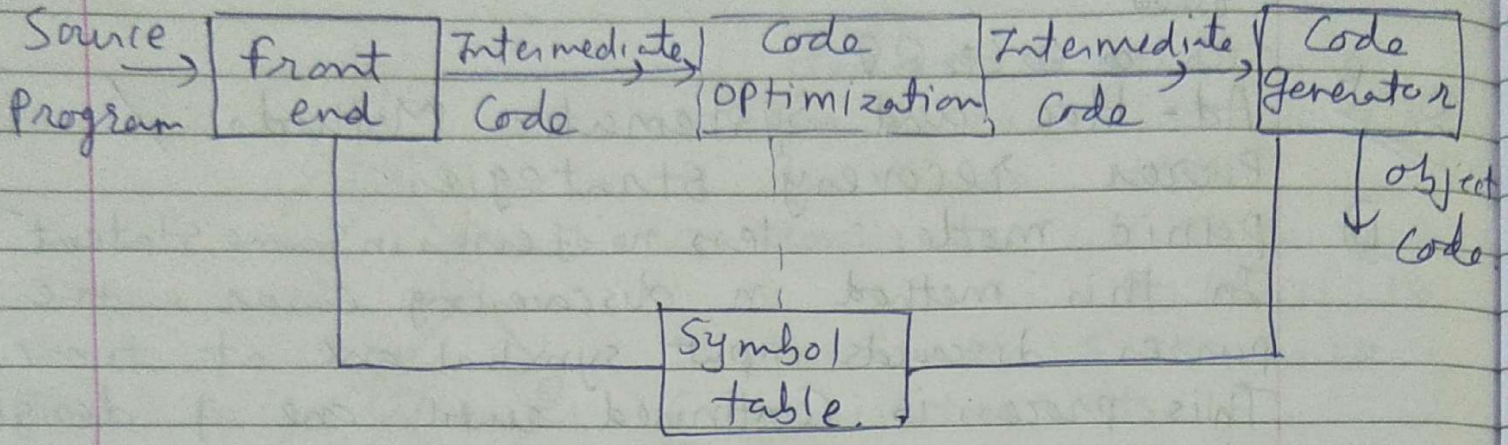
Chapter : 9 : Code generation.

Date: / /
Page No.

Code generation:



→ Code generation is the final phase in the process of Compilation. It takes Intermediate code as an input and generates target machine code as output. The position of code generator in Compilation process is illustrated by following figure.

Properties of Code generation:

→ Correctness
→ High quality
→ Efficient use of resources of the target machine.
→ Quick code generation.

Q-8 Issues in Code generation.

1 Input to Code generator.
2 Target programs.
3 Memory Management
4 Instruction Selection.

Date: / /
Page No.

5 Register allocation.
6 Choice of evaluation order.
7 Approaches to Code generation.

1) Input to the Code generator:

→ Code generation phase takes intermediate code as input.

→ This Intermediate Code along with the symbol table information is used to determine the runtime addressing of the data objects.
↓
denoted by names in IR

→ The intermediate code may be in any form: three address code, quadruple, posix or syntax tree or DAG.

→ Intermediate Code generated by front end should be such that target Machine can easily manipulate it.

→ In the front end: necessary type checking and type conversion needs to be done. The detection of the semantic error should be done before submitting the input to the Code generator. The Code generation phase requires Complete error free intermediate code as input.

2) Target Program:
output of code generator: target code.
the target code comes in three forms such as: absolute machine language.
relocatable machine language
assembly language.

advantage of producing target code in absolute machine code:
it can be placed directly at the fixed memory location and then can be executed immediately.
→ small programs can be quickly compiled.
ex: The WATFIV and PL/C are the compilers which producing the absolute code as output.

advantage of producing the relocatable machine code as output is that subroutines can be compiled separately → linked and loaded for execution by linking loader. If the target machine can not handle the relocation automatically then the compiler must provide explicit relocation information to the loader in order to link the separately compiled subroutine segments.

advantage of producing assembly code as output:
makes the code generation process easier.
The symbolic instructions and macro facilities of assembler can be used to generate the code.
advantageous to have assembly language as output for machines with small memory.

Date: / /
Page No.

3) Memory Management:
Both the front end and code generator performs the task of mapping the names in the source program to addresses to the data objects in run time memory.

