EXPERIMENT – 8

CLASS: TE CMPN B
DATE: 28/02/ '22
NAME: Keegan Vaz
ROLL NO.: 28

AIM: TO IMPLEMENT INTERMEDIATE CODE GENERATION

THEORY:

- Role of an intermediate code generator

Ans: Intermediate code generator receives input from its predecessor phase, semantic analyzer, in the form of an annotated syntax tree. That syntax tree then can be converted into a linear representation, e.g., postfix notation. Intermediate code tends to be machine independent code. Therefore, code generators assume an unlimited number of memory storage (registers) to generate code. Intermediate code eliminates the need of a new full compiler for every unique machine by keeping the analysis portion the same for all the compilers. The second part of the compiler, synthesis, is changed according to the target machine.

- Intermediate representations – DAG and 3AC

Ans:

DAG -

Directed Acyclic Graph (DAG) is a tool that depicts the structure of basic blocks, helps to see the flow of values flowing among the basic blocks, and offers optimization too. DAG provides easy transformation on basic blocks.

DAG can be understood by:

- Leaf nodes represent identifiers, names or constants.
- Interior nodes represent operators.
- Interior nodes also represent the results of expressions or the identifiers/name where the values are to be stored or assigned.

3AC -

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 a temporary variable generated by the compiler. The compiler decides the order of operation given by three address codes.

General representation – a = b op c

Where a, b or c represents operands like names, constants or compiler generated temporaries and op represents the operator

- Different representations of 3AC – quadruples, triples, SSA

Ans: Quadruples - It is structured with 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.

Advantage –

- Easy to rearrange code for global optimization.
- One can quickly access the value of temporary variables using the symbol table.

Disadvantage –

- Contain a lot of temporaries.
- Temporary variable creation increases time and space complexity.

Triples - This representation doesn't make use of extra temporary variables 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.

Disadvantage –

- Temporaries are implicit and difficult to rearrange code.
- It is difficult to optimize because optimization involves moving intermediate code. When a triple is moved, any other triple referring to it must be updated also. With the help of a pointer one can directly access symbol table entry.

SSA - Static Single Assignment is a means of structuring the Intermediate representation such that every variable is allotted a value only once and every variable is defined before it's use. The prime use of SSA is it simplifies and improves the results of compiler optimisation algorithms, simultaneously by simplifying the variable properties. Some Algorithms improved by application of SSA –

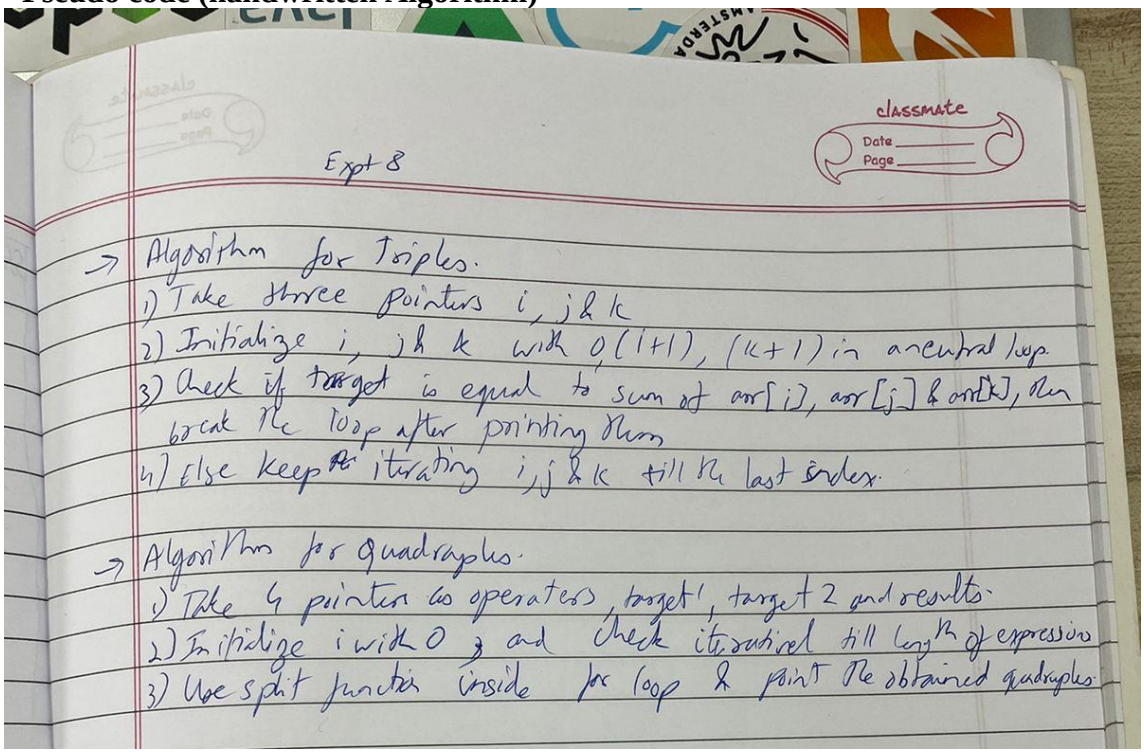
- Constant Propagation –
  Translation of calculations from runtime to compile time.
  E.g. – the instruction v = 2*7+13 is treated like v = 27
- Value Range Propagation –
  Finding the possible range of values a calculation could result in.
- Dead Code Elimination –
  Removing the code which is not accessible and will have no effect on results whatsoever.
- Strength Reduction –
  Replacing computationally expensive calculations by inexpensive ones.
- Register Allocation –
  Optimizing the use of registers for calculations.

