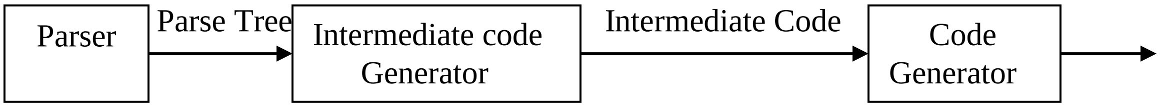
INTERMEDIATE CODE GENERATION

A compiler while translating a source program into a functionally equivalent object code representation may first generate an intermediate representation.

Advantages of generating intermediate representation

1. Ease of conversion from the source program to the intermediate code
2. Ease with which subsequent processing can be performed from the intermediate code



INTERMEDIATE LANGUAGES:

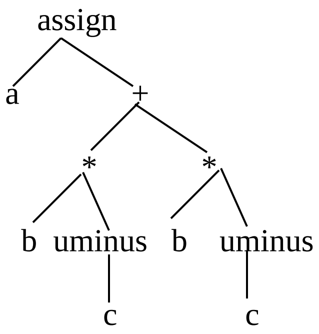
There are three kinds of Intermediate representation. They are,

1. Syntax Trees
2. Postfix Notation
3. Three address code

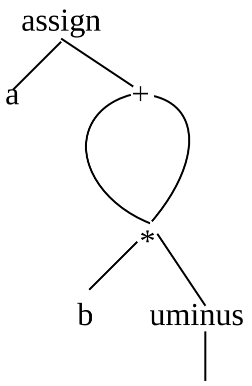
1. Syntax Tree:-

A syntax tree depicts the natural hierarchical structure of a source program. A DAG (Direct Acyclic Graph) gives the same information but in a more compact way because common sub expressions are identified.

A syntax tree and dag for the assignment statement a:= b* -c + b* -c



→ Syntax Tree



→ DAG

c

2. Postfix notation:-

Post fix notation is a linearized representation of a syntax tree. It is a list of nodes of the tree in which a node appears immediately after its children.

The postfix notation for the syntax tree is,

a b c uminus * b c uminus * + assign

3. Three Address Code:-

Three Address code is a sequence of statements of the general form

x := y op z

where x,y and z are names, constants or compiler generated temporaries.

op stands for any operator such as a fixed or floating point arithmetic operator or a logical operator on a Boolean valued data.

The Three Address Code for the source language expression like x+y*z is,

t_1:= y * z

t_2 := x + t_1

Where t_1 and t_2 are compiler generated temporary names

So, three address code is a linearized representation of a syntax tree or a dag in which explicit names correspond to the interior nodes of the graph.

Three Address Code Corresponding to the syntax tree and DAG is,

Code for Syntax Tree

t_1 := -c

t_2 := b * t_1

t_3 := -c

t_4 := b * t_3

t_5 := t_2 + t4

a := t_5

Code for DAG

t_1 := -c

t_2 := b * t_1

t_5 := t_2 + t_2

a := t_5

Types of Three Address Statements:-

1. Assignment statement of the form x := y op z
2. Assignment instructions of the form x := op z

   where op is a unary operation.
3. Copy statements of the form x := y

   where, the value of y is assigned to x.
4. The Unconditional Jump GOTO L
5. Conditional Jumps such as if x relop y goto l
6. param x and call p, n for procedure calls and return y.
7. Indexed assignments of the form x := y[i] and x[i] := y
8. Address and pointer assignments, x :=&y, x := *y and *x := y

Implementations of Three Address Statements:

It has three types,

1. Quadruples
2. Triples
3. Indirect Triples

Quadruples:-

A Quadruple is a record structure with four fields, which we call op, arg1, arg2, and result. The op field contains an internal code for the operator.

For Eg, the three address statements,

x := y op z is represented by

y in arg1

z in arg2

x in result.

The quadruples for the assignment a:=b* -c + b* -c are,

	op	arg1	arg2	result
(0)	uminus	c		t_1
(1)	*	b	t_1	t_2
(2)	uminus	c		t_3
(3)	*	b	t_3	t_4
(4)	+	t_2	t_4	t_5
(5)	:=	t_5		a

Triples:-

A triple is a record structure with three fields: op, arg1, arg2. This method is used to avoid entering temporary names into the symbol table.

Ex. Triple representation of a:= b * -c + b * -c

	op	arg1	arg2
(0)	uminus	c	
(1)	*	b	(0)
(2)	uminus	c	
(3)	*	b	(2)
(4)	+	(1)	(3)
(5)	assign	a	(4)

Indirect Triples:-

Listing pointers to triples rather than listing the triples themselves are called indirect triples.

Eg. Indirect Triple Representation of a := b * -c + b * -c

	statement
(0)	(10)
(1)	(11)
(2)	(12)
(3)	(13)
(4)	(14)
(5)	(15)

	op	arg1	arg2
(10)	uminus	c	
(11)	*	b	(10)
(12)	uminus	c	
(13)	*	b	(12)
(14)	+	(11)	(13)
(15)	assign	a	(14)

References

1. J. Tremblay, P.G. Sorenson,"The Theory and Practice of Compiler Writing “, McGRAW-HILL,1985.
2. W.M. Waite, L.R. Carter,"An Introduction to Compiler Construction",Harper Collins,New york,1993
3. A.W. Appel,"Modern Compiler Implementation in, CambridgeUniversity Press,1998
4. Internet Papers

5. Aho, R. Sethi, J.D. Ullman," Compilers- Principles, Techniques and Tools"Addison-Weseley, 2007