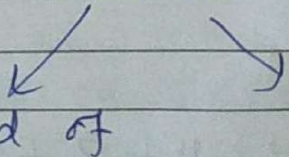
→ names used in three address code.
↳ refers to entry in the Symbol table.

The type in declaration statement → determines the amount of storage needed to store the declared identifier.

→ if three address code contains the labels then those labels can be converted into equivalent memory addresses.
for instance if a reference to goto j is encountered in three address code then appropriate jump instruction can be generated by computing the memory address for label j.

→4) Instruction selection:
The uniformity and completeness of instruction set is an important factor for code generator.

→ two important factors in selection of instructions:



If we do not consider the efficiency of target code then the instruction selection becomes a straightforward task.

Ex. X := a + b
MOV a, R0
ADD b, R0
MOV R0, X

In the above example the code is generated line by line.

→ Such a line by line code generation process generates the poor code because redundancy can be achieved by subsequent lines

for ex:
X := y + z
a := x + t
Code for above statements:
MOV y, R0
ADD z, R0 → y + z
MOV R0, x
MOV x, R0
ADD t, R0 → y + z + t
MOV R0, a.

→
efficient code:
MOV y, R0
ADD z, R0
ADD t, R0 → y + z + t
MOV R0, a.

5) Register allocation:

→ If the instruction contains register operands then such a use becomes shorter and faster than that of using operands in the memory.

→

There are two important activities done while using registers.

1. Register alleration: During register allocation select appropriate set of variables that will reside in registers.

2. Register assignment: During register assignment, pick up the specific register in which corresponding variable will reside.

ex: Consider the three address code:

$t_1 := a + b$

$t_1 := t_1 * c$

$t_1 := t_1 / d$

machine code:

```
MOV a, R0
ADD b, R0
MUL c, R0
DIV d, R0
MOV R0, t1
```

6) choice of evaluation order:

The evalnation order is an important factor.

Some order requires less number of registers to hold the intermediate results that other. Picking up the best order is one of the difficulty in code generation.

→ for avoid this problem by referring the order in which three address code is generated by semantic rules.

Approaches to Code generation:

→ The most important factor for a code generation is that it should produce the correct code. With this approach of code generation various algorithms for generating code are designed.

→ DAG Representation of basic blocks:

Q-B

Directed acyclic graph are useful data structures for implementing transformations on basic blocks.

A DAG for basic block is directed acyclic graph with the following labels on nodes:

1 leaves are labeled by unique identifiers, either variable names or constants. from the operator applied to a name we determine whether the l-value or r value. of a name is needed, most leaves represent r-values. The leaves represent initial values of names and we

Subscript them with 0 to avoid Confusion with labels denoting current values of names as in (3) below.

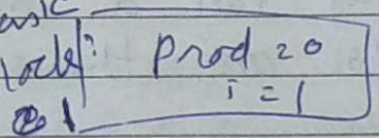
2 Interior nodes are labeled by an operator Symbol.

3 Nodes are also optionally given a sequence of identifiers for labels. The Intention is that interior nodes represent Computed values and the identifiers labeling a node are deemed to have that value.

Three address code for Block B2.

