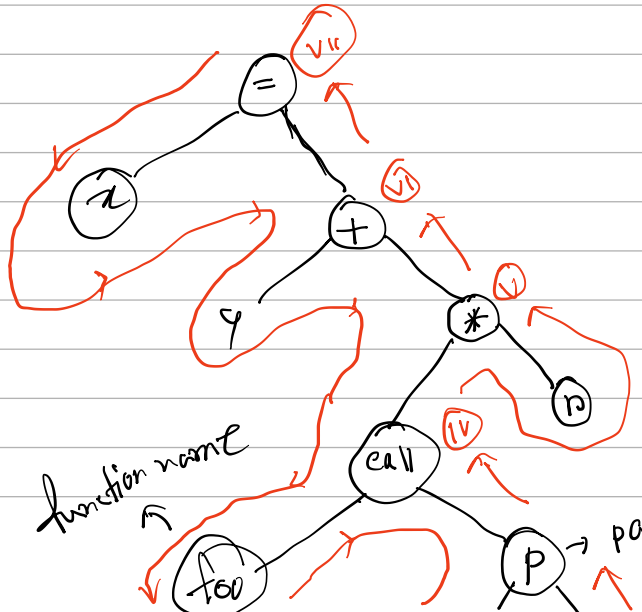
* by doing one or more post order traversal on the AST we can validate based on the type and scope informations

opcode destination, source1, source2

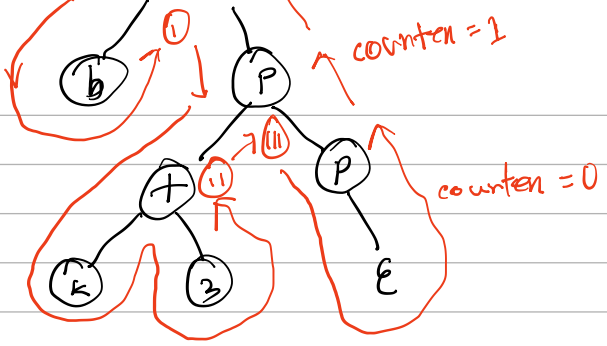
atmost three addresses can be worked with

* we can use as many as temporary variable we want.

* $x = y + foo(b, k+3) * r;$



call resets the counter: 0.



P → parameter

P → ε

P → E , P

E → E+E

E → E*E

E → id

E → num

P → ε { counter = 0 }

P → E { param E.address }, P { counter ++ }

E → E+E

E → E*E

E → id

E → num

(I) param b

(II) $t_1 = k + 3$

(III) param t_1

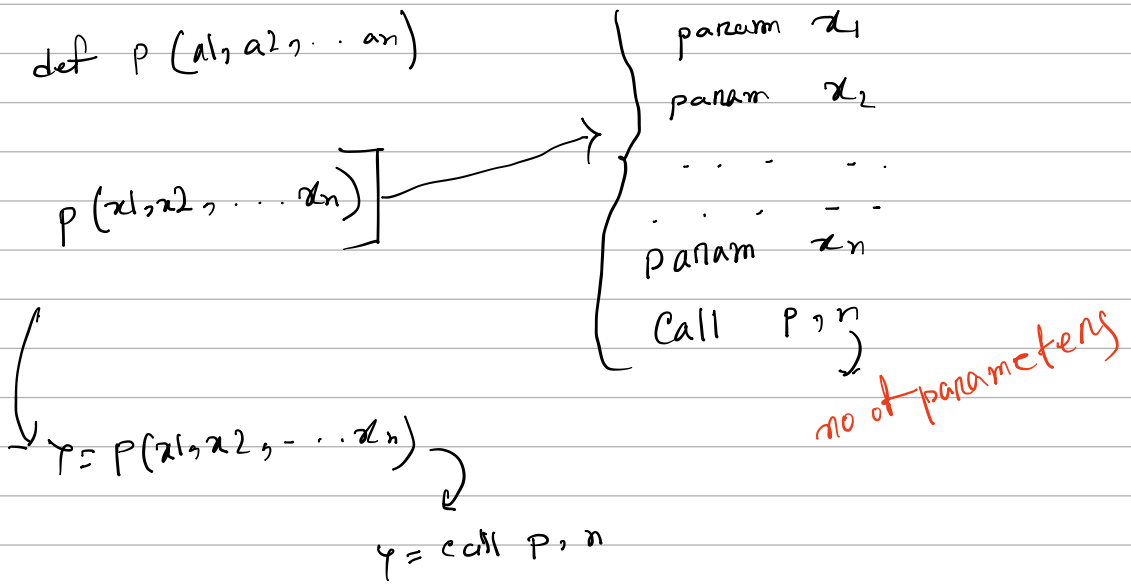
(IV) t_2 = call foo , 2

(V) $t_3 = t_2 * p$

(VI) $t_4 = y + t_3$

(VII) $x = t_4$

Though we are not storing in code but the parent of the call is a arithmetic operation that's why it will automatically be stored.



