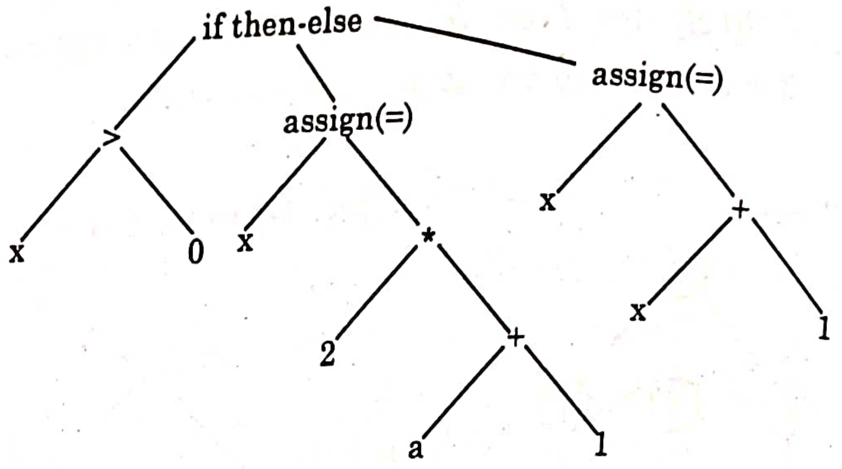
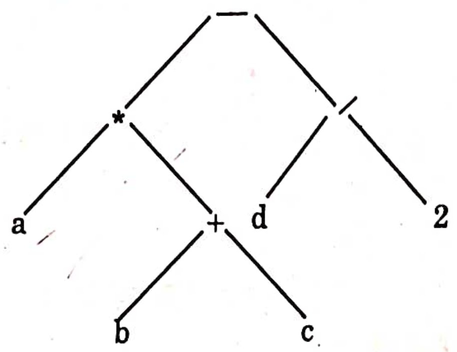
Ans.



Example 14. *Draw syntax tree for following arithmetic expression a * (b + c) – d/2. Also write given expression in postfix form.*

Ans.

Syntax Tree.



Postfix Notation

a b c + * d2/–

7.7 THREE ADDRESS CODE

This is most important way of representing intermediate code.

In three address code, atmost three addresses are used to represent any statement. Two addresses for operand & one for Result.

General form of Three-address Code representation is.

$$\underbrace{X}_{\substack{\downarrow \\ \textbf{Result}}} = \underbrace{Y}_{\substack{\downarrow \\ \textbf{Operand1}}} \ \textbf{op} \ \underbrace{Z}_{\substack{\downarrow \\ \textbf{Operand 2}}}$$

Here **op is an operator.**

Only single operation is allowed at a time at right side of expression.

E.g Expression a = b + c + d can be converted into following Three Address Code.

```
t1 = b + c
t2 = t1 +d
 a = t2
```

where t1 and t2 are temporary variables generated by compiler. Sometimes a statement contain less than three references, but it is still called a three address statement.

7.8 TYPES OF THREE ADDRESS CODE STATEMENTS

Following are the various types of three address statements.

1. **Assignment :** The three types of Assignment statements are

   x = y op z, op is a binary operator with y, z as operands

   x = op y, op is a unary operator

   x = y, value of y is assigned to x.

2. **Unconditional Jumps :** Unconditional Jump is of the form **goto L,** L being a label. Control flow to three address statement labeled at L.

3. **Conditional Jump :** Condition Jump is of the form

   if x relop y goto L

   Here relop can be < , > , < =, > =. If condition will be true, then it will execute three address statement at label L else statement following if statement will be executed.

4. **Array Statements :**

   x = y [i], value of ith location of array y is assigned to x

   x [i] = y, value of y is assigned to ith location of array x.

