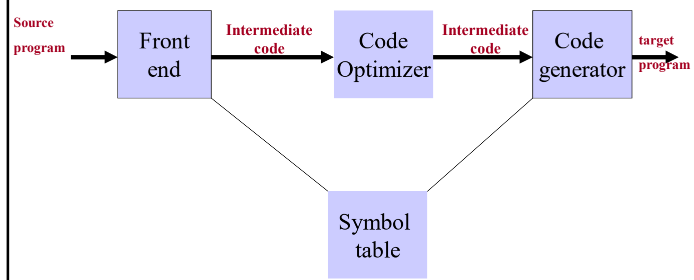
CSI2005
Principles of Compiler Design

MODULE - 7

Dr. WI. Sureshkumar
Associate Professor
School of Computer Science and Engineering (SCOPE)
VIT Vellore
wi.sureshkumar@vit.ac.in
wi.sSJT413A34

Introduction



Position of code generator

Issues in the Design of a Code Generator

- Details depend on
  - Target language
  - Operating System
- But following issues are inherent in all code generation problems
  - Memory management
  - Instruction Selection
  - Register allocation and
  - Evaluation order

Input to the Code Generator

- We assume, front end has
  - Scanned, parsed and translate the source program into a reasonably detailed intermediate representations
  - Type checking, type conversion and obvious semantic errors have already been detected
  - Symbol table is able to provide run-time address of the data objects
  - Intermediate representations may be
    - Postfix notations
    - Three address representations
    - Stack machine code
    - Syntax tree
    - DAG

Target Programs

- The output of the code generator is the target program.
- Target program may be
  - Absolute machine language
    - It can be placed in a fixed location of memory and immediately executed
  - Re-locatable machine language
    - Subprograms to be compiled separately
    - A set of re-locatable object modules can be linked together and loaded for execution by a linker
  - Assembly language
    - Easier

6/11/2021 5

Instruction Selection

- The nature of the instruction set of the target machine determines the difficulty of the instruction selection.
- Uniformity and completeness of the instruction set are important factors
- Instruction speeds is also important
  - Say, x = y + z

```
Mov y, R0     /*load y into register R0 */
Add z, R0     /* add z to R0 */
Mov R0, x     /* Store R0 into x */
```

Statement by statement code generation often produces poor code

6/11/2021 6

Instruction Selection (2)

```
a = b + c
d = a + e
```

```
MOV b, R0
ADD c, R0
MOV R0, a
MOV a, R0
ADD e, R0
MOV R0, d
```

If a is subsequently used

6/11/2021 7

Instruction Selection (3)

- The quality of the generated code is determined by its speed and size.
- Cost difference between the different implementation may be significant.
  - Say a = a + 1

```
Mov a, R0
Add #1, R0
Mov R0, a
```

  - If the target machine has increment instruction (INC), we can write

inc a

6/11/2021 8

Register Allocation

- Instructions involving register operands are usually shorter and faster than those involving operands in memory.
- Efficient utilization of register is particularly important in code generation.
- The use of register is subdivided into two sub problems
  - During register allocation, we select the set of variables that will reside in register at a point in the program.
  - During a subsequent register allocation phase, we pick the specific register that a variable will reside in.

Register Allocation-Example

t = a + b
t = t * c
t = t / d

The optimal machine code

```
Load   R0, a
Add    R0, b
Mul    R0, c
Div    R0,d
Store  R0,t
```


Basic Blocks and Flow Graphs

- A graph representation of three address statements, called flow graph.
- Nodes in the flow graph represent computations
- Edges represent the flow of control

Basic Block:

A basic block is a sequence of consecutive statements in which flow of control enters at the beginning and leaves at the end without halt or possibly of the branching except at the end.

Basic Blocks and Flow Graphs (2)

This is a basic block

$t_1 = a*a$
$t_2 = a*b$
$t_3 = 2*t_2$
$t_4 = t_1+t_3$
$t_5 = b*b$
$t_6 = t_4+ t_5$

Three address statement x = y + z is said to define x and to use y and z.

A name in a basic block is said to be live at a given point if its value is used after that point in the program, perhaps in another basic block

Basic Blocks and Flow Graphs (3)

❑ Partition into basic blocks

- **Method**
  - We first determine the leader
    - The first statement is a leader
    - Any statement that is the target of a conditional or unconditional goto is a leader
    - Any statement that immediately follows a goto or unconditional goto statement is a leader
  - For each leader, its basic block consists of the leader and all the statements up to but not including the next leader or the end of the program.

