5
UNIT

Code Generation

CONTENTS

PART-1

Code Generation : Design Issues.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 5.1. What is code generation ? Discuss the design issues of code generation.

Answer

1. Code generation is the final phase of compiler.
2. It takes as input the Intermediate Representation (IR) produced by the front end of the compiler, along with relevant symbol table information, and produces as output a semantically equivalent target program as shown in Fig. 5.1.1.

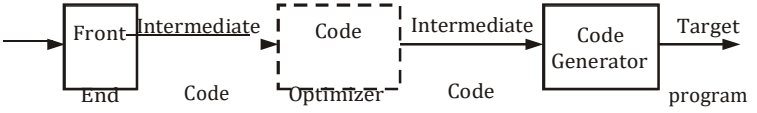


Fig. 5.1.1. Position of code generator.

Design issues of code generator are :

1. **Input to the code generator :**
   a. The input to the code generator is the intermediate representation of the source program produced by the front end, along with information in the symbol table.
   b. IR includes three address representations and graphical representations.
2. **The target program :**
   a. The instruction set architecture of the target machine has a significant impact on the difficulty of constructing a good code generator that produces high quality machine code.
   b. The most common target machine architectures are RISC (Reduced Instruction Set Computer), CISC (Complex Instruction Set Computer), and stack based.
3. **Instruction selection :**

a. The code generator must map the IR program into a code sequence that can be executed by the target machine.

b. If the IR is high level, the code generator may translate each IR statement into a sequence of machine instructions using code templates.

4. **Register allocation :**
   a. A key problem in code generation is deciding what values to hold in which registers on the target machine do not have enough space to hold all values.
   b. Values that are not held in registers need to reside in memory. Instructions involving register operands are invariably shorter and faster than those involving operands in memory, so efficient utilization of registers is particularly important.
   c. The use of registers is often subdivided into two subproblems :
      i. Register allocation, during which we select the set of variables that will reside in registers at each point in the program.
      ii. Register assignment, during which we pick the specific register that a variable will reside in.

5. **Evaluation order :**
   a. The order in which computations are performed can affect the efficiency of the target code.
   b. Some computation orders require fewer registers to hold intermediate results than others.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 5.2. **Discuss addresses in the target code.**

Answer

1. Addresses in the target code show how names in the IR can be converted into addresses in the target code by looking at code generation for simple procedure calls and returns using static and stack allocation.
2. Addresses in the target code represent executing program runs in its own logical address space that was partitioned into four code and data areas :
   a. A statically determined area code that holds the executable target code. The size of the target code can be determined at compile time.

b. A statically determined data area static for holding global constants and other data generated by the compiler. The size of the global constants and compiler data can also be determined at compile time.

c. A dynamically managed area heap for holding data objects that are allocated and freed during program execution. The size of the heap cannot be determined at compile time.

d. A dynamically managed area stack for holding activation records as they are created and destroyed during procedure calls and returns. Like the heap, the size of the stack cannot be determined at compile time.

PART-3

Basic Blocks and Flow Graphs, Optimization of Basic Blocks, Code Generator.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 5.3. **Write an algorithm to partition a sequence of three address statements into basic blocks.** **AKTU 2016-17, Marks 10**

Answer

The algorithm for construction of basic block is as follows :

Input : A sequence of three address statements.

Output : A list of basic blocks with each three address statements in exactly one block.

Method :

1. We first determine the set of leaders, the first statement of basic block. The rules we use are given as :
   a. The first statement is a leader.
   b. Any statement which is the target of a conditional or unconditional goto is a leader.
   c. Any statement which immediately follows a conditional goto is a leader.
2. For each leader construct its basic block, which consist of leader and all statements up to the end of program but not including the next leader. Any statement not placed in the block can never be executed and may now be removed, if desired.

Que 5.4. **Explain flow graph with example.**

Answer

1. A flow graph is a directed graph in which the flow control information is added to the basic blocks.
2. The nodes to the flow graph are represented by basic blocks.
3. The block whose leader is the first statement is called initial blocks.
4. There is a directed edge from block B_{i-1} to block B_i if B_i immediately follows B_{i-1} in the given sequence. We can say that B_{i-1} is a predecessor of B_i.

