Concepts Introduced in Chapter 8

- register assignment
- instruction selection
- run-time stack management
- code-generator generators

Target Program

The target program can be

- Absolute machine language
- Relocatable machine language
- Assembly language

Tasks Performed by a Code Generator

- Register assignment
- Instruction selection
- Management of the run-time stack

Register Assignment

- Register assignment is the assignment of temporaries to hardware registers.
- Challenges
  - Use of register pairs
  - Overlapping of registers
  - Operations in specific registers
  - Spills

Instruction Selection

- Instruction selection is the mapping of the intermediate language operations to machine instructions.
- Have to choose not only the appropriate instructions, but also the addressing mode of each operand.

Implementation of Conditional Branches

- Use a comparison instruction to set condition codes.
- Use condition codes that reflect the result of the last arithmetic operation.
- Use a comparison instruction to set a register to zero or a nonzero value based on two values and a relational operator.
- Use a comparison instruction to set a predicate register, which contains only a single bit.
- Compare and branch in a single instruction.

SPARC Addressing Modes

Name	Assembly	RTL
immediate	`n`	`n`
register	`%n`	`r[n]`
register deferred	`[%n]`	`M[r[n]]`
displacement	`[%n+m]`	`M[r[n]+m]`
	`[%n-m]`	`M[r[n]-m]`
indexed	`[%n+%m]`	`M[r[n]+r[m]]`

Run-Time Stack Management

- Allocating and deallocating space on the run-time stack when entering and leaving a function.
- Partitioning the register set into scratch and nonscratch sets.
- Saving and restoring nonscratch registers that are used in the function.
- Dedicating registers for managing the run-time stack.
- Passing arguments.
- Responsibility of callee versus caller.

Evaluation Order of Arguments

- What is printed by the following program?

```
int g = 0;
int f0() { return g; }
int f1() { return ++g;}
main()
{
   printf("%d %d\n", f0(), f1());
}
```

Code-Generator Generators

- Generates a code generator from a specification.
- Types of pattern matching
  - Tree rewriting
  - Parsing

Code Generation by Tree Rewriting

- Each rule is of the form

  replacement ← template { action }

  where replacement is a node

  template is a subtree

  action is a code fragment
- Accomplished by a depth-first translation.

Code Generation by Parsing

- Use an LR parser to accomplish the pattern matching.
- Input tree is represented as a string.