Unit -5

Q1. Issues in code generation

The following issues arise during the code generation phase -

- 1. Input to code generator
- 2. Target program
- 3. Memory allocati management
- 4. Instruction Selection
- 5. Register allocation
- 6. Evaluation order

1. Input to code generator -

The input to code generation consists of intermediate representation of the source program produced by front end, together with information in the symbol table.

Intermediate representation can be:

- i) Linear representation (eg. postfix notation)
- ii) Graphical representation (eg. syntax trees and dags)
- iii) Three address representation (eg. Quadruples)
- iv) Virtual machine representation (eg. stack machine code)

Prior to code generation, the front end must be scanned, parsed and translated into intermediate representation along with necessary type checking.

Therefore, input to the code generator is assumed to be emor free.



2. Target code -

The output of the code generator is is the target program

The output may be:

- i) Absolute machine language: It can be placed in a fixed memory location and can be executed immediately.
- ii) Relocatable machine language: It allows subprograms
 to be compiled separately.
- iii) Assembly language: Code generation is made easier
- 3. Memory management -
 - Names in the source program are conv mapped to
 the addresses of data objects in run-time memory
 by the front end and code generator
 - It makes use of symbol table
 - A name in the three address statement refers to a symbol table entry for the name.

4. Instruction selection -

The instructions of the target machine should be complete and uniform.

Instruction speeds and machine idioms are important factors in the efficiency of the target program.

The quality of the generated target code may be determined by speed and size.

eg. x=y+z => three address statement form

code sequence generated =>

MOV y, Ro ADD z, Ro

MOV Ro, 2

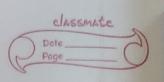
Unfortunately this kind of statement by statement code generation often produces poor code

eg. a = b + c d = a + e

MOV b, Ro
ADD C, Ro
MOV Ro, a

MOV a, Ro-ADD e, Ro

Mov Ro, d



I target machine with a rich instruction set can provide several ways to implement a given operation.

eg. If a target machine how "increment" instruction INC, then for

a = a + 1

instead of

MOV a, Ro
ADD #1, Ro
MOV Ro, a

we can implement it more efficiently by

INC a

5. Register allocation -

Instructions involving register operands one Shorter and faster than instructions involving operands in memory.

Register allocation — The set of variables in the program that will reside in the registers is chosen.

Register assignment — The specific registers that the variables will reside in is chosen.

6. Evaluation order -

At last, the code generator decides which order executed the instructions will be evaluated in

The order in which computations are performed can affect the efficiency of the target program.

cg. a+b- (c+d) *e

Three address coole	Code
t1= a+b	MOV a, Ro
t2= c+d	ADD b, Ro
t3= t2*e	Mov Ro, t1
t4= t1-t3	Mov c, R1
	ADD d, R1
The same that the same to	Mov e, Ro
ais in a memory location	MUL R1, Ro
	Mov +1, R1
	SUB +3, R1
	MOV R1, +4

Re ordered 3 address code Code

t2= c+ d	110V,C, Ko
t3 = t2 * e	ADD d, Ro
$+1 = \alpha + b$	Mov e, R1
t4=+1-+3	MUL Ro, R1
each registers	Mov a, Ro
	ADD b, Ro
Story surrent location relice	XXXX SUB R1, Ro
current value of a name or	MOV Ro, ty

Reduces the number of final code by 2, thus saves cost

Q2. A Simple Code Generator

I simple code generator generates the target code for a sequence of three address code statements and effectively uses registers to store operands of the statement.

eg. a = b + c

This can have the following sequence of codes -

ADD Rj, R; if Rj contains c and R; contains

Cost = 1.

OR.

ADD C, R; if c is in a memory location

Cost = 2

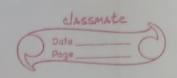
OR

MOV c, R; moving c from memory to register
ADD R; R;

Cost = 3

Register descriptor -> tracks what is currently in each registers

Address descriptor - stores current location where current value of a name can be found.



Generating code for assignment statement

Code generation algorithm -

for each 3 address statement of the form x=yopz, perform the following actions-

- 1. Invoke function getteg to find the Location L where the result of y op 2 should be stored.
- 2. Consult address descriptor of y to find y', (the current location of y), then Mov y', L
- 3. Generate instruction of z', 1, where z' is the current location of z
- 4. Update the address descriptor of x to indicate that x is in Location L

Generating code for Assignment Statements -

the assignment d = (a-b) + (a-c) + (a-c)can be translated into the following three address code \rightarrow

$$V = + + U$$

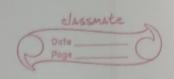
Code sequence	for the	example	is	-
---------------	---------	---------	----	---

			1
Statements	Code gen	Register descriptor	Address descriptor
t=a-b	MOV a, Ro SUB b, Ro	Ro contains t	t in Ro
<i>U</i> =α-c	Mova, R, SUBC, R,	Ro contains t R, contains u	tin Ro u in R,
v= ++u	ADD R, , Ro	Ro contains v R, contains u	vin Ro uin R,
d= V+U	ADD R, Ro Mov Ro, d	Ro contains d	d in Ro and memory

Generating code for indexed assignment -

Statements /	code	(gen/	Cost
b=a67	Mov	a[ii], b	//
a[i]=b	MOV	b/ali]

Statements	code gen	cost
a = b[i]	MOV b[Ri], R	2
a[i] = b	MOV b, a[R;]	3



Generating code for Pointer assignments -

Statements	Code gen	cost	
a=*p	MOV *Rp, a	2	
*p=a	MOV a, *Rp	2	

Generating code for conditional statements -

target code ⇒

MOV y, Ro
ADD z, Ro
MOV Ro, x