For example : Consider the three address code as :

1. prod := 0
2. i := 1
3. t_1 := 4 * i
4. t_2 := $a[t_1]$ /* computation of $a[i]$ */
5. t_3 := 4 * i
6. t_4 := $b[t_3]$ /* computation of $b[i]$ */
7. t_5 := t_2 * t_4
8. t_6 := prod + t_5
9. prod := t_6
10. t_7 := i + 1
11. i := t_7
12. if i <= 10 goto (3)

The flow graph for the given code can be drawn as follows :

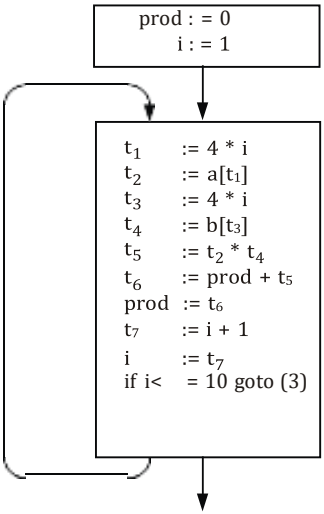


Fig. 5.4.1. Flow graph.

Que 5.5. **What is loop ? Explain what constitute a loop in a flow graph.**

Answer

Loop is a collection of nodes in the flow graph such that :

1. All such nodes are strongly connected *i.e.*, there is always a path from any node to any other node within that loop.
2. The collection of nodes has unique entry. That means there is only one path from a node outside the loop to the node inside the loop.
3. The loop that contains no other loop is called inner loop.

Following term constitute a loop in flow graph :

1. Dominators :

a. In control flow graphs, a node d dominates a node n if every path from the entry node to n must go through d. This is denoted as d dom n.

b. By definition, every node dominates itself.

c. A node d strictly dominates a node n if d dominates n and d is not equal to n.

d. The immediate dominator (or *i*dom) of a node n is the unique node that strictly dominates n but does not strictly dominate any other node that strictly dominates n. Every node, except the entry node, has an immediate dominator.

e. A dominator tree is a tree where each node's children are those nodes it immediately dominates. Because the immediate dominator is unique, it is a tree. The start node is the root of the tree.

2. Natural loops :

a. The natural loop can be defined by a back edge $n \rightarrow d$ such that there exists a collection of all the nodes that can reach to n without going through d and at the same time d can also be added to this collection.

b. Loop in a flow graph can be denoted by $n \rightarrow d$ such that d dom n.

c. These edges are called back edges and for a loop there can be more than one back edge.

d. If there is $p \rightarrow q$ then q is a head and p is a tail and head dominates tail.

3. Pre-header :

a. The pre-header is a new block created such that successor of this block is the header block.

b. All the computations that can be made before the header block can be made before the pre-header block.

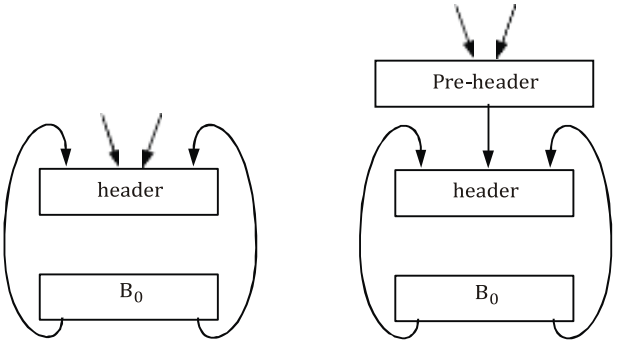


Fig. 5.5.1. Pre-header.

4. **Reducible flow graph :**
   a. A flow graph G is reducible graph if an only if we can partition the edges into two disjointed groups *i.e.*, forward edges and backward edges.
   b. These edges have following properties :
      i. The forward edge forms an acyclic graph.
      ii. The backward edges are such edges whose heads dominate their tails.
   c. The program structure in which there is exclusive use of if-then, while-do or goto statements generates a flow graph which is always reducible.

Que 5.6. **Discuss in detail the process of optimization of basic blocks. Give an example.** **AKTU 2016-17, Marks 10**

OR

What are different issues in code optimization ? Explain it with proper example.

Answer

Different issues in code optimization are :

1. **Function preserving transformation :** The function preserving transformations are basically divided into following types :
   a. Common sub-expression elimination :
      i. A common sub-expression is nothing but the expression which is already computed and the same expression is used again and again in the program.
      ii. If the result of the expression not changed then we eliminate computation of same expression again and again.