Basic Blocks and Flow Graphs (4)

(1) prod = 0
(2) i = 1

B_1

(3) t_1=4*I

----
(11) I = t_7
(12) If I <= 20 goto (3)

B_2

Transformation on Basic Block

- ❑ A basic block computes a set of expressions.
- ❑ Transformations are useful for improving the quality of code.
- ❑ Two important classes of local optimizations that can be applied to a basic block
  - Structure Preserving Transformations
  - Algebraic Transformations

Structure Preserving Transformations

❑ <u>Common sub-expression elimination</u>

a = b + c	a = b + c
b = a – d	b = a – d
c = b + c	c = b + c
d = a - d	d = b

same

b is redefined

Structure Preserving Transformations

- **Dead – Code Elemination**

  Say, x is dead, that is never subsequently used, at the point where the statement $x = y + z$ appears in a block.

  We can safely remove x

- **Renaming Temporary Variables**
  - say, t = b+c where t is a temporary var.
  - If we change u = b+c, then change all instances of t to u.
- **Interchange of Statements**
  - $t_1 = b + c$
  - $t_2 = x + y$
  - We can interchange iff neither x nor y is t_1 and neither b nor c is t_2

Algebraic Transformations

- Replace expensive expressions by cheaper one
  - X = X + 0 eliminate
  - X = X * 1 eliminate
  - X = y**2 (why expensive? Answer: Normally implemented by function call)
    - by X = y * y
- **Flow graph:**
  - We can add flow of control information to the set of basic blocks making up a program by constructing directed graph called flow graph.
  - There is a directed edge from block B_1 to block B_2 if
    - There is conditional or unconditional jump from the last statement of B_1 to the first statement of B_2 or
    - B_2 is immediately follows B_1 in the order of the program, and B_1 does not end in an unconditional jump.

Loops

- A loop is a collection of nodes in a flow graph such that
  - All nodes in the collection are strongly connected, that is from any node in the loop to any other, there is a path of length one or more, wholly within the loop, and
  - The collection of nodes has a unique entry, that is, a node in the loop such that, the only way to reach a node from a node out side the loop is to first go through the entry.

Exercise

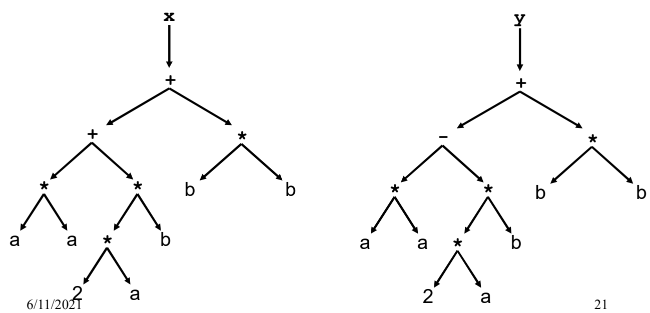
- given the code fragment

```
x := a*a + 2*a*b + b*b;
y := a*a – 2*a*b + b*b;
```

draw the dependency graph before and after common subexpression elimination.

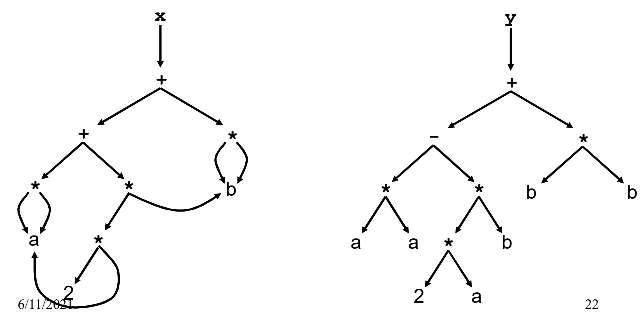
Answers

dependency graph before CSE



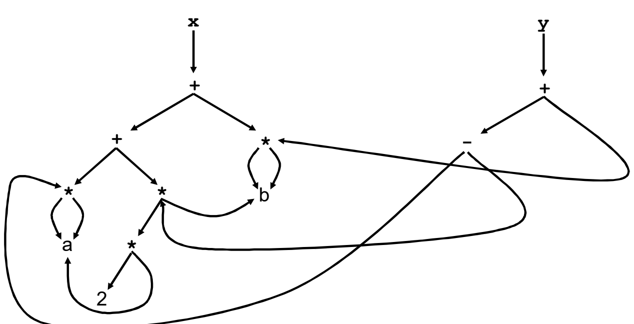
Answers

dependency graph after CSE



Answers

dependency graph after CSE



Conditional statements

if x < y goto z
CMP x , y
CJ< z { if x-y < 0 then goto z}

x = y + z
If x< 0 goto z

MOV y, R0
ADD z, R0
MOV R0, x
CJ< z {jump to z if conditional code is negative}