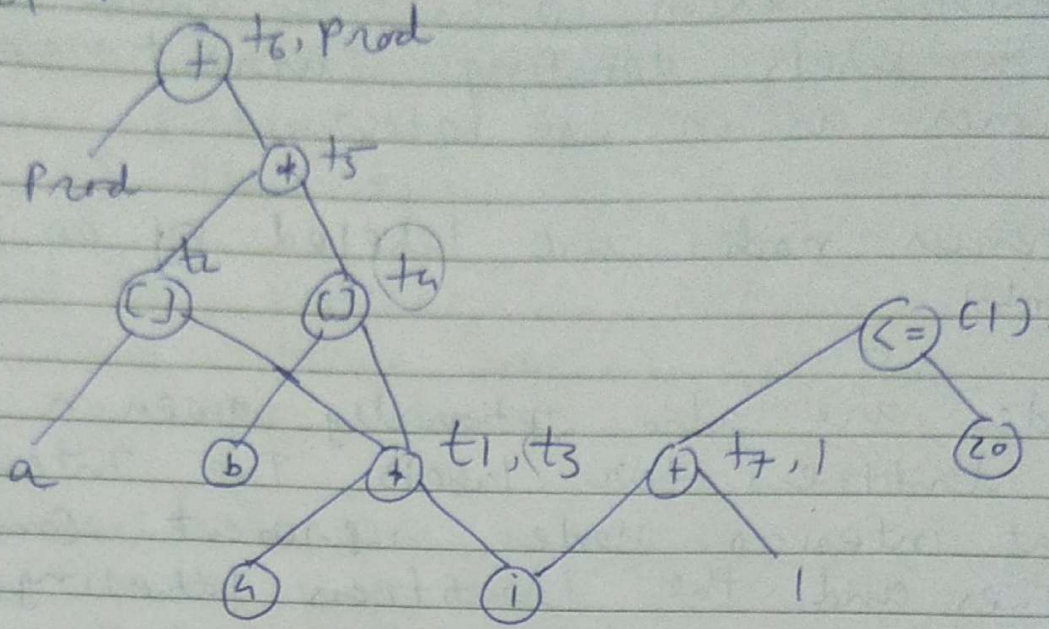
Basic block 2

```
t1 := 4 * i
t2 := a[t1]
t3 := 4 * i
t4 := b[t3]
t5 := t2 * t4
t6 := prod + t5
prod := t6
t7 := i + 1
i := t7
if i <= 20 goto (1)
```



Internal node : is any node of tree that has child node and is thus not leaf node.
Intermediate node between root and leaf : Internal node

DAG :



→ A dag for a basic block has following labels on the nodes:

leaves are labeled by unique identifiers, either variable names or constants.

→ Interior nodes are labeled by an operator symbol.

→ Nodes are also optionally given a sequence of identifiers for labels.

Algorithm for Construction of DAG:

Case (i) $x := y \text{ op } z$
Case (ii) $x := \text{op } y$
Case (iii) $x := y$

With the help of following steps DAG can be Constructed:

Step: 1 If y is undefined then create node(y). Similarly if z is undefined create a node(z)

Step: 2 for the case(i) create a node (op) whose left child is node(y) and node(z) will be the right child. Also check for any Comman Subexpression. for the case(ii) determine whether is a node labeled op, such node will have a child node(y). In case(iii) node n will be node(y).

Step: 3 Delete x from list of identifiers for node(x). Append x to list of attached identifiers for node n found in 2.

Application of DAG:

DAG is used in:

1. Determining the Comman Sub expression
2. Determining which names are used inside the block and Computed outside the block

Date: / /
Page No.

3 Determining which statements of the block could have their computed value outside the block.

4 Simplifying the list of quadruples by eliminating the common sub expressions and not performing the assignment of the form x := y unless and until it is a must.

→ Peephole Optimization:

→ Definition:

Peephole optimization is a simple and effective technique for locally improving target code. This technique is applied to improve the performance of the target program by examining the short sequence of target instructions and replacing these instructions by shorter or faster sequence

peephole: a small window is moved over the target code and transformations can be made. And hence is the name.

Characteristics of peephole optimization can be applied on the target code using following characteristics:

1 Redundant instruction elimination
2 flow of control optimization
3 Algebraic Simplification.
4 Use of machine idioms
5 Strength reduction.

1 Redundant instruction elimination:
Especially the redundant loads and Stores can be eliminated in this types of transformations:

for example:

```
Mov R0, x
Mov x, R0
```

→ We can eliminate the second instruction since x is already in R0. But if (Mov x, R0) is a label Statement then we can not remove it.

We can eliminate the unreachable instructions for example, following is a piece of C code.

