Implmenting Three address code using | Quadruples | Triples & | Indirect Triples

#1. Quadruples

Let the expression be $P\_new = P + ((P * n * r) / 100)$.

It's equalent three address code is:

$t_1 = n * r$
$t_2 = P * t_1$
$t_3 = t_2 / 100$
$t_4 = P + t_3$
$P_{new} = t_4$

" Quadruple is a structure which consists of 4 fields namely ~~arg1 & arg2 &~~ opr, arg1, arg2 and res.

where: opr = operator
arg1 & arg2 = operands
res = stores the result of the expression "

some address

@	Opr	Arg1	Arg2	Result
(0)	*	n	r	t_1
(1)	*	P	t_1	t_2
(2)	/	t_2	100	t_3
(3)	+	P	t_3	t_4
(4)	=	t_4		P_new

Merits	Demerits
~ Easy to rearrange code for global optimization	~ More no. of temporary variables mean no optimization
~ Value of temporary variable can be quicly access using symbol table	~ It means more space and time complexity.

Triples

"This is the sandard form of representation of the Three Address Code (TAC). It contains three fields Opr, Arg 1, Arg 2. The address is refrenced to an operation, that which was previously performed to optimize space little more"

~ For the same ~~operat~~ TAC (pg: 25) the triple represtation would be:

Some address

@	Opr	Arg 1	Arg 2
(0)	*	n	n
(1)	*	p	(0)
(2)	/	(1)	100
(3)	+	p	(2)
(4)	=	(3)	

Demerits

~ Relocation of a triple to an address is costly as it requires updation of the whole table.

~ Rearragemet of code is hence difficult.

Indirect Triples (Not in syllabus)

- This is an enhancement over triple representation
- It uses an additional instruction array to list the pointers to the triples in the desired order
- Thus instead of position, pointers are used to store the results.
- It enables the optimizer to easily re position the sub-expression for producing the optimized code

∴ The Indirect Triple presentation would be:

pointers — triple address

#	@		@	OPr	Arg1	Arg2
1001	(0)	→	(0)	*	n	r
1002	(1)	→	(1)	*	p	(0)
1003	(2)	→	(2)	*	(1)	100
1004	(3)	→	(3)	/	p	(2)
1005	(4)	→	(4)	+	(3)	

Q.) Consider the following expression:

$p = a + b \times c / e \uparrow f + b \times c$

translate it to three address code and represent it in quadruples, triples and indirect triples.

Ans

Scanning the operator: Order acc. to precidance

$P = a + b \times c / e \uparrow f + b \times c$

$\uparrow$, $\underbrace{\times /}_{L \to R}$, $\underbrace{+ -}_{L \to R}$

$= a + [(b*c)/(e \uparrow f)] + (b \times c)$

$\therefore$ The three address code is:

$t_1 = a$

$t_2 = b \times c$

$t_3 = e \uparrow f$

$t_4 = t_2 / t_3$

$t_5 = \cancel{b \times c}\ t_1 + t_4$

$P = t_2 + t_5$

* Quadruples

@	Opr	Arg1	Arg2	Result
(0)	=	a		t_1
(1)	$\times$	b	c	t_2
(2)	$\uparrow$	e	f	t_3
(3)	/	t_2	t_3	t_4
(4)	+	t_1	t_4	t_5
(5)	+	t_2	t_5	P

* Triples

@	Opr	Arg1	Arg2
(0)	=	a	
(1)	$\times$	b	c
(2)	$\uparrow$	e	f

(3)	/	(1)	(2)
(4)	+	(0)	(3)
(5)	+	(1)	(4)

* - Indirect triples

#	@
1001	(0)
1002	(1)
1003	(2)
1004	(3)
1005	(4)
1006	(5)

@	Opr	Arg1	Arg2
(0)	=	a	
(1)	×	b	c
(2)	↑	e	f
(3)	/	(1)	(2)
(4)	+	(0)	(3)
(5)	+	(1)	(4)

☐ Types of three address assignments statement

(i) $y := x \text{ op } z$: Assignment Statement
op - is binary / logial operators eg (+, -, &, ∗)
x, z - operands

