EXPERIMENT 09

CLASS: TE CMPN A ROLL NO. : 19
NAME: REBECCA DIAS PID: 182027

Aim:

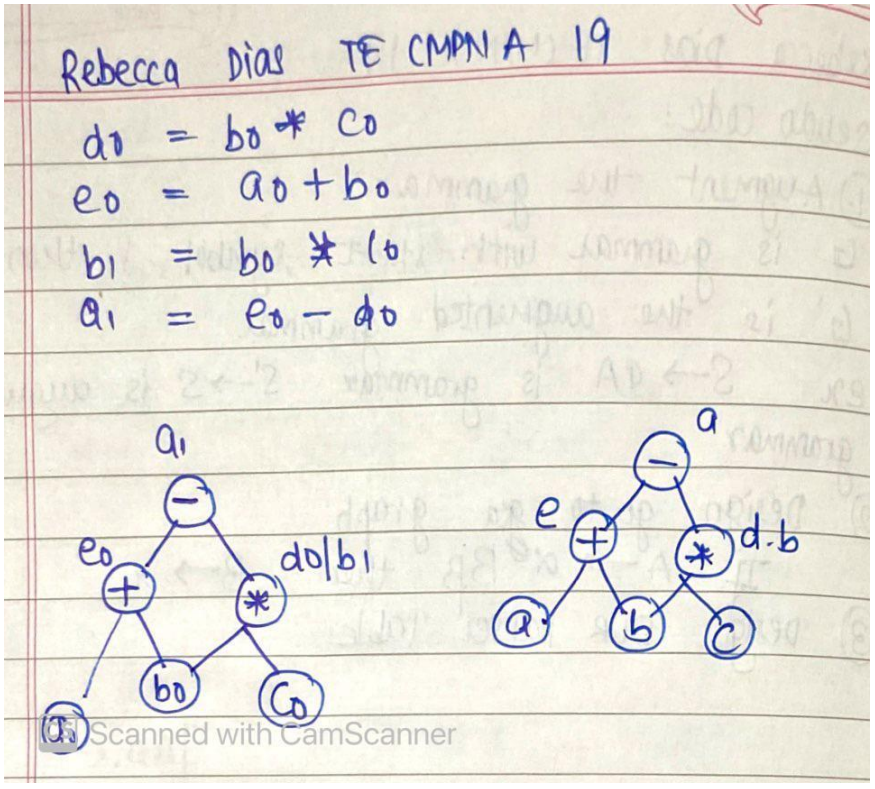
To implement intermediate code generation.

Theory:

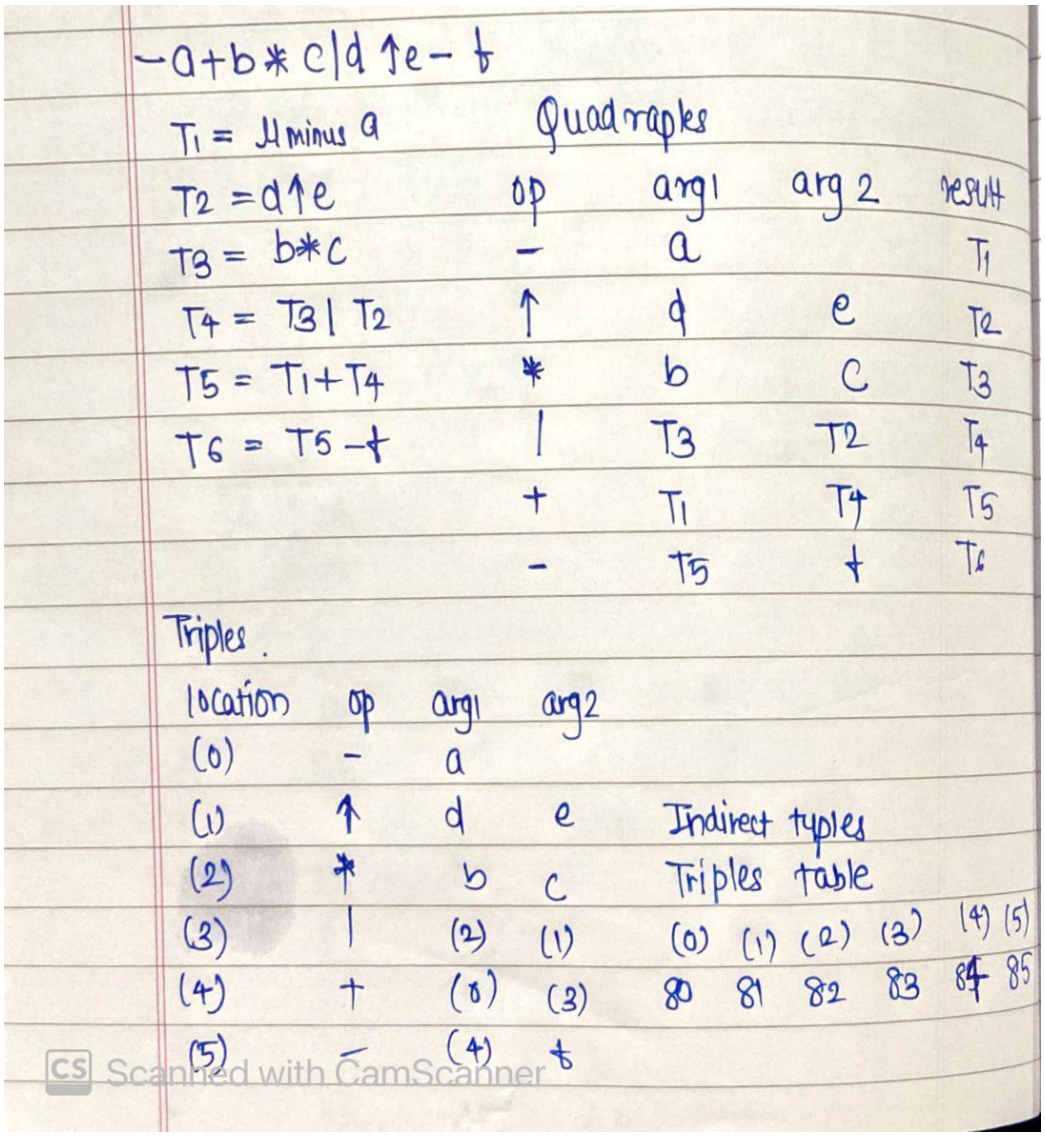
Role of an intermediate code generator

Intermediate code can be either language specific (e.g., Byte Code for Java) or language independent (three-address code). 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 generator assumes to have unlimited number of memory storage (register) to generate code.

Intermediate representations – DAG and 3AC



Different representations of 3AC – quadruples, triples, SSA



Implementation:

Pseudo Code

Rebecca Dias TECMPNA (19)

Pseudo code

1. Use 3 address code representation of intermediate code generator
2. Input total number of statements
3. Enter the statement in a way a=b+c or a=b
4. Convert the i/p to three address code
5. Represent the 3 address code in quadraples, triples and indirect triples
6. Quadraples
   - Split each instruction into four fields operator, arg1, arg2 & result
   - operator stores internal code of operator
   - argument 1 and 2 stores operand
   - if statement is the form a=b then result operator - = arg1-b arg2-null
   - for unconditional and jump statements use label
7. Triples
   - refrences the instruction
   - temporary variables are not used
   - split instruction to state ref, arg1, arg2 and operator
8. Indirect triples
   - enhance representation of triples by using additional instructional array
   - array element points the triple in desired order
   - use pointer instead of position
   - print quadraple, triple and indirect triple.

Code

```
def quadruples(e ,l):
    a = 0
    if l==3:
        a = 1
        arg2.append(' ')
    else:
        a = 2
    for i in range(a,l):
        if e[i].isalpha():
            if i == 2:
                arg1.append(e[i])
            else:
                arg2.append(e[i])
        elif e[i] in operator:
            op.append(e[i])
    result.append(e[0])
    location[e[0]] = j
    return
print('REBECCA DIAS TE CMPN A 19/182027')
n = int(input('Enter the number of expressions: '))
exp = []
print(f'Enter the {n} expressions in 3 address code:')
for i in range(n):
    exp.append(input())

operator = ['+','-','*','/','^','=']
op = []
arg1 = []
arg2 = []
result = []
j = 0
location = {}
for e in exp:
    quadruples(e,len(e))
    j = j + 1
print('\n__________Quadruples__________\nop\targ1\targ2\tresult\n')
for i in range(n):
    print(op[i],'\t',arg1[i],'\t',arg2[i],'\t',result[i],'\n')
print('\n___________Triples___________\nlocation\top\targ1\targ2\n')
for i in range(n):
    if arg1[i] in location and arg2[i] in location:
        print(i,'\t\t',op[i],'\t',location[arg1[i]],'\t',location[arg2[i]])
    elif arg1[i] in location:
        print(i,'\t\t',op[i],'\t',location[arg1[i]],'\t',arg2[i])
    elif arg2[i] in location:
        print(i,'\t\t',op[i],'\t',arg1[i],'\t',location[arg2[i]])
    else:
        print(i,'\t\t',op[i],'\t',arg1[i],'\t',arg2[i])
print('\n_________Indirect Triples_________\nlocation\top\targ1\targ2\n')
j = 1001
for i in range(n):
    if arg1[i] in location and arg2[i] in location:
```

```
        print(j,'\t\t',op[i],'\t',location[arg1[i]],'\t',location[arg2[i]])
    elif arg1[i] in location:
        print(j,'\t\t',op[i],'\t',location[arg1[i]],'\t',arg2[i])
    elif arg2[i] in location:
        print(j,'\t\t',op[i],'\t',arg1[i],'\t',location[arg2[i]])
    else:
        print(j,'\t\t',op[i],'\t',arg1[i],'\t',arg2[i])
    j = j + 1
```

Output

```
PS E:\SEM6\SPCC>  cd 'e:\SEM6\SPCC'; & 'C:\Users\Rebecc
2020.9.114305\pythonFiles\lib\python\debugpy\launcher'
REBECCA DIAS TE CMPN A 19/182027
Enter the number of expressions: 4
Enter the 4 expressions in 3 address code:
a=b
f=c+d
e=a-f
g=b*c

_________Quadruples_________
op      arg1    arg2    result

=       b               a

+       c       d       f

-       a       f       e

*       b       c       g
```

```
_____________Triples_____________
location        op      arg1    arg2

0               =       b
1               +       c       d
2               -       0       1
3               *       b       c

_________Indirect Triples________
location        op      arg1    arg2

1001            =       b
1002            +       c       d
1003            -       0       1
1004            *       b       c
```

Conclusion:

The intermediate code is generated from the statements. In this experiment we studied the role of intermediate code generator and computed the different representations of three address code for the given expressions. Thus we successfully implemented the experiment.