Om Shete
THADOMAL SHAHANI TSEC ENGINEERING COLLEGE
TE C32
2103163

Experiment No : 10

Aim : WAA to implement code generation.

Theory :

- Code generation is need to produce the target code for three address statements. It uses register to store the operands of the additional statements.
- Consider 3 additional statements $x = y + z$, It can have following sequence of codes.

```
MOV  X , R0
ADD  Y , R0
```

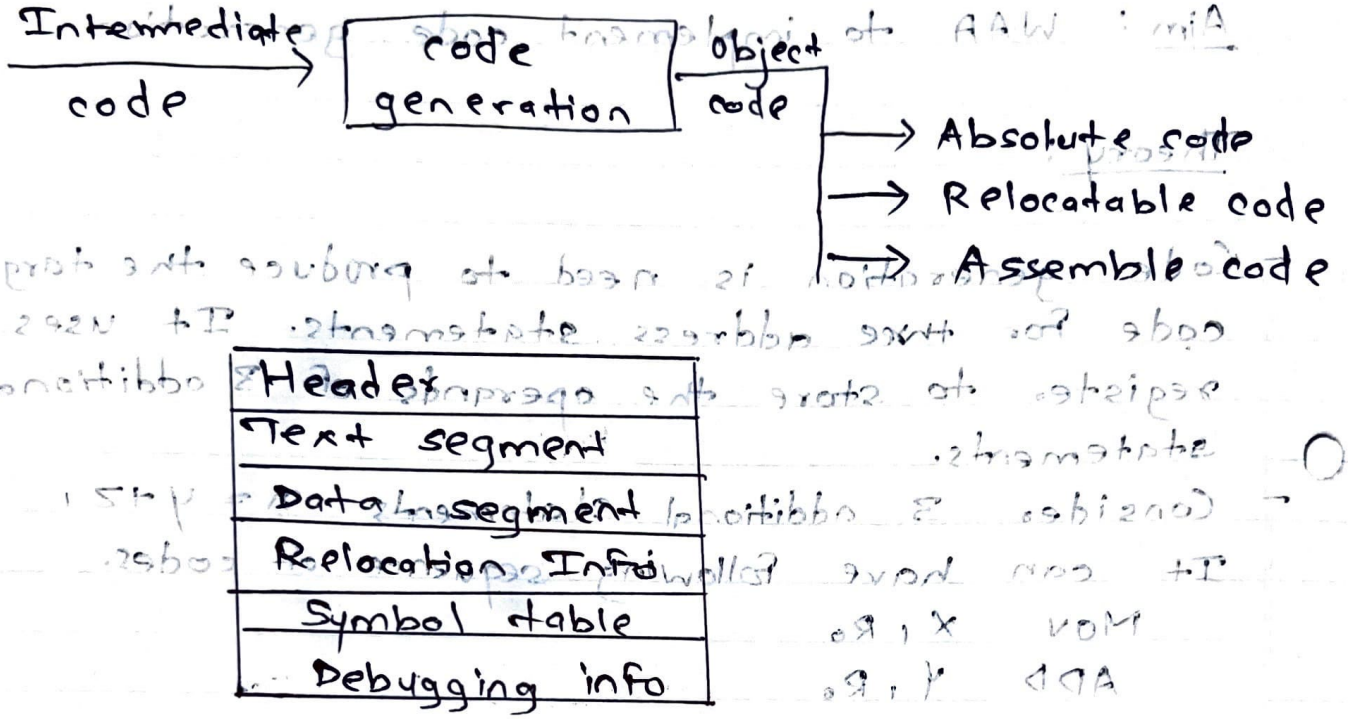
- Register description contains the track of what is in currently each register.
- An address descriptor is need to store the location where current value of name can be found at run time.

- **Code Generation Algorithm :**

The algorithm takes a sequence of three address statements as input. For each three address statement of the form a : b op c perform the various actions.

These actions are as follows

1) Invoke a function getreg to find out the location where the result of computation b op c should be stored.



Statement	Code generated	Register Description	Address descriptor
t = a - b	Mov a, R0 SUB b, R0	R0 contains t	t in R0
u = a - c	Mov a, R1 SUB c, R1	R0 contains t R1 contains u	t in R0 u in R1
v = t + u	~~Mov t, R2~~ ADD R1, R0	R0 contains v R1 contains u	u in R1 v in R1
d = v + u	ADD R1, R0 Mov R0, d	R0 contained	d in R0 d in R0 d mem

THADOMAL SHAHANI
TSEC
ENGINEERING COLLEGE

2) Consult the address description for y to determine y'. If the value of y currently in memory and register both then prefer the register y'. If the value of y currently in memory and register both then prefer the register y'. If the value of y is not already in L there generates the information MOV y', L to place a copy of y in L.

3) Generate the information OP z', L where z' is used to show the current location of z. If z is in both then prefer a register to a memory location. Update the address descriptor to x to indicate that x is in location L. If x is in L then update its descriptor and remove x from all other descriptor.

4) If the current value of y have no next uses or not have an exit from block or in register then allows the register descriptor to indicate that after execution of x := y op z those register with no longer contains y on x.

- Conclusion: In this, I learned and implemented code generation and its algorithm.

QA
28/03/2024.

Code:

```
op1, op2, op3, op4 = "", "", "", ""
print("Enter operation and operands (e.g., + o1 o2 o3): ")

while True:
  line = input().split()
  if not line:
    break
  op1, op2, op3 = line[0], line[1], line[2]
  if len(line) > 3:
    op4 = line[3]
  else:
    op4 = "Result"
  if op1 == "+":
   print(f"MOV AX, {op2}")
   print(f"MOV BX, {op3}")
   print("ADD AX, BX")
   print(f"MOV {op4}, AX")
  elif op1 == "-":
   print(f"MOV AX, {op2}")
   print(f"MOV BX, {op3}")
   print("SUB AX, BX")
   print(f"MOV {op4}, AX")
  elif op1 == "*":
   print(f"MOV AX, {op2}")
   print(f"MOV BX, {op3}")
   print("MUL AX, BX")
   print(f"MOV {op4}, AX")
  elif op1 == "/":
   print(f"MOV AX, {op2}")
   print(f"MOV BX, {op3}")
   print("DIV AX, BX")
   print(f"MOV {op4}, AX")
  elif op1 == "=":
    print(f"MOV {op2}, {op3}")
  else:
    print("Invalid operation")
print("\nCode generation successful")
```

Output:

```
Enter operation and operands (e.g., + o1 o2 o3):
- a b t
MOV AX, a
MOV BX, b
SUB AX, BX
MOV t, AX
- a c u
MOV AX, a
MOV BX, c
SUB AX, BX
MOV u, AX
+ t u v
MOV AX, t
MOV BX, u
ADD AX, BX
MOV v, AX
+ v u d
MOV AX, v
MOV BX, u
ADD AX, BX
MOV d, AX
* v u k
MOV AX, v
MOV BX, u
MUL AX, BX
MOV k, AX


Code generation successful
```