For example :

Before common sub-expression elimination :

```
a = t * 4 – b + c;
.....
.....
m = t * 4 – b + c;
.....
.....
n = t * 4 – b + c;
```

After common sub-expression elimination :

```
temp = t * 4 – b + c;
a = temp;
.....
.....
m = temp;
.....
.....
n = temp;
```

iii. In given example, the equation $a = t * 4 – b + c$ is occurred most of the times. So it is eliminated by storing the equation into temp variable.

b. Dead code elimination :

i. Dead code means the code which can be emitted from program and still there will be no change in result.

ii. A variable is live only when it is used in the program again and again. Otherwise, it is declared as dead, because we cannot use that variable in the program so it is useless.

iii. The dead code occurred during the program is not introduced intentionally by the programmer.

For example :

```
# Define False = 0
!False = 1
If(!False)
{
.....
.....
}
```

iv. If false becomes zero, is guaranteed then code in 'IF' statement will never be executed. So, there is no need to generate or write code for this statement because it is dead code.

c. Copy propagation :

i. Copy propagation is the concept where we can copy the result of common sub-expression and use it in the program.

ii. In this technique the value of variable is replaced and computation of an expression is done at the compilation time.

For example :

$pi = 3.14;$

$r = 5;$

Area = $pi * r * r;$

Here at the compilation time the value of *pi* is replaced by 3.14 and *r* by 5.

d. Constant folding (compile time evaluation) :

i. Constant folding is defined as replacement of the value of one constant in an expression by equivalent constant value at the compile time.

ii. In constant folding all operands in an operation are constant. Original evaluation can also be replaced by result which is also a constant.

For example : $a = 3.14157/2$ can be replaced by $a = 1.570785$ thereby eliminating a division operation.

2. Algebraic simplification :

a. Peephole optimization is an effective technique for algebraic simplification.

b. The statements such as

$x := x + 0$

or $x := x * 1$

can be eliminated by peephole optimization.

Que 5.7. Write a short note on transformation of basic blocks.

Answer

Transformation :

1. A number of transformations can be applied to basic block without changing set of expression computed by the block.
2. Transformation helps us in improving quality of code and act as optimizer.
3. There are two important classes as local transformation that can be applied to the basic block :

a. Structure preserving transformation : They are as follows :

i. Common sub-expression elimination : Refer Q. 5.6, Page 5–7C, Unit-5.

ii. Dead code elimination : Refer Q. 5.6, Page 5–7C, Unit-5.

iii. Interchange of statement : Suppose we have a block with the two adjacent statements,

temp1 = $a + b$

temp2 = $m + n$

Then we can interchange the two statements without affecting the value of the block if and only if neither 'm' nor 'n' is temporary variable temp1 and neither 'a' nor 'b' is temporary variable temp2. From the given statements we can conclude that a normal form basic block allow us for interchanging all the statements if they are possible.

b. Algebraic transformation : Refer Q. 5.6, Page 5–7C, Unit-5.

PART-4

Machine Independent Optimizations, Loop Optimization.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 5.8. What is code optimization ? Discuss the classification of code optimization.

Answer

Code optimization :

1. The code optimization refers to the techniques used by the compiler to improve the execution efficiency of generated object code.
2. It involves a complex analysis of intermediate code and performs various transformations but every optimizing transformation must also preserve the semantic of the program.

Classification of code optimization :

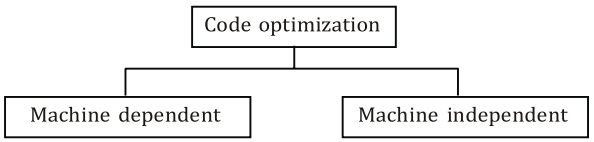


Fig. 5.8.1. Classification of code optimization.

1. **Machine dependent :** The machine dependent optimization can be achieved using following criteria :
   a. Allocation of sufficient number of resources to improve the execution efficiency of the program.
   b. Using immediate instructions wherever necessary.

c. The use of intermix instructions along with the data increases the speed of execution.

2. **Machine independent :** The machine independent optimization can be achieved using following criteria :
   a. The code should be analyzed completely and use alternative equivalent sequence of source code that will produce a minimum amount of target code.
   b. Use appropriate program structure in order to improve the efficiency of target code.
   c. By eliminating the unreachable code from the source program.
   d. Move two or more identical computations at one place and make use of the result instead of each time computing the expressions.

Que 5.9. **Explain local optimization.**

Answer

1. Local optimization is a kind of optimization in which both the analysis and the transformations are localized to a basic block.
2. The transformations in local optimization are called as local transformations.
3. The name of transformation is usually prefixed with 'local' while referring to the local transformation.
4. There are "local" transformations that can be applied to program to attempt an improvement.