Any code can be converted to SSA form by simply replacing the target variable of each code segment with a new variable and substituting each use of a variable with the new edition of

the variable reaching that point.

IMPLEMENTATION:

- Pseudo code (handwritten Algorithm)



- CODE (C / Java) – printed code

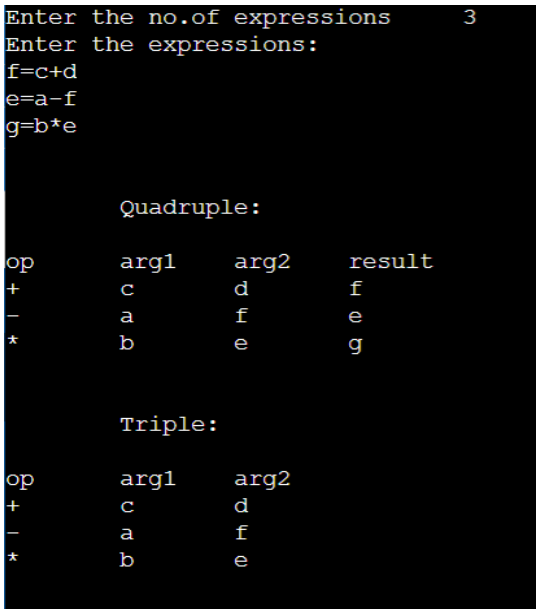
```
import java.util.*;

public class Main {

public static void qQuadruple(String expression[]) {
      System.out.println("op\targ1\targ2\tresult");
      for (int i = 0; i <expression.length;i++) {
         String[] expR = expression[i].split("");

System.out.println(expR[3]+"\t"+expR[2]+"\t"+expR[4]+"\t"+expR[0]);

      }
   }
   public static void tTriples(String expression[]) {
      System.out.println("op\targ1\targ2");
      for(int i=0;i<expression.length;i++){
         String[] expR = expression[i].split("");
         System.out.println(expR[3]+"\t"+expR[2]+"\t"+expR[4]);
      }
   }
   public static void main(String args[]) {
      Scanner sc = new Scanner(System.in);
```

```
        System.out.print("Enter the no.of expressions\t");
        int n = sc.nextInt();
        System.out.println("Enter the expressions: ");
        sc.nextLine();
        String exp[] = new String[n];
        for (int i = 0; i < n; i++) {
            exp[i] = sc.nextLine();
        }
        System.out.println("\n\n\tQuadruple: ");
        System.out.println();
        qQuadruple(exp);
        System.out.println("\n\n\tTriple: ");
        System.out.println();
        tTriples(exp);
    }
}
```

OUTPUT:

Enter the no. of statements
3
Enter the statements
f=c+d
e=a-f
g=b*e

Output will be in the form of Quadruples and Triples



```
Enter the no.of expressions     3
Enter the expressions:
f=c+d
e=a-f
g=b*e


        Quadruple:

op      arg1    arg2    result
+       c       d       f
-       a       f       e
*       b       e       g


        Triple:

op      arg1    arg2
+       c       d
-       a       f
*       b       e
```

CONCLUSION:

The Intermediate code is generated from the statements.