CD: UNIT-5 CODE GENERATION

FALL SEMESTER, YEAR (V/VI, 3rd)

Published: August, 2022

FALL SESSION: 2022-23

Shweta Tiwari

UNIT-5 CD COMPILER DESIGN (KIT-052)

UNIT V – CODE GENERATION

PREPARED FOR
Engineering Students
All Engineering College

PREPARED BY
SHWETA TIWARI

Guest Faculty

Rajkiya Engineering College | Ambedkar Nagar, UP, India

Faculty Name: Miss. Shweta Tiwari, Subject: CD-COMPILER DESIGN (KIT-052), Year- 3rd Year, Semester- 5th/6th Sem, Branch- CS and IT, Session- Odd/Even Semester (2022-23)

CD-COMPILER DESIGN (KIT-052)

This **(Question Bank Part-1 Easy to Advanced Level (Belong your Syllabus))** only attempts to discover some questions with proper explanation that can be generated in "CD-COMPILER DESIGN" with the answer to all these questions. There can be some errors to these answers. If you find any errors then please do write to us.

CD-COMPILER DESIGN (KIT-052)

UNIT V CODE GENERATION -

(CODE OPTIMISATION AND CODE GENERATION)

1. **Define code generations with ex?**
   It is the final phase in the compiler model and it takes as an input an intermediate representation of the source program and output produced as equivalent target programs. Then intermediate instructions are each translated into a sequence of machine instructions that perform the same task.

2. **What are the issues in the design of code generators?**

- Input to the generator Target programs Memory management Instruction selection
- Register allocation
- Choice of evaluation order Approaches to code generation.

3. **Give the variety of forms in the target program**.

- Absolute machine language.
- Relocatable machine language.
- Assembly language.

4. **Give the factors of instruction selections.**

- Uniformity and completeness of the instruction sets Instruction speed and machine idioms
- Size of the instruction sets.

5. **What are the sub problems in register allocation strategies?**

- During register allocation, we select the set of variables that will reside in register at a point in the program.
- During a subsequent register assignment phase, we pick the specific register that a variable resides in.

6. **Give the standard storage allocation strategies**.

- Static allocation
- Stack allocation.

7. **Define static allocations and stack allocations**

   Static allocation is defined as laid out for all data objects at compile time. Names are bound to storage as a program is compiled, so there is no need for a Run time support package.

Stack allocation is defined as a process which manages the run time as a Stack. It is based on the idea of a control stack; storage is organized as a stack, And activation records are pushed and popped as activations begin and end.

8. Write the addressing mode and associated costs in the target machine.

MODE		ADDRESS	ADDED COST
Absolute	M	M	1
Register	R	R	0
Indexed	c(R)	c+contents(R)	1
Indirect register	*	contents(R)	0
Indirect indexed		contents(c+contents(R))	1

9.Define basic block and flow graph.

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 possibility Of branching except at the end.
A flow graph is defined as the adding of flow of control information to the Set of basic blocks making up a program by constructing a directed graph.

10. Write the step to partition a sequence of 3 address statements into basic blocks.

1. First determine the set of leaders, the first statement of basic blocks. The rules we can use
- are the following.
- 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 conditional goto statement is a leader.

2. For each leader, its basic blocks consist of the leader and all statements Up to but not including the next leader or the end of the program.

11. Give the important classes of local transformations on basic blocks

- Structure preserving transformations Algebraic transformations.
-

12.Describe algebraic transformations.

It can be used to change the set of expressions computed by a basic block into A algebraically equivalent sets. The useful ones are those that simplify the Expressions place expensive operations by cheaper ones.

X = X+ 0 X = X * 1

13.What is meant by register descriptors and address descriptors?

A register descriptor keeps track of what is currently in each register. It is Consulted whenever a new register is needed. An address descriptor keeps track of the location wherever the current Value of the name can be found at run time. The location might be a register, a Stack location, a memory address,

14.What are the actions to perform the code generation algorithms?

- Invoke a function get reg to determine the location L.
- Consult the address descriptor for y to determine y‟, the current location of y.
- If the current values of y and/or z have no next uses, are not live on exit from the block, and are in register, alter the register descriptor.

15.Write the code sequence for the d:=(a-b)+(a-c)+(a-c).

Statement	Code generation	Register descriptor	Address
			descriptor
t:=a-b	MOV a,R0 SUB b,R	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 con	u in R1 v in R
d:=v+u	ADD R1,R0 MOV R	R0 contains d	d in R0 d in R0 and m

16.Write the labels on nodes in DAG.

A DAG for a basic block is a directed acyclic graph with the following Labels on 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.

17.Give the applications of DAG.

- Automatically detect the common sub expressions
- Determine which identifiers have their values used in the block.
- Determine which statements compute values that could be used outside the blocks.

18. **Define Peephole optimization**.

A Statement by statement code generation strategy often produces target code that contains redundant instructions and suboptimal constructs. "Optimizing" is misleading because there is no guarantee that the resulting code is optimal. It is a method for trying 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.

19.Write the characteristics of peephole optimization?

- Redundant-instruction elimination Flow-of-control optimizations.
- Algebraic simplifications Use of machine idioms
-

20. What are the structures preserving transformations on basic blocks?

- Common subexpression elimination Dead-code elimination
- Renaming of temporary variables
- Interchange of two independent adjacent statement
-

21. Define Common subexpression elimination with ex.

It is defined as the process in which the statements which have the Same expressions. Hence this basic block may be transformed into the equivalent Block.

Ex:

a : =b + c b

:=a - d

c :=b + c

After elimination:

a : =b + c b

:=a - d c :=a

22. Define Dead-code elimination with ex.

It is defined as the process in which the statement x=y+z appears in a basic block, where x is a dead object that is never subsequently used. Then this statement may be safely removed without changing the value of basic blocks.

23. Define Renaming of temporary variables with ex.

We have the statement u:=b + c ,where u is a new temporary variable, and change all uses of this instance of t to u, then the value of the basic block is not changed.

24. Define reduction in strength with ex.

Reduction in strength replaces expensive operations by equivalent cheaper ones on the target machines. Certain machine instructions are cheaper than others and can often be used as special cases of more expensive operators. Ex:

X^2 is invariably cheaper to implement as x*x than as a call to an exponentiation routine.

25. Define use of machine idioms.

The target machine may have harder instructions to implement certain specific operations efficiently. Detecting situations that permit the use of these instructions can reduce execution time significantly.

26. Define code optimization and optimizing compiler

The **term code-optimization** refers to techniques a compiler can employ in an attempt to produce a better object language program than the most obvious for a given source program.

Compilers that apply code-improving transformations are called **Optimizing-compilers.**

PART B:

1. What are the issues in the design of code generators? Explain in detail.
2. Discuss about the run time storage management.
3. Explain basic blocks and flow graphs.
4. Explain about transformation on a basic block.
5. Write a code generation algorithm. Explain about the descriptor and function getreg().Give an example.
6. Explain peephole optimization
7. Explain DAG representation of basic blocks.
8. Explain principle sources of code optimization in detail.
9. Explain the Source language issues with details.
10. Explain the Storage organization strategies with examples.
11. Explain storage allocation strategy.
12. Explain about Parameter passing.
13. Explain the non local names in runtime storage managements.
14. Explain about activation records and its purpose.
15. Explain about Optimization of basic blocks.
16. Explain the various approaches to compiler development.
17. Explain simple code generators with suitable examples.
18. Discuss about the following:

    a) Copy Propagation b) Dead-code Elimination and c) Code motion

************************************THE END***************************