   For example : The elimination of common sub-expression, provided A is not an alias for B or C, the assignments :

$$A := B + C + D$$
$$E := B + C + F$$

   might be evaluated as

$$T_1 := B + C$$
$$A := T_1 + D$$
$$E := T_1 + F$$

   In the given example, $B + C$ is stored in T_1 which act as local optimization of common sub-expression.

Que 5.10. **Explain what constitute a loop in a flow graph and how will you do loop optimizations in code optimization of a compiler.**

AKTU 2018-19, Marks 07

OR

Write a short note on loop optimization.

AKTU 2017-18, Marks 05

Answer

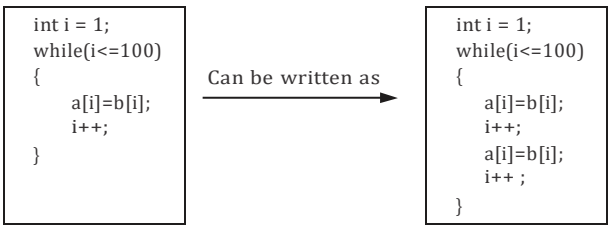
Following term constitute a loop in flow graph : Refer Q. 5.5, Page 5–5C, Unit-5.

Loop optimization is a process of increasing execution time and reducing the overhead associated with loops.

The loop optimization is carried out by following methods :

1. **Code motion :**
   a. Code motion is a technique which moves the code outside the loop.
   b. If some expression in the loop whose result remains unchanged even after executing the loop for several times, then such an expression should be placed just before the loop (*i.e.*, outside the loop).
   c. Code motion is done to reduce the execution time of the program.
2. **Induction variables :**
   a. A variable *x* is called an induction variable of loop *L* if the value of variable gets changed every time.
   b. It is either decremented or incremented by some constant.
3. **Reduction in strength :**
   a. In strength reduction technique the higher strength operators can be replaced by lower strength operators.
   b. The strength of certain operator is higher than other.
   c. The strength reduction is not applied to the floating point expressions because it may yield different results.
4. **Loop invariant method :** In loop invariant method, the computation inside the loop is avoided and thereby the computation overhead on compiler is avoided.
5. **Loop unrolling :** In this method, the number of jumps and tests can be reduced by writing the code two times.

   For example :



6. **Loop fusion or loop jamming :** In loop fusion method, several loops are merged to one loop.

For example :

```
for i:=1 to n do
for j:=1 to m do
a[i,j]:=10
```

Can be written as

```
for i:=1 to n*m do
a[i]:=10
```

Que 5.11. **Consider the following three address code segments :**

PROD **: = 0**
1. ***I*** **: = 1**
2. ***T1*** **: = 4*** ***I***
3. ***T2*** **: =** ***addr*****(*****A*****) – 4**
4. ***T3*** **: =** ***T2*** **[*****T1*****]**
5. ***T4*** **: =** ***addr*****(*****B*****) – 4**
6. ***T5*** **: =** ***T4*****[*****T1*****]**
7. ***T6*** **: =** ***T3**********T5**
8. ***PROD*** **: =** ***PROD*** **+** ***T6***
9. ***I*** **: =** ***I*** **+ 1**
10. **If** ***I*** **< = 20 goto (3)**

a. Find the basic blocks and flow graph of above sequence.

b. Optimize the code sequence by applying function preserving transformation optimization technique.

AKTU 2017-18, Marks 10

OR

Consider the following sequence of three address codes :
1. **Prod : = 0**
2. ***I*** **: = 1**
3. T_1 **: = 4*** ***I***
4. T_2 **: = addr (*****A*****) – 4**
5. T_3 **: =** T_2 **[**T_1**]**
6. T_4 **: = addr (*****B*****) – 4**
7. T_5 **: =** T_4 **[**T_1**]**
8. T_6 **: =** T_3***** T_5
9. **Prod : = Prod +** T_6
10. ***I*** **=** ***I*** **+ 1**
11. **If** ***I*** **<= 20 goto (3)**

Perform loop optimization. **AKTU 2015-16, Marks 10**

Answer

a. Basic blocks and flow graph :

1. As first statement of program is leader statement.

   ∴ PROD = 0 is a leader.

