Three Address Code

Course Name: Compiler Design
Course Code: CSE331
Level:3, Term:3
Department of Computer Science and Engineering
Daffodil International University

Three address code in Compiler

Three address code is a type of intermediate code which is easy to generate and can be easily converted to machine code. It makes use of at most three addresses and one operator to represent an expression and the value computed at each instruction is stored in temporary variable generated by compiler. The compiler decides the order of operation given by three address code.

Converting the expression a / (b + c) into three address code:

$$t_1 = b + c$$
$$t_2 = a / t_1$$

Intermediate Code Generation

In the analysis-synthesis model of a compiler, the front end of a compiler translates a source program into an independent intermediate code, then the back end of the compiler uses this intermediate code to generate the target code (which can be understood by the machine).

EXERCISE: Three address code for the following expression: a + b * c – d / (b * c)

$t_1 = b * c$

$t_2 = a + t_1$

$t_3 = b * c$

$t_4 = d / t_3$

$t_5 = t_2 - t_4$

Implementation of Three Address Code

There are 3 representations of three address code:

1. Quadruple
2. Triples
3. Indirect Triples

For the following expression: a + b * c – d / (b * c)

Three Address Code:

$t_1 = b * c$

$t_2 = a + t_1$

$t_3 = b * c$

$t_4 = d / t_3$

$t_5 = t_2 – t_4$

Quadruples

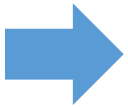
#	op	arg1	arg2	result
0	*	b	c	t_1
1	+	a	t_1	t_2
2	*	b	c	t_3
3	/	d	t_3	t_4
4	-	t_2	t_4	t_5

It is structure with consist of 4 fields namely op, arg1, arg2 and result. op denotes the operator and arg1 and arg2 denotes the two operands and result is used to store the result of the expression. Easy to rearrange code for global optimization. One can quickly access value of temporary variables using symbol table. Contain lot of temporaries. Temporary variable creation increases time and space complexity.

For the following expression: a + b * c – d / (b * c)

Quadruples

#	op	arg1	arg2	result
0	*	b	c	t_1
1	+	a	t_1	t_2
2	*	b	c	t_3
3	/	d	t_3	t_4
4	-	t_2	t_4	t_5



Triples

#	op	arg1	arg2
0	*	b	c
1	+	a	(0)
2	*	b	c
3	/	d	(2)
4	-	(1)	(3)

This representation doesn't make use of extra temporary variable to represent a single operation instead when a reference to another triple's value is needed, a pointer to that triple is used. So, it consist of only three fields namely op, arg1 and arg2. Temporaries are implicit and difficult to rearrange code.

For the following expression: a + b * c – d / (b * c)

