Parul® University

Introduction to machine code generation and optimization-

Nisha Panchal, Asst. Professor
Computer Science & Engineering

Parul® University

CHAPTER-8

Introduction to machine code generation and optimization

Code Generation

Code generation is the last phase of compilation. We all know that compiler converts High-level language into Low-level language that generated lower-level object code which has the exact meaning of source code and has efficiently used memory and CPU.

We will see the steps to convert intermediate code into the target object code or assembly code.

For example if what to do a summation of 2 variables like x= y+z

Add R0, R1; - where R0 contains y and R1, contains z.

Descriptive:

Register descriptive keeps track of each register value and initially all registers are empty.

Address descriptive stores location where you will find the current value of name at run time.

Code generation algorithm

Input: a sequence of three address code

Every three address code is a form of a= b op c. where op is the operand.

Here are the steps of the algorithm:

Invoke the "getreg" function to get the location L where we will store the value of b op c.

1. Find out the current value of b by consulting Address descriptive of b. If b is not in the register L then write the following instruction to copy the value of b in L.

MOV b', L - here b' denoted same value as b.

2. Find out the current value of c by using step 2 and generate the below instruction.

OP c', L

3. Now L contains the value of b OP c, which is intended to be assigned to **a**. So, if L is a register, update its descriptor to indicate that it contains the value of **a**. Update the descriptor of **a** to indicate that it is stored at location **L**.

4. If b and c have no further use, they can be given back to the system.

Example

1. t:= a-b
2. u:= a-c
3. v:= t +u
4. d:= v+u

Statement	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 R1
d:= v + u	ADD R1, R0 MOV R0, d	R0 contains d	d in R0 d in R0 and memory

Directed Acyclic Graph

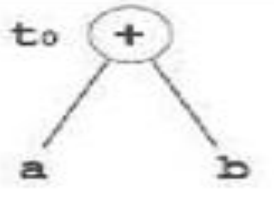
Directed Acyclic Graph is a tool that shows the structure of basic blocks and will help to understand the flow of the value among the basic blocks. It also provides optimization. In the Directed Acyclic graph, the leaf node represents the identifier, variable name, function name, and constants. The intermediate node represents the operators. The intermediate node also holds the value until that expression.

Parul® University

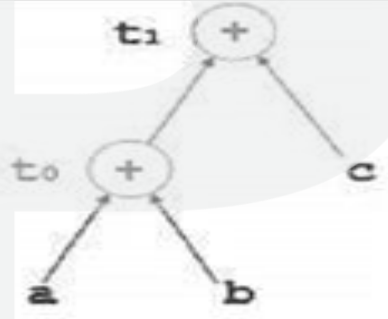
DIGITAL LEARNING CONTENT

Directed Acyclic Graph

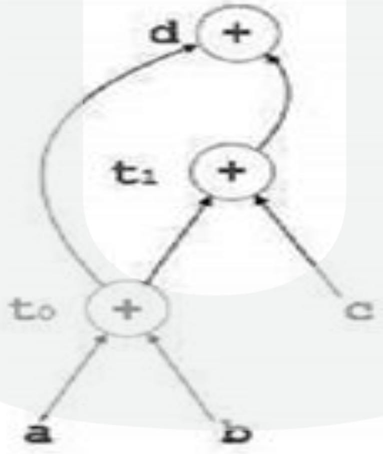
```
t0 = a + b
t1 = t0 + c
d  = t0 + t1
```



[t_0 = a + b]



[t_1 = t_0 + c]



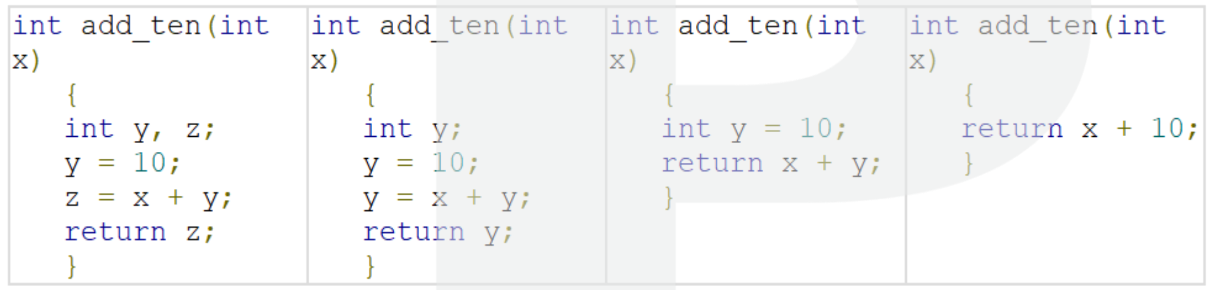
[d = t_0 + t_1]

Peephole Optimization

The peephole optimization technique works for optimization of a small portion of the code but it can be used for intermediate code and target codes. A bunch of statements is checked for optimization.

1. Redundant instruction elimination

We can remove the statement if it is a redundant statement or have multiple occurrences for the same operation.



```
int add_ten(int      int add_ten(int      int add_ten(int      int add_ten(int
x)                   x)                   x)                   x)
   {                    {                    {                    {
   int y, z;            int y;               int y = 10;          return x + 10;
   y = 10;              y = 10;              return x + y;        }
   z = x + y;           y = x + y;           }
   return z;            return y;
   }                    }
```

At the compilation level, the compiler searches for instructions redundant in nature. Multiple loading and storing of instructions may carry the same meaning even if some of them are removed.

For example here in this example, we have multiple move statements.

MOV x, R0
MOV R0, R1

We can be optimized by writing like this,

MOV x, R1

It will not change the outcome of the program.

2. Unreachable code

It is a piece of the code that is never accessed in the program.

```
void add_ten(int x)
{
    return x + 10;
    printf("value of x is %d", x);
}
```

We can see in the above example that the print statement will never execute. So we can remove it from the program without changing the meaning.

3. The flow of control optimization

Sometimes some statements unconditionally jump back and forth without performing any particular task. We can remove those jump instructions.

```
...
MOV R1, R2
GOTO L1
...
L1  :   GOTO L2
L2  :   INC R1

instead of this we can write

...
MOV R1, R2
GOTO L2
...
L2  :   INC R1
```

4. Algebraic expression simplification

1. Instead of writing a=a+1, we can write a++.
2. Instead of assigning a=a+0, we can directly write a.

5. Strength reduction

Some statements consume more time and space so instead of that statement we can use another.

Machine-Independent Optimization

Machine independent optimization will attempt to improve the intermediate code to get a better target code. The piece of the code that is transformed here does not involve any absolute memory location or any CPU registers.

The process of intermediate code generation introduces many inefficiencies like using variables instead of constants, extra copies of variables, repeated evaluation of an expression. Through code optimization, you can remove such efficiencies and improves code.

- It can change the structure of the program sometimes beyond recognition like: unrolls loops, inline functions, eliminates some variables that are programmer-defined.

Example

Code Optimization can perform in the following different ways:

(1) Compile Time Evaluation:

(a) z = 5*(45.0/5.0)*r

Perform 5*(45.0/5.0)*r at compile time.

(b) x = 5.7

y = x/3.6

Evaluate x/3.6 as 5.7/3.6 at compile time.

DIGITAL LEARNING CONTENT

Parul® University

www.paruluniversity.ac.in