2. Fragmented code represented by two blocks is shown below :

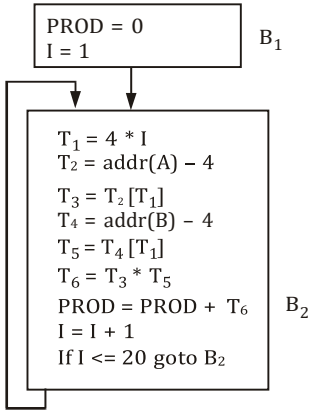


Fig. 5.11.1.

b. Function preserving transformation :

1. **Common sub-expression elimination :** No any block has any sub expression which is used two times. So, no change in flow graphs.
2. **Copy propagation :** No any instruction in the block B_2 is direct assignment *i.e.*, in the form of $x = y$. So, no change in flow graph and basic block.
3. **Dead code elimination :** No any instruction in the block B_2 is dead. So, no change in flow graph and basic block.
4. **Constant folding :** No any constant expression is present in basic block. So, no change in flow graph and basic block.

Loop optimization :

1. **Code motion :** In block B_2 we can see that value of T_2 and T_4 is calculated every time when loop is executed. So, we can move these two instructions outside the loop and put in block B_1 as shown in Fig. 5.11.2.

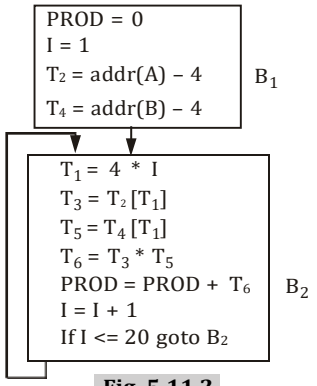


Fig. 5.11.2.

2. **Induction variable :** A variable I and T_1 are called an induction variable of loop L because every time the variable I change the value of T_1 is also change. To remove these variables we use other method that is called reduction in strength.

3. **Reduction in strength :** The values of I varies from 1 to 20 and value T_1 varies from (4, 8, ... , 80).

   Block B_2 is now given as

$$T_1 = 4 * I \leftarrow \text{In block } B_1$$

$$\boxed{T_1 = T_1 + 4}\ B_2$$

   Now final flow graph is given as

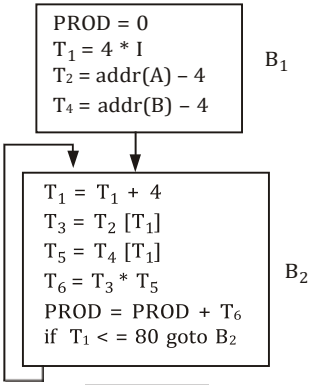


Fig. 5.11.3

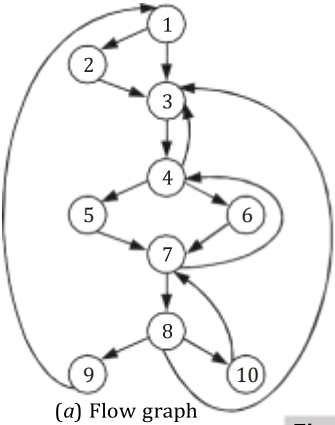
Que 5.12. **Write short notes on the following with the help of example :**

i. Loop unrolling
ii. Loop jamming
iii. Dominators
iv. Viable prefix

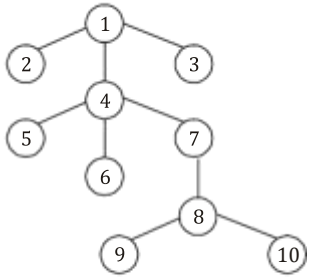
AKTU 2018-19, Marks 07

Answer

i. Loop unrolling : Refer Q. 5.10, Page 5–11C, Unit-5.
ii. Loop jamming : Refer Q. 5.10, Page 5–11C, Unit-5.
iii. Dominators : Refer Q. 5.5, Page 5–5C, Unit-5.
For example : In the flow graph,



(*a*) Flow graph



(*b*) Dominator tree

Fig. 5.12.1.

Initial Node, Node1 dominates every Node.

Node 2 dominates itself. Node 3 dominates all but 1 and 2. Node 4 dominates all but 1,2 and 3.

Node 5 and 6 dominates only themselves, since flow of control can skip around either by go in through the other. Node 7 dominates 7, 8, 9 and 10. Node 8 dominates 8, 9 and 10.

Node 9 and 10 dominates only themselves.

iv. Viable prefix : Viable prefixes are the prefixes of right sentential forms that can appear on the stack of a shift-reduce parser.

For example :

Let : $S \rightarrow x_1x_2x_3x_4$

$A \rightarrow x_1\,x_2$

Let $w = x_1x_2x_3$

SLR parse trace :

STACK	INPUT
\$	$x_1x_2x_3$
\$ x_1	x_2x_3
\$ x_1x_2	x_3
\$$A$	x_3
\$$AX_3$	\$
.	