5. **Address & Pointer Assignments :** Languages like Pascal & C allow pointer assignments.

   x = & y, address of y assigned to x

   x = * y, content of location pointed to by y is assigned to x

   ***x = y**, It sets r-value of object pointed to by x to r-value of y.

6. **Procedure call/Return :** A call to the procedure P $(x_1, x_2, \ldots\ldots x_n)$ with the parameters $x_1, x_2, \ldots\ldots, x_n$ is written as

```
param x1
param x2
   ⋮
param xn
call p, n
```

Here **param** refers to parameter, & **call p, n** will call procedure p with n arguments.

7.9 REPRESENTATION OF THREE ADDRESS STATEMENTS

There are three Representations used for three address code which are given below :

1. Quadruples
2. Triples
3. Indirect Triples.

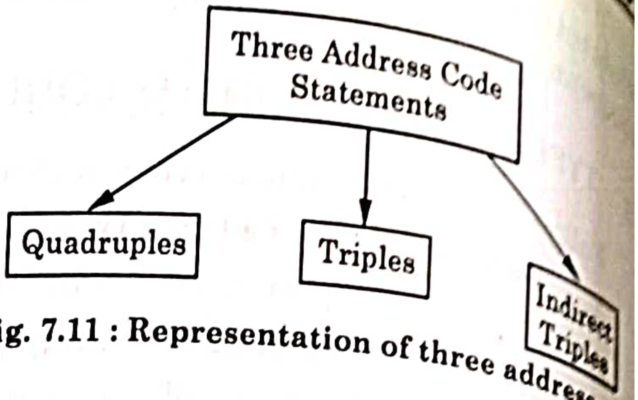


Fig. 7.11 : Representation of three address code

7.9.1 Quadruples

Quadruple is a structure which contains atmost four fields *i.e* Operator, Argument 1, Argument 2 and Result

Operator	Argument 1	Argument 2	Result

For a statement a = b + c, Quadruple Representation places + in the operator field, **a** in the Argument 1 field, **b** in Argument 2 and **c** in Result field.

For example : Consider the statement

a = b + c ∗ d

First convert this statement in Three Address code

∴ Three Address code will be

t1 = c*d

t2 = b + t1

a = t2.

After construction of Three Address code, it will be changed to Quadruple representation as follows :

Quadruple

	Operator	arg 1	arg 2	Result
(0)	*	c	d	t1
(1)	+	b	t1	t2
(2)	=	t2		a

The contents of fields arg1, arg2 and Result are basically pointers to symbol table entries for names represented by these entries.

7.9.2 Triples

This three address code representation contain three (3) fields *i.e.* one for operator and two for Arguments (*i.e.* Argument 1 and Argument 2)

Operator	Argument 1	Argument 2

In this representation, temporary variables are not used. Instead of temporary variables we use a number in parenthesis to represent pointer to that particular record of symbol table. For example consider statement

a = b + c * d

First of all, it will be converted to Three Address code.

t1 = c * d

∴ t2 = b + t1

a = t2

Triple for this Three-Address code will be:

Triple

	Operator	arg 1	arg 2
(0)	*	c	d
(1)	+	b	(0)
(2)	=	a	(1)

Here (0) represents a pointer which refer the result c*d, which can be used in further statements *i.e.* when c*d is added with b. This result will be saved at position pointed by (1). Pointer (1) will be used further when it is assigned to a.

7.9.3 Indirect Triples

In Indirect triples, all the pointers used in Triples are indexed such that one pointer will reference another pointer & that pointer will consist of the triple.

For the previous example, Indirect triple will be

Indirect Triples

	Statement
(0)	(11)
(1)	(12)
(2)	(13)

	Operator	arg1	arg2
(11)	*	c	d
(12)	+	b	(11)
(13)	=	a	(12)

In this, we only need to refer to pointers (0), (1), (2) which will further refer pointers (11), (12), (13) respectively & then pointers (11), (12), (13) point to triples, that is why this representation is called Indirect Triple Representation.