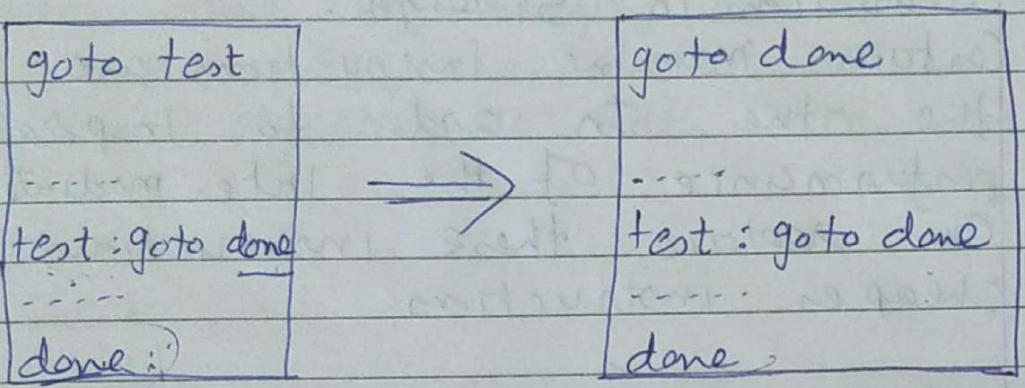
unreachable
```
Sum = 0
if ( Sum@ = = 1)
Printf("/d", Sum);
```

Now this if Statement will never get executed hence we can eliminate such a unreachable code.

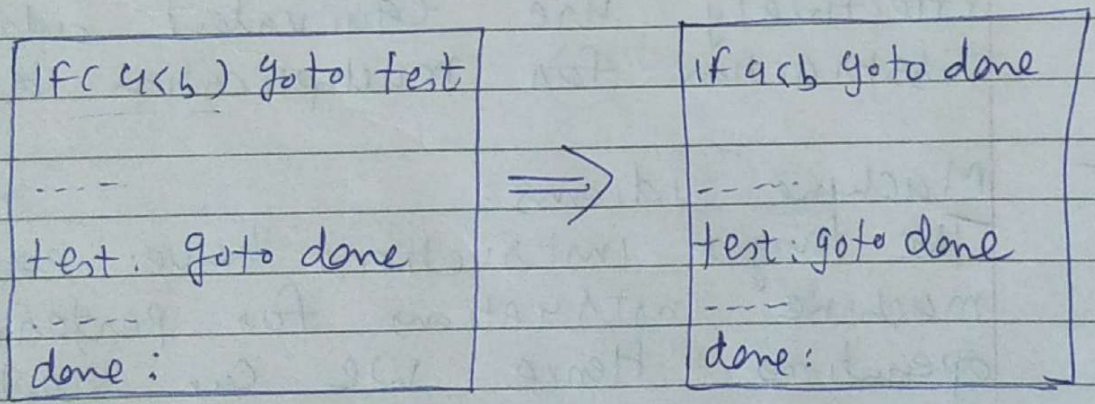
```
int func(int a, int b)
{
 c = a + b;
return c;
printf("%d", c);
```

2 flow of Control optimization:
Using peephole optimization unnecessary jumps on jumps can be eliminated.

for ex:



→ Thus we reduce one jump by this transformation:



3 Algebric Simplification:
peephole optimization is an effective technique for algebric Simplification

The statement such as:

```
x := x + 0
```

or

```
x := x * 1
```

Can be eliminated by peephole optimizatio

4 Reduction in strength:
Certain machine instructions are cheaper than the other. In order to improve the performance of the intermediate code we can replace these instructions by equivale cheaper instructions.

ex: x^2 is cheaper than $x * x$.
addition and substraction is cheaper than multiplication and division. So, we can effectively use equivalent addition and substraction for multiplication division

5 Machine Idioms:
The target instructions have equivalent machine instructions for performing some operations. Hence we can replace these instructions by equivalent machine instructio in order to improve the efficiency. for ex:

Date: / /
Page No.

Generating Code from DAGs:

- Rearranging order
- Heuristic ordering
- labeling algorithm.

1 Rearranging order:

The order of three address code affects the cost of the object code being generated. In the sense that by changing the order in which computations are done we can obtain the object code with minimum cost.

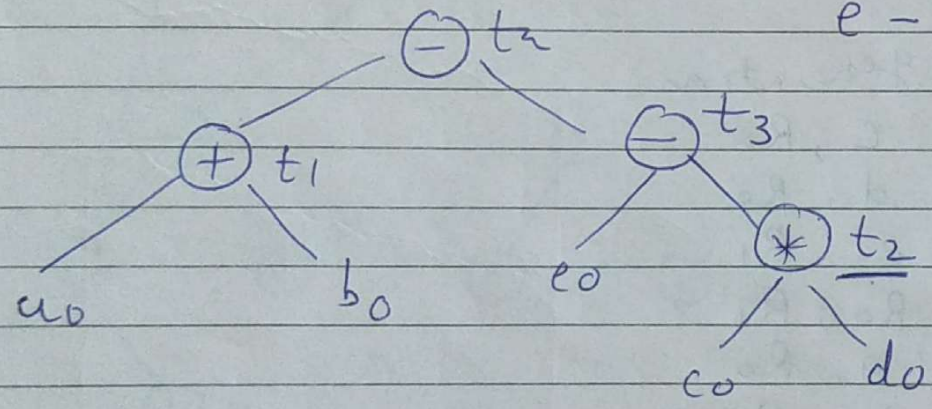
for example:

$t_1 := a + b$

$t_2 := c + d$

$t_3 := e - t_2$

$t_4 := t_1 - t_3$ for the expression $(a+b) - (e - (c+d))$



Code generation :

```
MOV   a, R0
ADD   b, R0      R0 : a+b
MOV   c, R1
ADD   d, R1      R1 : c+d
MOV   R0, t1     t1 : a+b
MOV   e, R0      R0 : e
SUB   R1, R0     R0 : e - c+d
MOV   t1, R1     R1 : a+b
SUB   R0, R1     R1 : a+b - (e - (c+d))
MOV   R1, t4     t4 : a+b - (e - (c+d))
```

Suppose we rearranged the order of the statements so that the computation of t_1 occurs immediately before that of t_4 is:

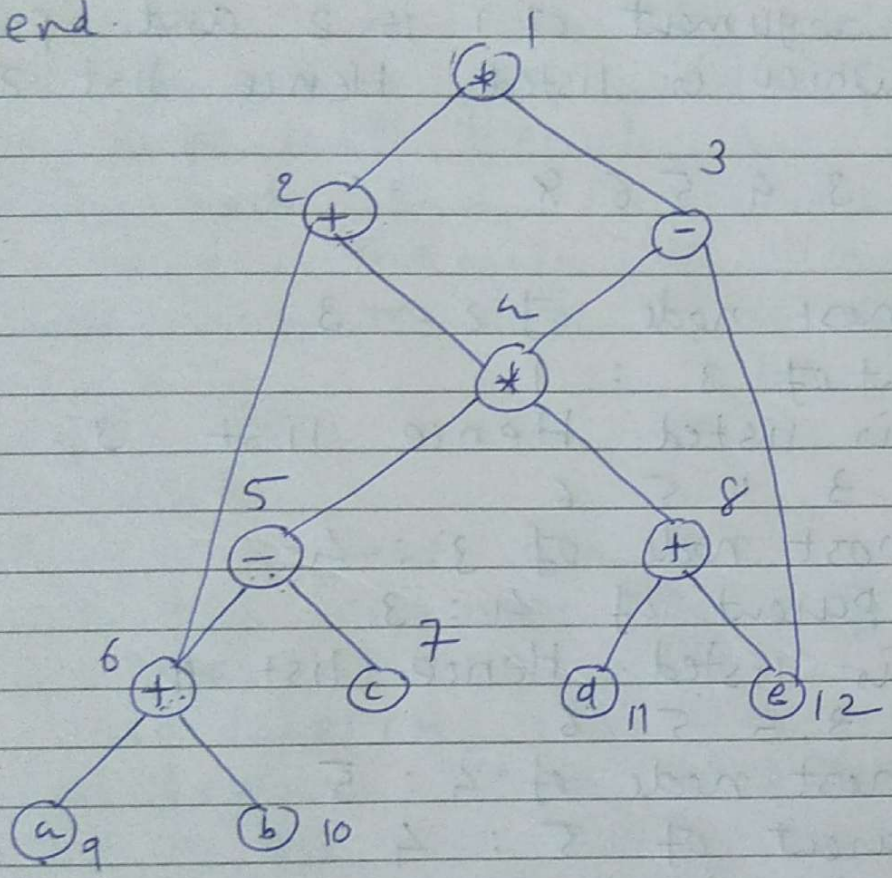
$t_2 := c + d$
$t_3 := e - t_2$
$t_1 := a + b$
$t_4 := t_1 - t_3$

Code generation:

```
MOV   c, R0
ADD   d, R0
MOV   e, R1
SUB   R0, R1
MOV   a, R0
ADD   b, R0
SUB   R1, R0
MOV   R0, t4
```