As we see, $x_1x_2x_3$ will never appear on the stack. So, it is not a viable prefix.

PART-5

DAG Representation of Basic Blocks.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 5.13. **What is DAG ? What are its advantages in context of optimization ?**

OR

Write a short note on direct acyclic graph.

AKTU 2017-18, Marks 05

Answer

DAG :

1. The abbreviation DAG stands for Directed Acyclic Graph.
2. DAGs are useful data structure for implementing transformations on basic blocks.
3. A DAG gives picture of how the value computed by each statement in the basic block is used in the subsequent statement of the block.
4. Constructing a DAG from three address statement is a good way of determining common sub-expressions within a block.
5. A DAG for a basic block has following properties :
   a. Leaves are labeled by unique identifier, either a variable name or constants.
   b. Interior nodes are labeled by an operator symbol.
   c. Nodes are also optionally given a sequence of identifiers for labels.
6. Since, DAG is used in code optimization and output of code optimization is machine code and machine code uses register to store variable used in the source program.

Advantage of DAG :

1. We automatically detect common sub-expressions with the help of DAG algorithm.
2. We can determine which identifiers have their values used in the block.
3. We can determine which statements compute values which could be used outside the block.

Que 5.14. **What is DAG ? How DAG is created from three address code ? Write algorithm for it and explain it with a relevant example.**

Answer

DAG : Refer Q. 5.13, Page 5–17C, Unit-5.

Algorithm :

Input : A basic block.

Output : A DAG with label for each node (identifier).

Method :

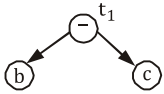
1. Create nodes with one or two left and right children.
2. Create linked list of attached identifiers for each node.
3. Maintain all identifiers for which a node is associated.
4. Node (identifier) represents value that identifier has the current point in DAG construction process. Symbol table store the value of node (identifier).
5. If there is an expression of the form $x = y$ op z then DAG contain "op" as a parent node and node(y) as a left child and node(z) as a right child.

For example :

Given expression : $a * (b - c) + (b - c) * d$

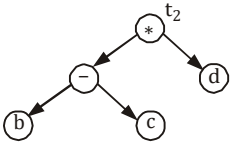
The construction of DAG with three address code will be as follows :

Step 1 :



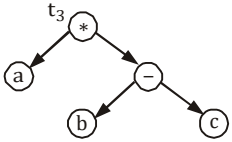
$t_1 = b - c$

Step 2 :



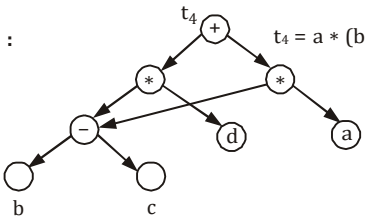
$t_2 = (b - c) * d$

Step 3 :



$t_3 = a * (b - c)$

Step 4 :



$t_4 = a * (b - c) + (b - c) * d$

Que 5.15. **How DAG is different from syntax tree ? Construct the DAG for the following basic blocks :**

$a := b + c$

$b := b - d$

$c := c + d$

$e = b + c$

Also, explain the key application of DAG. **AKTU 2015-16, Marks 15**

Answer

DAG v/s Syntax tree :

1. Directed Acyclic Graph is a data structure for transformations on the basic block. While syntax tree is an abstract representation of the language constructs.
2. DAG is constructed from three address statement while syntax tree is constructed directly from the expression.

DAG for the given code is :

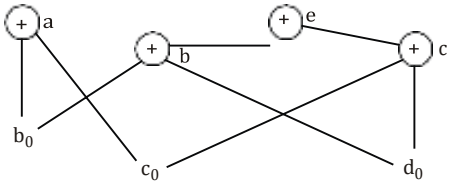


Fig. 5.15.1.

1. The two occurrences of sub-expressions $b + c$ compute the same value.
2. Value computed by a and e are same.

Applications of DAG :

1. **Scheduling :** Directed acyclic graphs representations of partial orderings have many applications in scheduling for systems of tasks.
2. **Data processing networks :** A directed acyclic graph may be used to represent a network of processing elements.
3. **Data compression :**Directed acyclic graphs may also be used as a compact representation of a collection of sequences. In this type of application, one finds a DAG in which the paths form the sequences.
4. It helps in finding statement that can be recorded.

Que 5.16. **Define a directed acyclic graph. Construct a DAG and write the sequence of instructions for the expression :**

$a + a * (b - c) + (b - c) * d$. **AKTU 2016-17, Marks 15**

Answer

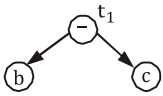
Directed acyclic graph : Refer Q. 5.13, Page 5–17C, Unit-5.