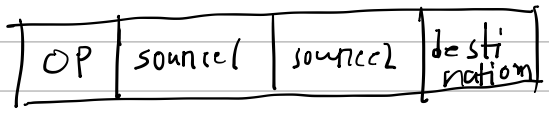
* Three address codes can only address one dimensional arrays due to the three address limitation.

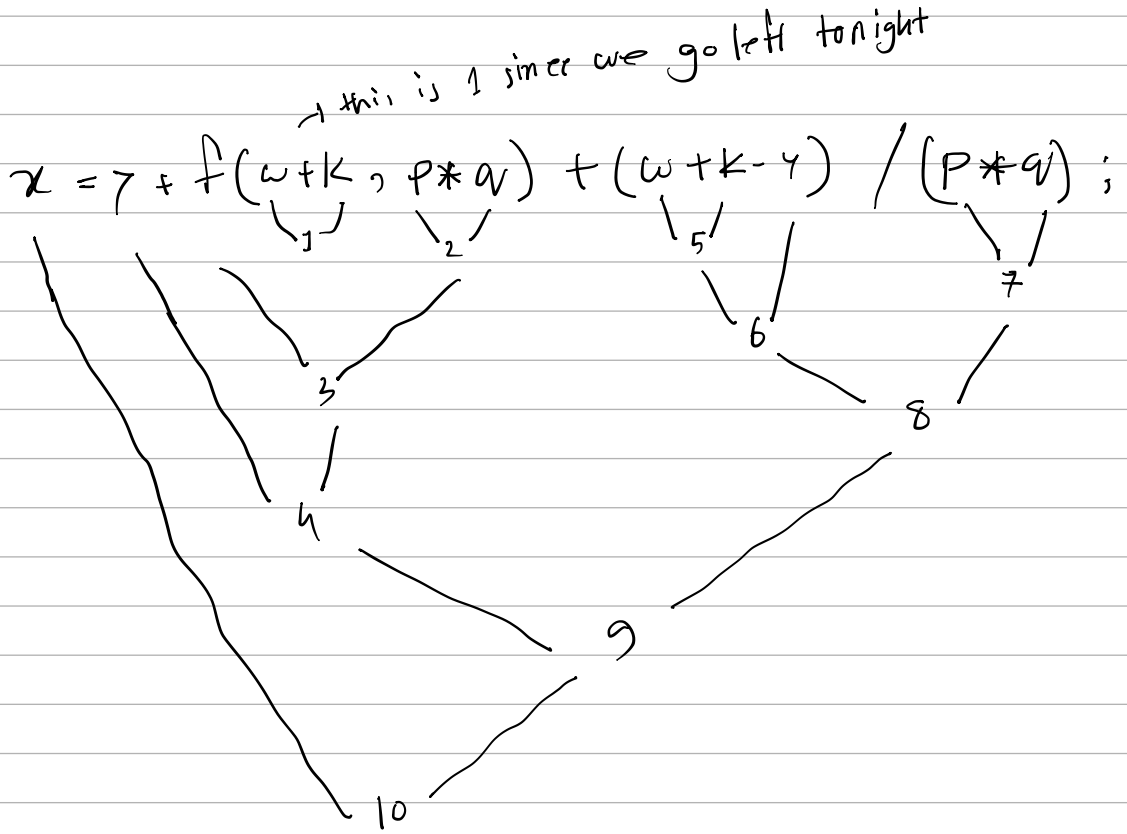
* The three address code order will follow the order of traversal of AST.

Lecture 16

Quadruple table

each row is a three address code, there are four columns per row.





$t_1 = w + k$

$t_2 = p * q$

param t_1

param t_2

t_3 = call f, 2

$t_4 = 7 + t_3$

op	src1	src2	dest
+	w	k	t_1
*	p	q	t_2
param	t_1		
param	t_2		
call	f	2	t_3
+	7	t_3	t_4

$t_5 = w + k$

$t_6 = t_5 - 7$

$t_7 = p * q$

$t_8 = t_6 / t_7$

$t_9 = t_4 + t_8$

$x = t_9$

+	w	k	t_5
−	t_5	7	t_6
*	p	q	t_7
/	t_6	t_7	t_8
+	t_4	t_8	t_9
copy	t_9		x

op	src1	src2	dest
+	w	k	t_1
*	p	q	t_2
param	t_1		
param	t_2		
call	f	2	t_3
+	7	t_3	t_4
+	w	k	~~t_5~~ t_1
−	~~t_5~~ t_1	7	t_6
*	p	q	~~t_7~~
/	t_6	~~t_7~~ t_2	t_8
+	t_4	t_8	t_9
copy	t_9		x

I already have a temp value that does the same.

w & k are not in any destination column before. This means means w & k's values are unchanged.

∴ We don't need t_5.

This can propagate as well.

$f((w + k + 7) / k)$

(w+k+7)

1 / 2

triple format :

getting rid of temporary variables.

	OP	SRC1	SRC2
0	+	w	k
1	*	p	q
2	param	(0)	
3	param	(1)	
4	call	f	2
5	+	y	(4)
6	−	(0)	y
7	/	(6)	(1)
8	+	(5)	(7)
9: x	copy	(8)	x

special instruction

but we can't store

like this : do this

This was used before to save memory but not used now.