Experiment 06 : Code Generator

Learning Objective: Student should be able to apply code generator for target machine architecture.

Tools: Jdk1.8,Turbo C/C++, Python, Notepad++

Theory:

Code Generator: It takes an input from an intermediate representation of the source program and produces as output an equivalent target program.

Code-Generation Algorithm:

Code-Generation algorithm takes as a input a sequence of three address statements constituting a basic block. For each three address statement of the form X = Y op Z we perform the following actions:

1. Invoke a function getreg to determine the location L where the result of the computation Y op Z should be stored. L will be a register or memory location.
2. Consult the address descriptor for y to determine y', the current location of y. Prefer the register for y' if the value of y is currently both in memory and a register. If the value of y is not already in L, generate the instruction MOV y', L to place a copy of y in L.
3. Generate the instruction OP z', L where z' is a current location of z. Prefer register entry of z. update the address descriptor of x t indicate that x is in location L. If L is a register, update its descriptors to indicate that it contains the value of x, and remove x from all other register descriptors.
4. If the current values of y and/or z have no next uses, are not live on exit from the block, and are in registers, alter the register descriptor to indicate that, after execution of x = y op z, those registers no longer will contain y and/or z, respectively.

The Function getreg

The function getreg returns the location to hold the value of x for the assignment x = y op z.

1. If the name y is in a register that holds the value of no other names and y is not live and has no next use after execution of x = y op z, then returns the register of y is no longer in L.
2. failing 1, return an empty register for L if there is one.

3. failing 2, if x has a next use in a block, op is an operator, such as indexing, that requires a register, find an occupied register R. Store the value of R into a memory location (by MOV R,M) if it is not already in the proper memory location M, update the address descriptor for M, and return R. If R holds the value of several variables, a MOV instruction must be generated for each variable that needs to be stored. A suitable occupied register might be one whse value is also in memory.
4. If x is not used in the block, or no suitable occupied register can be found, select the memory location of x as L.

INPUT:

A=B+C

B=A-D

C=B+C

D=B

OUTPUT:

Mov R1, B

Mov R2, C

Add R1, R2

Mov R2, D

Sub R1, R2

Mov R2, C

Add R1, R2

Mov D, R1

Design:

Result and Discussion:

Learning Outcomes: The student should have the ability to

LO1 **Define** the role of Code Generator in Compiler design.
LO2: ***Apply*** the code generator algorithm to generate the machine code.
LO3: ***Generate*** target code for the optimized code, considering the target machines.

Course Outcomes: Upon completion of the course students will be able to evaluate the synthesis phase to produce object code optimized in terms of high execution speed and less memory usage.

Conclusion:

For Faculty Use

Correction Parameters	Formative Assessment [40%]	Timely completion of Practical [40%]	Attendance / Learning Attitude [20%]	
Marks Obtained				

CODE:

```
from tabulate import tabulate
def threeaddr(s):
  l=s.split(" ")
  l=l[2:]

  op=['+','-','*','/','^']
  arg1=[]
  arg2=[]
  res=[]
  oper=[]
  n=(len(l))//2
  if(l[n] not in op):
    while(l[n] not in op):
      n=n-1
  p1=l[:n]
  p2=l[n+1:]
  ind=1
  '''
  oper.append('OPERATOR:')
  arg1.append('ARGUMENT 1:')
  arg2.append('ARGUMENT2:')
  res.append('RESULT:')
  '''
  if(len(l)==3):
    oper.append(l[n])
    arg1.append(l[0])
    arg2.append(l[2])
    res.append("t"+str(ind))
    oper.append("=")
    arg1.append(s[0])
    arg2.append("t"+str(ind))
    res.append("t"+str(ind+1))
    ans=[]
    z1=zip(oper,arg1,arg2,res)
    for a1,a2,a3,a4 in list(z1):
      aq=[]
      aq.append(a1)
      aq.append(a2)
      aq.append(a3)
      aq.append(a4)
      ans.append(aq)
    print("QUADRAPLE TABLE:")
```

```
  print(tabulate(ans, headers=["OPERATORS","ARG 1","ARG 2","RESULT"],tablefmt='orgtbl'))
 else:
  m=0
  for i in p1:
    if(i[0] in op and len(i)>1):
        oper.append("unary"+i[0])
        arg1.append(i[1])
        arg2.append("nill")
        res.append("t"+str(ind))
        ind=ind+1
    if(i in op and len(i)==1):

        oper.append(i)
        arg1.append(p1[m-1])
        #print(p1.index(i)+1)
        arg2.append(p1[m+1])
        res.append("t"+str(ind))
        my="t"+str(ind)
        ind=ind+1
    m=m+1
  j=0
  for i in p2:
     if(i[0] in op and len(i)>1):

        oper.append("unary"+i[0])
        arg1.append(i[1])
        arg2.append("nill")
        res.append("t"+str(ind))
        ind=ind+1
     if(i in op and len(i)==1):

        oper.append(i)
        arg1.append(p2[j-1])
        arg2.append(p2[j+1])
        res.append("t"+str(ind))
        you="t"+str(ind)
        ind=ind+1
     j=j+1
  oper.append(l[n])
  arg1.append(my)
  arg2.append(you)
  res.append("t"+str(ind))
  oper.append("=")
  arg1.append(s[0])
  arg2.append("t"+str(ind))
```

```
    res.append("t"+str(ind+1))
    z=zip(oper,arg1,arg2,res)
    ans=[]
    for a1,a2,a3,a4 in list(z):
      aq=[]
      aq.append(a1)
      aq.append(a2)
      aq.append(a3)
      aq.append(a4)
      ans.append(aq)
    print("QUADRAPLE TABLE:")
    print(tabulate(ans, headers=["OPERATORS","ARG 1","ARG 2","RESULT"],tablefmt='orgtbl'))

    # print(a1,a2,a3,a4)
s=input("Enter code:")
threeaddr(s)
```

OUTPUT:

```
Enter code:a = b * -c + b * -c
QUADRAPLE TABLE:
| OPERATORS   | ARG 1   | ARG 2   | RESULT   |
|-------------+---------+---------+----------|
| *           | b       | -c      | t1       |
| unary-      | c       | nill    | t2       |
| *           | b       | -c      | t3       |
| unary-      | c       | nill    | t4       |
| +           | t1      | t3      | t5       |
| =           | a       | t5      | t6       |
```

```
Enter code:a = b + c
QUADRAPLE TABLE:
| OPERATORS   | ARG 1   | ARG 2   | RESULT   |
|-------------+---------+---------+----------|
| +           | b       | c       | t1       |
| =           | a       | t1      | t2       |
```