Numerical :

Given expression : $a + a * (b - c) + (b - c) * d$

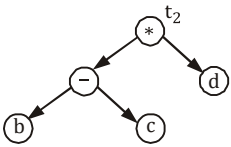
The construction of DAG with three address code will be as follows :

Step 1 :



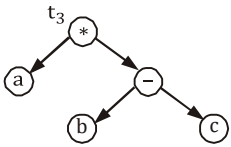
$t_1 = b - c$

Step 2 :



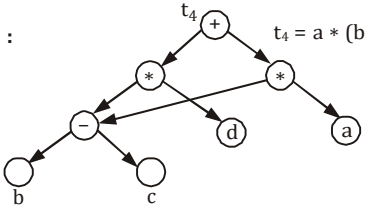
$t_2 = (b - c) * d$

Step 3 :



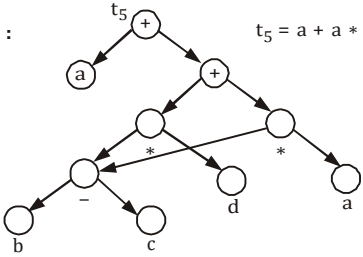
$t_3 = a * (b - c)$

Step 4 :



$t_4 = a * (b - c) + (b - c) * d$

Step 5 :



$t_5 = a + a * (b - c) + (b - c) * d$

DAG ?

How would you represent the following equation

Que 5.17.

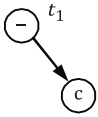
AKTU 2018-19, Marks 07

a = b * – c + b * – c

Answer

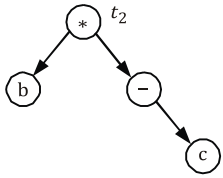
Code representation using DAG of equation : $a = b * - c + b * - c$

Step 1 :



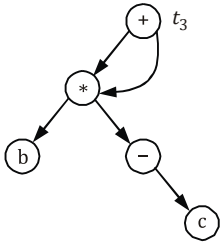
$t_1 = - c$

Step 2 :



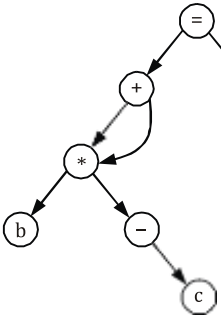
$t_2 = b * t_1$

Step 3 :



$t_3 = t_2 + t_2$

Step 4 :



$t_4 = a$

Que 5.18. Give the algorithm for the elimination of local and global common sub-expressions algorithm with the help of example.

AKTU 2017-18, Marks 10

Answer

Algorithm for elimination of local common sub-expression : DAG algorithm is used to eliminate local common sub-expression.

DAG : Refer Q. 5.13, Page 5–17C, Unit-5.

Algorithm for elimination of global common sub-expressions :

1. An expression is defined at the point where it is assigned a value and killed when one of its operands is subsequently assigned a new value.
2. An expression is available at some point p in a flow graph if every path leading to p contains a prior definition of that expression which is not subsequently killed.
3. Following expressions are used :
   a. avail[*B*] = set of expressions available on entry to block *B*
   b. exit[*B*] = set of expressions available on exit from *B*
   c. killed[*B*] = set of expressions killed in *B*
   d. defined[*B*] = set of expressions defined in *B*
   e. exit[*B*] = avail[*B*] – killed[*B*] + defined[*B*]

Algorithm :

1. First, compute defined and killed sets for each basic block
2. Iteratively compute the avail and exit sets for each block by running the following algorithm until we get a fixed point:
   a. Identify each statement *s* of the form *a* = *b* op *c* in some block *B* such that *b* op *c* is available at the entry to *B* and neither *b* nor *c* is redefined in *B* prior to *s*.
   b. Follow flow of control backwards in the graph passing back to but not through each block that defines *b* op *c*. the last computation of *b* op *c* in such a block reaches *s*.
   c. After each computation *d* = *b* op *c* identified in step 2(*a*), add statement *t* = *d* to that block (where *t* is a new temp *d*).
   d. Replace *s* by *a* = *t*

PART-6

Value Numbers and Algebraic Laws, Global Data Flow Analysis.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 5.19. **Write a short note on data flow analysis.**

OR

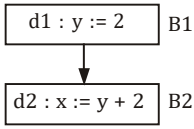
What is data flow analysis ? How does it use in code optimization ?

