Three Address Code

- In the *analysis-synthesis* model of a compiler, the *front end* analyzes a source program and creates an *intermediate representation*, from which the *back end* generates target code.
- The two most important intermediate representations are:
    - *Trees*, including parse trees and (abstract) syntax trees.
    - Linear representations, especially "*three-address code*".
- In the three-address code, the given expression is broken down into several separate instructions.
- In this kind of instructions, at most three addresses can be used.
- There is at most one operator on the right side of an instruction.
- General Form:

  ```
  x = y op z
  ```

  where *x*, *y* and *z* are addresses and *op* is a binary operator.
- <u>Example</u>: A source-language instruction like *x+y*z* might be translated into the sequence of three-address instructions:

  $$t_1 = y * z$$
  $$t_2 = x + t_1$$

  where t_1 and t_2 are compiler-generated temporary names.
- An address can be one of the following:
    - *A name* (source-program names).
    - *A constant*.
    - *A compiler-generated temporary*.
- Common three-address instruction forms:
    - <u>Assignment Instructions:</u>
        - *x = y op z*, where *op* is a binary operator and *x*, *y*, *z* are addresses.
        - *x = op y*, where op is a unary operator.
    - <u>Copy Instructions</u>: *x = y*, where *x* is assigned the value of *y*.
    - <u>Unconditional Jump</u>: *goto L*. The three-address instruction with label L is the next to be executed.
    - <u>Conditional Jumps</u>:
        - ***if x goto L***: This instruction executes the instruction with label L next if x is true.
        - ***if False x goto L***: This instruction executes the instruction with label L next if x is false.

- ***if x relop y goto L***: This instruction executes the instruction with label L next if *x* stands in relation *relop* to *y*.

Otherwise, the following three-address instruction in sequence is executed next, as usual.

- Procedure Calls and Returns:
  - *param x* for parameters.
  - *call p, n* and *y = call p, n* for procedure calls.
  - *return y*, where *y*, representing a returned value, is optional.

- Index Copy Instructions:
  - ***x = y[i]***: This instruction sets *x* to the value in the location *i* memory units beyond location *y*.
  - ***x[i] = y***: This instruction sets the contents of the location *i* units beyond *x* to the value of *y*.

- Address and Pointer Assignments:
  - ***x = &y***
  - ***x = *y***
  - ****x = y***

Examples:

1. x = a + b * c + d
2. y = –(a * b) + (c+d)-(a+b+c+d)
3. If a<b then a=3 else b=2
4. i=2;
   ```
   j=1;
   for( ; i<5; i++){
       j++;
   }
   ```

Implementation of Three Address Code:

There are 3 representations of the three-address code namely:

1. Quadruples
2. Triples
3. Indirect Triples

Quadruples:

It is a structure with 4 fields namely *op*, *arg1*, *arg2* and *result*.

Example: a = b * -c + b * -e

t1 = uminus c
t2 = b * t1
t3 = uminus e
t4 = b * t3
t5 = t2 + t4
a = t5

#	Op	Arg1	Arg2	Result
(0)	uminus	c		t1
(1)	*	b	t1	t2
(2)	uminus	e		t3
(3)	*	b	t3	t4
(4)	+	t2	t4	t5
(5)	=	t5		A

Triples:

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 consists of only three fields namely op, arg1 and arg2.

#	Op	Arg1	Arg2
(0)	uminus	c	
(1)	*	b	(0)
(2)	uminus	e	
(3)	*	b	(2)
(4)	+	(1)	(3)
(5)	=	a	(4)

Indirect Triples:

This representation makes use of pointer to the listing of all references to computations which is made separately and stored.

#	Op	Arg1	Arg2
(14)	uminus	c	
(15)	*	b	(14)
(16)	uminus	e	
(17)	*	b	(16)
(18)	+	(15)	(17)
(19)	=	a	(18)

#	Statement
(0)	(15)
(1)	(14)
(2)	(16)
(3)	(17)
(4)	(18)
(5)	(19)