Triples

#	op	arg1	arg2
0	*	b	c
1	+	a	(0)
2	*	b	c
3	/	d	(2)
4	-	(1)	(3)

Indirect Triples

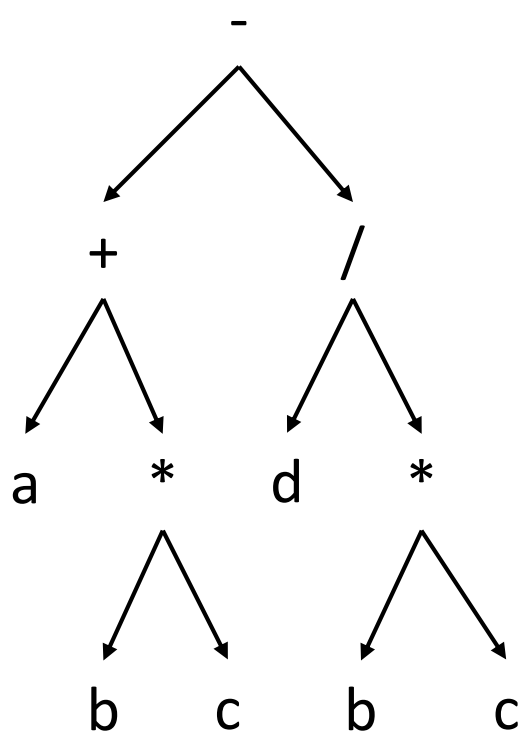
#	Statement
0	21
1	22
2	23
3	24
4	25

#	op	arg1	arg2
0	*	b	c
1	+	a	(0)
2	*	b	c
3	/	d	(2)
4	-	(1)	(3)

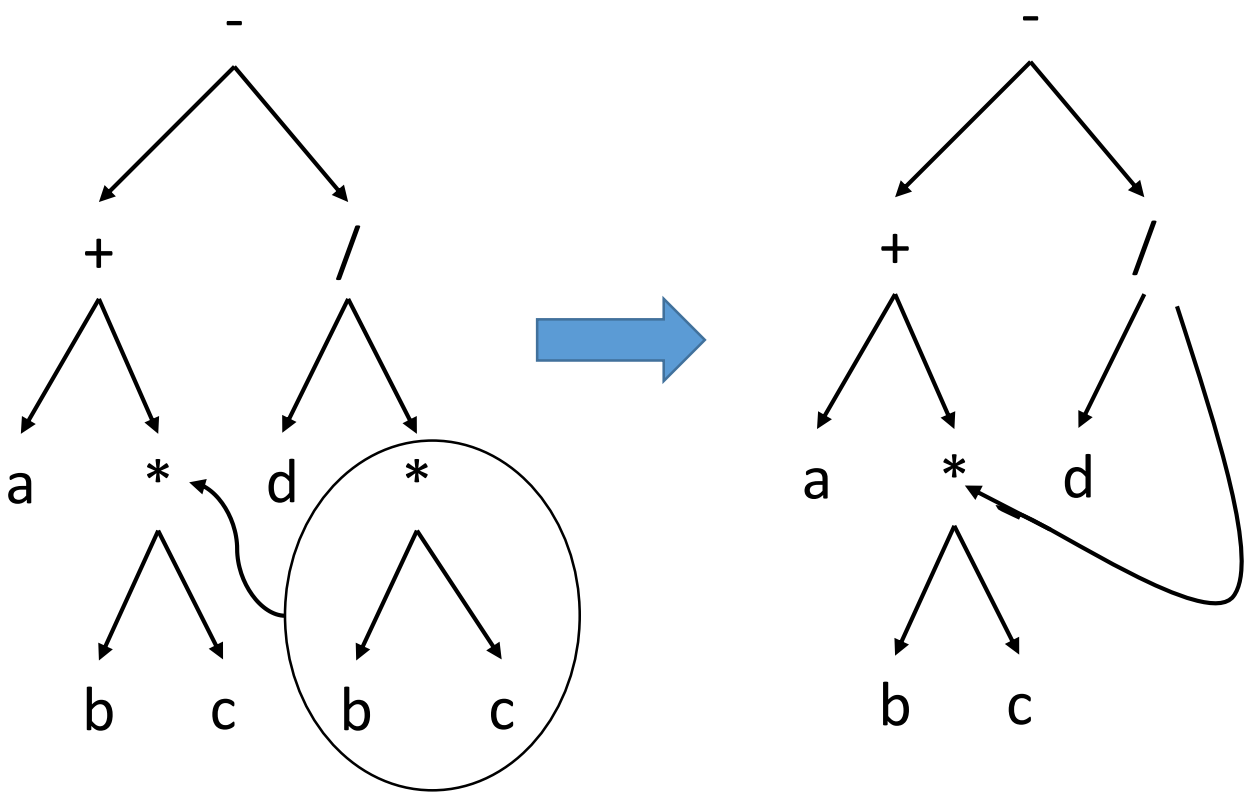
This representation makes use of pointer to the listing of all references to computations which is made separately and stored. Its similar in utility as compared to quadruple representation but requires less space than it. Temporaries are implicit and easier to rearrange code.

Directed Acyclic Graph (DAG)

For the following expression: a + b * c – d / (b * c)



Syntax directed tree



Directed Acyclic Graph (DAG)

THANK YOU