Answer

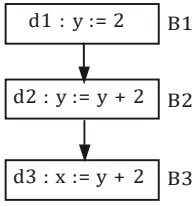
1. Data flow analysis is a process in which the values are computed using data flow properties.
2. In this analysis, the analysis is made on data flow.
3. A program's Control Flow Graph (CFG) is used to determine those parts of a program to which a particular value assigned to a variable might propagate.
4. A simple way to perform data flow analysis of programs is to set up data flow equations for each node of the control flow graph and solve them by repeatedly calculating the output from the input locally at each node until the whole system stabilizes, *i.e.*, it reaches a fix point.
5. Reaching definitions is used by data flow analysis in code optimization.

Reaching definitions :

1. A definition *D* reaches at point *p* if there is a path from *D* to *p* along which *D* is not killed.



2. A definition *D* of variable *x* is killed when there is a redefinition of *x*.



3. The definition *d*1 is said to a reaching definition for block *B*2. But the definition *d*1 is not a reaching definition in block *B*3, because it is killed by definition *d*2 in block *B*2.

Que 5.20. Write short notes (any two) :

i. Global data flow analysis
ii. Loop unrolling
iii. Loop jamming

AKTU 2015-16, Marks 15

OR

Write short note on global data analysis.

AKTU 2017-18, Marks 05

Answer

i. Global data flow analysis :

1. Global data flow analysis collects the information about the entire program and distributed it to each block in the flow graph.
2. Data flow can be collected in various block by setting up and solving a system of equation.
3. A data flow equation is given as :

$$OUT(s) = \{IN(s) - KILL(s)\} \cup GEN(s)$$

OUT(s) : Definitions that reach exist of block B.

GEN(s) : Definitions within block B that reach the end of B.

IN(s) : Definitions that reaches entry of block B.

KILL(s) : Definitions that never reaches the end of block B.

ii. Loop unrolling : Refer Q. 5.10, Page 5–11C, Unit-5.

iii. Loop fusion or loop jamming : Refer Q. 5.10, Page 5–11C, Unit-5.

Que 5.21. Discuss the role of macros in programming language.

Answer

Role of macros in programming language are :

1. It is use to define word that are used most of the time in program.
2. It automates complex task.
3. It helps to reduce the use of complex statement in a program.
4. It makes the program run faster.

VERY IMPORTANT QUESTIONS

Following questions are very important. These questions may be asked in your SESSIONALS as well as UNIVERSITY EXAMINATION.

Q. 1. What is code generation ? Discuss the design issues of code generation.

Ans. Refer Q. 5.1.

Q. 2. Write an algorithm to partition a sequence of three address statements into basic blocks.

Ans. Refer Q. 5.3.

Q. 3. What is loop ? Explain what constitute a loop in a flow graph.

Ans. Refer Q. 5.5.

Q. 4. Discuss in detail the process of optimization of basic blocks. Give an example.

Ans. Refer Q. 5.6.

Q. 5. Explain what constitute a loop in a flow graph and how will you do loop optimizations in code optimization of a compiler.

Ans. Refer Q. 5.10.

Q. 6. Consider the following three address code segments :

PROD **: = 0**

1. ***I*** **: = 1**
2. ***T*1 : = 4*I**
3. ***T*2 : = *addr*(*A*) – 4**
4. ***T*3 : = *T*2 [*T*1]**
5. ***T*4 : = *addr*(*B*) – 4**
6. ***T*5 : = *T*4[*T*1]**
7. ***T*6 : = *T*3**T*5**
8. ***PROD*** **: = *PROD* + *T*6**
9. ***I*** **: = *I* + 1**
10. **If *I* < = 20 goto (3)**

a. **Find the basic blocks and flow graph of above sequence.**

b. **Optimize the code sequence by applying function preserving transformation optimization technique.**

Ans. Refer Q. 5.11.

Q. 7. Write short notes on the following with the help of example :

i. **Loop unrolling**
ii. **Loop jamming**
iii. **Dominators**
iv. **Viable prefix**

Ans. Refer Q. 5.12.

Q. 8. What is DAG ? What are its advantages in context of optimization ?

Ans. Refer Q. 5.13.

Q. 9. Define a directed acyclic graph. Construct a DAG and write the sequence of instructions for the expression :

$a + a * (b - c) + (b - c) * d$.

Ans. Refer Q. 5.16.

Q. 10. Write short notes (any two) :

i. Global data flow analysis

ii. Loop unrolling

iii. Loop jamming

Ans. Refer Q. 5.20.