**Syntax-Directed Translation into Three-Address Code:**

|  |  |
| --- | --- |
| Production | Semantic Rules |
| S->id:=E  E->E1+E2  E->E1\*E2  E->-E1  E->( E1 )  E->id | S.code := E.code || gen(id.place ‘:=’ E.place)  E.place := newtemp;  E.code :=E1.code || E2.code || gen(E.place ‘:=’ E1.place ‘+’ E2.place)  E.place := newtemp;  E.code := E1.code || E2.code || gen(E.place ‘:=’ E1.place ‘\*’ E2.place)  E.place := newtemp;  E.code :=E1.code || gen(E.place ‘:=’ ‘uminus’ E1.place)  E.place := E1.place;  E.code := E1.code;  E.place := id.place;  E.code := “” |

**Implementation of Three-Address Statements:**

Eg. a := b \* -c + b \* -c

Quadruples:(easy to rearrange code for global optimization, lots of temporaries)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| # | Op | Arg1 | Arg2 | Res |
| (0) | uminus | c |  | t1 |
| (1) | \* | b | t1 | t2 |
| (2) | uminus | c |  | t3 |
| (3) | \* | b | t3 | t4 |
| (4) | + | t2 | t4 | t5 |
| (5) | := | t5 |  | a |

Triples: (temporaries are implicit, difficult to rearrange code)

|  |  |  |  |
| --- | --- | --- | --- |
| # | Op | Arg1 | Arg2 |
| (0) | uminus | c |  |
| (1) | \* | b | (0) |
| (2) | uminus | c |  |
| (3) | \* | b | (2) |
| (4) | + | (1) | (3) |
| (5) | := | a | (4) |

Indirect Triples: (temporaries are implicit & easier to rearrange code.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| # | Stmt |  | # | Op | Arg1 | Arg2 |
| (0) | (14) | 🡪 | (14) | uminus | c |  |
| (1) | (15) | 🡪 | (15) | \* | b | (14) |
| (2) | (16) | 🡪 | (16) | uminus | c |  |
| (3) | (17) | 🡪 | (17) | \* | b | (16) |
| (4) | (18) | 🡪 | (18) | + | (15) | (17) |
| (5) | (19) | 🡪 | (19) | := | a | (18) |

Program Triple container

**Regular Definitions**

The regular expression id is the pattern A regular definition gives names to certain regular expressions and uses those names in other regular expressions.

Here is a regular definition for the set of Pascal identifiers that is define as the set of strings of letter and digits beginning with a letters.

letter → A | B | . . . | Z | a | b | . . . | z  
digit  → 0 | 1 | 2 | . . . | 9  
   id  → letter (letter | digit)\*

for the Pascal identifier token and defines**letter** and **digit**.  
Where **letter** is a regular expression for the set of all upper-case and lower case letters in the alphabet and **digit** is the regular for the set of all decimal digits.

Relop → < | <= | > | >= | !=

Also you can draw the transition diagram (DFA) for id, letter, or digit.

SLR DFA (you can use the notation as in DFA for final state and normal state)