2 Heuristic Ordering :
obtain all the interior nodes. Consider those interior nodes as unlisted interior nodes.

(1) while unlisted interior nodes remain do begin
(2) select an unlisted node n, all of whose parents have been listed;
(3) list n;
(4) while the left most child of m of n has no unlisted parents and is not a leaf do begin
(5) list m;
(6) n := m
end
end



1	2	3	4	5	6	8

8	6	5	4	3	2	1

$t_8 := d + e$
$t_6 := a + b$
$t_5 := t_6 - c$
$t_4 := t_5 * t_8$
$t_3 := t_4 - e$
$t_2 := t_6 + t_4$
$t_1 := t_2 * t_3$.

→ Consider the unlisted interior nodes 1 2 3 4 5 6 8

→ Initially the only node with unlisted parent is 1.

→ Now left argument of 1 is 2 and parent of 2 is 1 which is listed. Hence list 2.

→

t 2 3 4 5 6 8

→ left most node of 2 : 3
parent of 3 : 1
which is listed. Hence list 3.
t 2 3 4 5 6
left most node of 3 : 4
parent of 4 : 3
which is listed. Hence list 4.
t 2 3 4 5 6
left most node of 4 : 5
parent of 5 : 4
which is listed. Hence list 5
only node 8 is remaining from the unlisted interior nodes we will list it.
reversing the list : 8 6 5 4 3 2 1.

Date: / /
Page No.

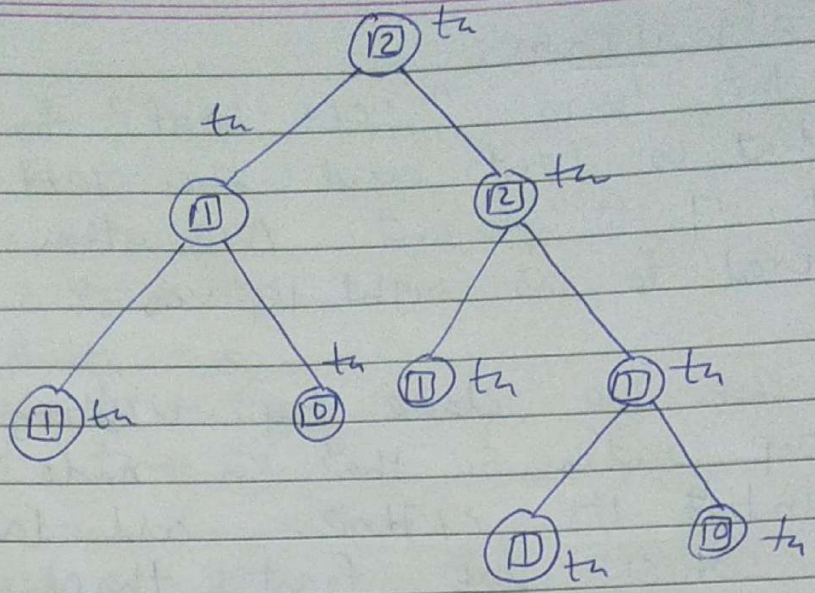
Labeling algorithm:
We use the term "left leaf" to mean a node that is leaf and the left most descendent of its parent. All other leaves are referred to as right leaves.

labeling can be done by visiting node in a bottom up order so that a node is not visited until its children all labeled. The order in which the first three nodes are created is suitable if the parse tree is used as intermediate code. So in this case the labels can be computed as syntax directed translation. The algorithm below is for computing the label at node n. In the important special case that n is a binary node and its children have labels l_1 and l_2, formula of line (6) reduces to

$$label(n) = \begin{cases} \max(l_1, l_2) & \text{if } l_1 \neq l_2 \\ l_1 + 1 & \text{if } l_1 == l_2 \end{cases}$$

Algorithm:

```
1  if n is a leaf then
2    if n is left most child of its parents then
3      label(n) = 1
4    else label(n) = 0
   else begin   /* n is interior node */
5    let n1, n2, .... nk be the children of n
   ordered by label.
     So label(n1) >= label(n2) >= .... >= label(nk)
     label(n) = max(label(ni) + i - 1)
   end.
```



Prepared by kinjal patel