Building a Runnable Program

1. What is a code generator? Why might it be useful?

2. What is a basic block? A control flow graph?

3. What are virtual registers? What purpose do they serve?

4. What is the difference between local and global code improvement?

5. What is register spilling?

6. Explain what is meant by the ”level” of an intermediate form (IF). What are the comparative advantages and disadvantages of high-, medium-, and low-level IFs?

7. What is the IF most commonly used in Ada compilers?

8. Name two advantages of a stack-based IF. Name one disadvantage.

9. Explain the rationale for basing a family of compilers (several languages, several target machines) on a single IF.

10. Why might a compiler employ more than one IF?

11. Outline some of the major design alternatives for back-end compilers organizaion and structure.

12. What is sometimes called the ”middle end” of a compiler?

13. Why is management of a limited set of physical registers usually deferred until late in the compilation process?

14. What are the distinguishing characteristics of a relocatable object file? An executable object file?

15. Why do operating systems typically zero-fill pages used for unitinitalized data?

16. List four tasks commonly performed by an assembler.

17. Summarize the comparative advantages of assembly languages and object code as the output of a compiler.

18. Give three examples of pseudoinstructions and three examples of directives that an assembler might be likely to provide.

19. Why might an assembler perform its own final pass of instruction scheduling?

20. Explain the distinction between absolute and relocatable words in an object file. Why is the notion of ”relocatability” more complicated than it used to be?

21. What is the difference between linking and loading?

22. What are the principal tasks of a linker?

23. How can a linker enforce type checking across compilation units?

24. What is the motivation for dynamic linking?

25. Characterize GIMPLE, RTL, Java bytecode, and Common Intermediate Language as high-, medium, or low-level intermediate forms.

26. Nae three languages (other than C) for which there exist `gcc` front ends.

27. What is the internal IF of `gcc`'s front ends?

28. Give brief descriptions of GIMPLE and RTL. How do they differ? Why was GIMPLE introduced?

29. Explain the addressing challenge faced by dynamic linking systems.

30. What is position-independent code? What is it good for? What special precautions must a compiler follow in order to produce it?

31. Explain the need for PC-relative addressing in position-independent code. How is it accomplished on the x86-32?

32. What is the purpose of a linkage table?

33. What is lazy dynamic linking? What is its purpose? How does it work?