(ii) $y := \text{op } x$: Assignmt Instruction
op - is unarry (eg:- ++, +, -, --)
x - operand

(iii) $y := x$: Copy statement

(iv) goto L : Unconditional Jump
goto lable L

(v) If x relop y goto L : Conditional Jump
relop = relation operation (>, <, !=)

(vi) param x : Parameters passed to procedures
call p , n (actual & formal parameters)
Return statements : return y

(vii) Indexed Assignment : Assigned to the indices of data structures like arrays
$x := y[j]$

(viii) $x := \&y$
$x := *y$ } → Address & Pointer assignment - providing actual location.

Translation of assignment statements

- Assignment statements are used when there is a need to store certain values.

~~Consider the following production:~~

~~$S \rightarrow i$~~

- An expression with more than one operator like $a+b*c$, will translate into instructions with atmost one operator per instruction.
- An array refrence $A[i][j]$ will expand into a sequence of three address instructions that calculate an address for the refrence

~ The syntax directed definition builds up the three address code for assignment statements.

Consider the expression:

$S \longrightarrow id := E;$

$E \longrightarrow E_1 + E_2 \mid E_1 \mid (E_1) \mid id$

It's corresponding syntax directed definition would be:

$S \rightarrow id := E$

$E \rightarrow E_1 + E_2$

?

Doubt?

Control Flow

- The transation of statements such as if-else statements and while statements is tied to the translation of boolean expression.
- In programming language boolean expression are used ~~a cond~~ to:

(i) Alter the flow of control: Boolean expressions is implicit in a position reached in a program.

if (E) $\Rightarrow$ if the program reaches S
S E must be true.

(ii) Compute Logical Values: An boolean expression is always computed in if-else or while statements.

Boolean Expressions

- They are composed of boolean operators (&&, ||, !) applied to elements that are boolean variables or relational expression.

Let's consider a boolean expression generated by the following grammar:

$$B \rightarrow B \| B \mid B \&\& B \mid !B \mid (B) \mid E \text{ rel } E \mid \text{true} \mid \text{false}$$

- (rel : <, <=, >, >=, !=, ==)
- ||, && are left associative
- || has lower precedence, then &&, then !

* Methods of translating Boolean Expressions

1. Numerical Representation

To encode true & false numerically with 1 & 0 respectively.

eg:-

$t1 := \text{not } c$

$t2 := b \text{ not } t1$

$t3 = a \text{ or } t2$

~~B → !B G~~

~~C →~~

#2. Short Circuit Code

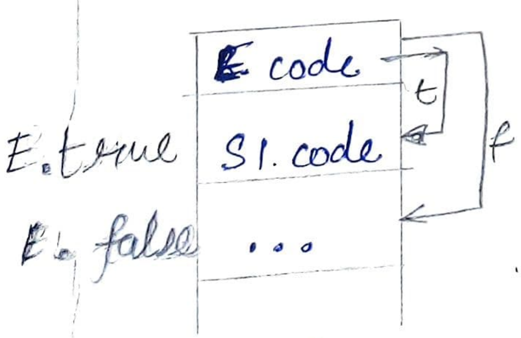
- In short-circuit (or jumping) code, the boolean operators &&, ||, and ! translate to jumps
- The operators themselves do not appear in the code; insead the value of a boolean expression is represendd by a position in the code. sequence

eg:- if (x < 100 || x > 200 && x != y) x = 0;

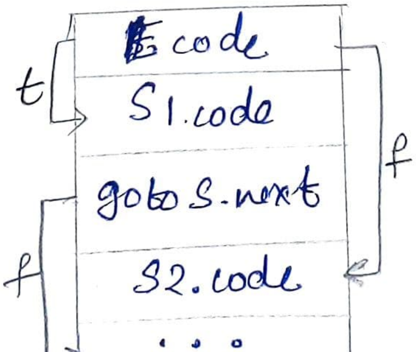
is translated into:

```
if x < 100 goto L2
if False x > 200 goto L1
if False x != y goto L1

L2 : x = 0
L1 : // some other code
```



if



if -else