Assignment No.13

AIM:

Generation of target code for given 3-address code.

THEORY:

Issues in the design of code generator

1. Input to the code generator - The input to the code generation consists of the intermediate representation of the source program produced by front end , together with information in the symbol table to determine run-time addresses of the data objects denoted by the names in the intermediate representation.

2. Target programs - The output of the code generator is the target program. The output may be :
    a. Absolute machine language - It can be placed in a fixed memory location and can be executed immediately.
    b. Relocatable machine language - It allows subprograms to be compiled separately.
    c. Assembly language - Code generation is made easier.

3. Memory management - Names in the source program are mapped to addresses of data objects in run-time memory by the front end and code generator. It makes use of symbol table, that is, a name in a three-address statement refers to a symbol-table entry for the name. Labels in three-address statements have to be converted to addresses of instructions.

4. Instruction selection- The instructions of target machine should be complete and uniform. Instruction speeds and machine idioms are important factors when efficiency of target program is considered. The quality of the generated code is determined by its speed and size.

5. Register allocation- Instructions involving register operands are shorter and faster than those involving operands in memory. The use of registers is subdivided into two subproblems :
    a. Register allocation – the set of variables that will reside in registers at a point in the program is selected.
    b. Register assignment – the specific register that a variable will reside in is picked.
6. Choice of evaluation order - The order in which the computations are performed can affect the efficiency of the target code. Some computation orders require fewer registers to hold intermediate results than others.

Code generation algorithm

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

A **register descriptor** is used to keep track of what is currently in each registers.

The register descriptors show that initially all the registers are empty.

An **address descriptor** stores the location where the current value of the name can be found at run time.

The algorithm takes as input a sequence of three-address statements constituting a basic block.

For each three-address statement of the form x : = y op z, perform the following actions:

For each statement x:= y op z
1. Set location L = getreg(y, z)
2. If y is not in L, then generate MOV y',L
   y' denotes one of the locations where the value of y is available
   (choose register if possible)
3. Generate OP z',L
   z' is one of the locations of z;
   Update register/address descriptor of x to include L
4. If y and/or z have no next use and are stored in register, update register descriptors to remove y and/or z

Finally, we store all names that are live on exit and not in their memory locations

To compute L=getreg(y, z)

1. If y is stored in a register R, R only holds the value y, and y has no next use, then return R;
   - ✓ Update address descriptor: value y no longer in R
2. Else, return a new empty register if available
3. Else, find an occupied register R;
   - ✓ Store contents in memory (if not there) by generating MOV R,M
   - ✓ Return register R
4. Return the memory location of x if no proper register is found or x is not used in the block

The assignment d : = (a-b) + (a-c) + (a-c) might be translated into the following three address code sequence:

t : = a – b
u : = a – c
v : = t + u
d : = v + u

with d live at the end.

Statements	Code Generated	Register descriptor Register empty	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	ADD R1, R0	R0 contains v R1 contains u	u in R1 v in R0
d : = v + u	ADD R1, R0 MOV R0, d	R0 contains d	d in R0 d in R0 and memory

INPUT:

Input is:

i1+i2*60.0

OUTPUT:

Output is:

```
MOVF i2, R2
MULF   #60.0, R2
MOVF i1, R1
ADDF R2, R1
```

CONCLUSION:

- A code generator generates target code for a sequence of three- address statements and effectively uses registers to store operands of the statements.
- The target code is generated for given three address code.

REFERENCES:

- Compilers - Principles, Techniques and Tools - A.V. Aho, R. Shethi and J. D. Ullman (Pearson Education)