Example 15. *Write quadruples, triples and indirect triples for the expression*

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

Ans. First of all this statement will be converted into Three Address code as:

$t1 = a + b$

$t2 = -t1$

$t3 = c + d$

$t4 = t2 * t3$

$t5 = t1 + c$

$t6 = t4 - t5$

Quadruple :

	Operator	arg1	arg2	Result
(0)	+	a	b	t1
(1)	–	t1		t2
(2)	+	c	d	t3
(3)	*	t2	t3	t4.
(4)	+	t1	c	t5
(5)	–	t4	t5	t6

Triple

	Operator	arg1	arg2
(0)	+	a	b
(1)	–	(0)	
(2)	+	c	d
(3)	*	(1)	(2)
(4)	+	(0)	c
(5)	–	(3)	(4)

Indirect Triple

	Statement
(0)	(11)
(1)	(12)
(2)	(13)
(3)	(14)
(4)	(15)
(5)	(16)

	Operator	arg1	arg2
(11)	+	a	b
(12)	–	(11)	
(13)	+	c	d
(14)	*	(12)	(13)
(15)	+	(11)	c
(16)	–	(14)	(15)

Example 16. *Consider the Expression :*

$$a * -b + c * -d$$

Convert it into Quadruple, Triple and Indirect Triples.

Ans. Three address code for $a * -b + c * -d$ will be.

t1 = uminus (b)
t2 = a * t1
t3 = uminus (d)
t4 = c * t3
t5 = t2 + t4

Quadruples

	operator	arg1	arg2	Result
(0)	uminus	b		t1
(1)	*	a	t1	t2
(2)	uminus	d		t3
(3)	*	c	t3	t4
(4)	+	t2	t4	t5

Triple

	operator	arg1	arg2
(0)	uminus	b	
(1)	*	a	(0)
(2)	uminus	d	
(3)	*	c	(2)
(4)	+	(1)	(3)

Indirect Triple

	Statement
(0)	(11)
(1)	(12)
(2)	(13)
(3)	(14)
(4)	(15)

	Operator	arg1	arg2
(11)	uminus	b	
(12)	*	a	(11)
(13)	uminus	d	
(14)	*	c	(13)
(15)	+	(12)	(14)

Example 17. *Convert the expression into Triples*

x[i] = y and x = y[i].

Ans.

x[i] = y

	operator	arg1	arg2
(0)	[ ] =	x	i
(1)	=	y	(0)

x = y [i]

	operator	arg1	arg2
(0)	= [ ]	y	i
(1)	=	(0)	x

7.9.4 Comparison between Representations

1. **Space :** Quadruples fill the symbol table with lot of temporary variables. Quadruples & Indirect triples require about same amount of space. But Indirect triples can save some space compared with quadruples if same temporary value is used more then once. Because two or more entries in statement array can point to same line of operator -arg1-arg2 table.

2. **Accessing Temporaries :** In quadruples, location of each temporary can be directly accessed using Symbol Table. But In triple Representation, We have to scan the code to determine how many temporaries are active simultaneously or how many words must be allocated for temporaries.
3. **Degree of Indirection :** In Quadruple representation, symbol table has extra degree of indirection between computation of value & its use. If we move statement computing x, then statement using x will not change. But. In triples, moving a statement requires us to change all pointers to that statement. Therefore, Quadruples are used in optimizing compiler, in which statements are moved often. But Triples cannot be used in optimizing compiler.

7.9.5 Array Representation

As, Quadruples & triples causes some wastage of memory because some fields are not occupied. To prevent wastage of space, expression can be represented in single Array. E.g consider a statement

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

Its Three Address code will be

$t1 = -b$

$t2 = c*d$

$t3 = t1 + t2$

$a = t3$

Quadruple will be :

	Operator	arg1	arg2	Result
(0)	–	b		t1
(1)	*	c	d	t2
(2)	+	t1	t2	t3
(3)	=	t3		a

Since, there is wastage of space in the Quadruple, So it can be converted into Array Representation as.

–	b	t1	*	c	d	t2	+	t1	t2	t3	=	t3	a

Advantage : It saves memory space.

Disadvantage : We cannot recognize a word *i.e.* whether it is an operator or an operand. In Quadruple, it can be easily done as operators & operands are written in their corresponding fields.