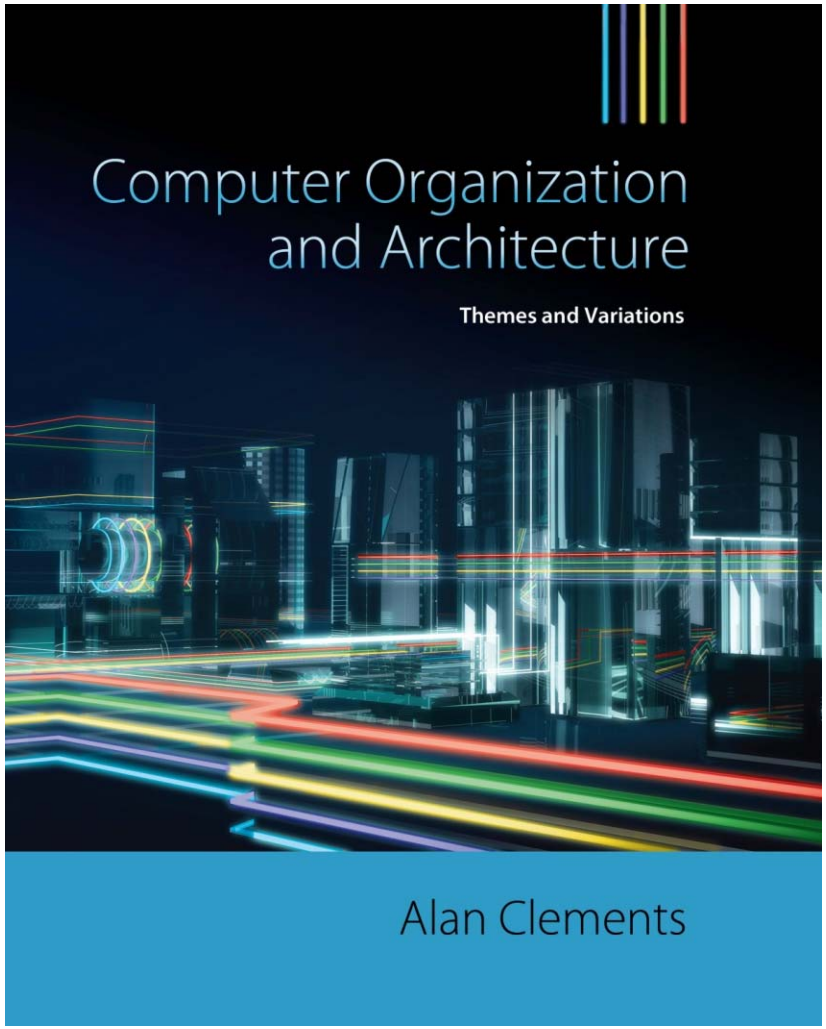
CHAPTER 4

Computer Organization and Architecture



CENGAGE Learning

ISAs Breadth and Depth

- This chapter extends the overview of ISAs in both breadth and depth.
  - *Yet, we will only cover the depth part in lectures this term*
- In particular, we will look at the role of the stack and architectural support for subroutines and parameter passing.

The Stack and Data Storage

- Let's begin by looking at some background issues concerning *data storage*, *procedures*, and *parameter passing*.
- ***Computer programs*** consist of
  - ***data elements*** and
  - ***procedures*** which operate on these ***data elements***
- High-level language programmers use variables to represent *data elements*
- Variables are declared by:
  - *assigning* names to them and by *reserving* storage for them.
- *Reserving* memory storage for variables can be performed
  - at compilation time (***static memory allocation***)
  - at runtime (***dynamic memory allocation***)
- *Statically allocated variables* will have *static addresses* which *will not* be changed during execution
- *Dynamically allocated variables* will have *dynamic addresses* which *will* be changed during execution, as they will be allocated at runtime

The Stack and Data Storage

- Procedures often require *local workspace* for their temporary variables.
- The term *local* means that the workspace is private to the procedure and is never accessed by the calling program or by other subroutines.
- If a procedure is to be made re-entrant or to be used recursively, its local variables must be bounded up not only with the procedure itself, but with the occasion of its use.
  - Each time the procedure is called, a new workspace must be assigned to it.

The Stack and Data Storage

- A variable has a *scope* associated with it.
  - The scope of a variable defines the range of its *visibility* or *accessibility* within a program.
    - ***Global*** variables are *visible* (accessible) from the moment they are loaded into memory to the moment when the program stops running ***(static memory allocation)***
    - ***Variables*** declared within a procedure, as well as ***parameters*** are *visible* (accessible) *within* that procedure but *invisible* (inaccessible) *outside* the procedure ***(dynamic memory allocation)***

- Here, we are interested to learn more about ***dynamic memory allocation***

The Stack and Data Storage

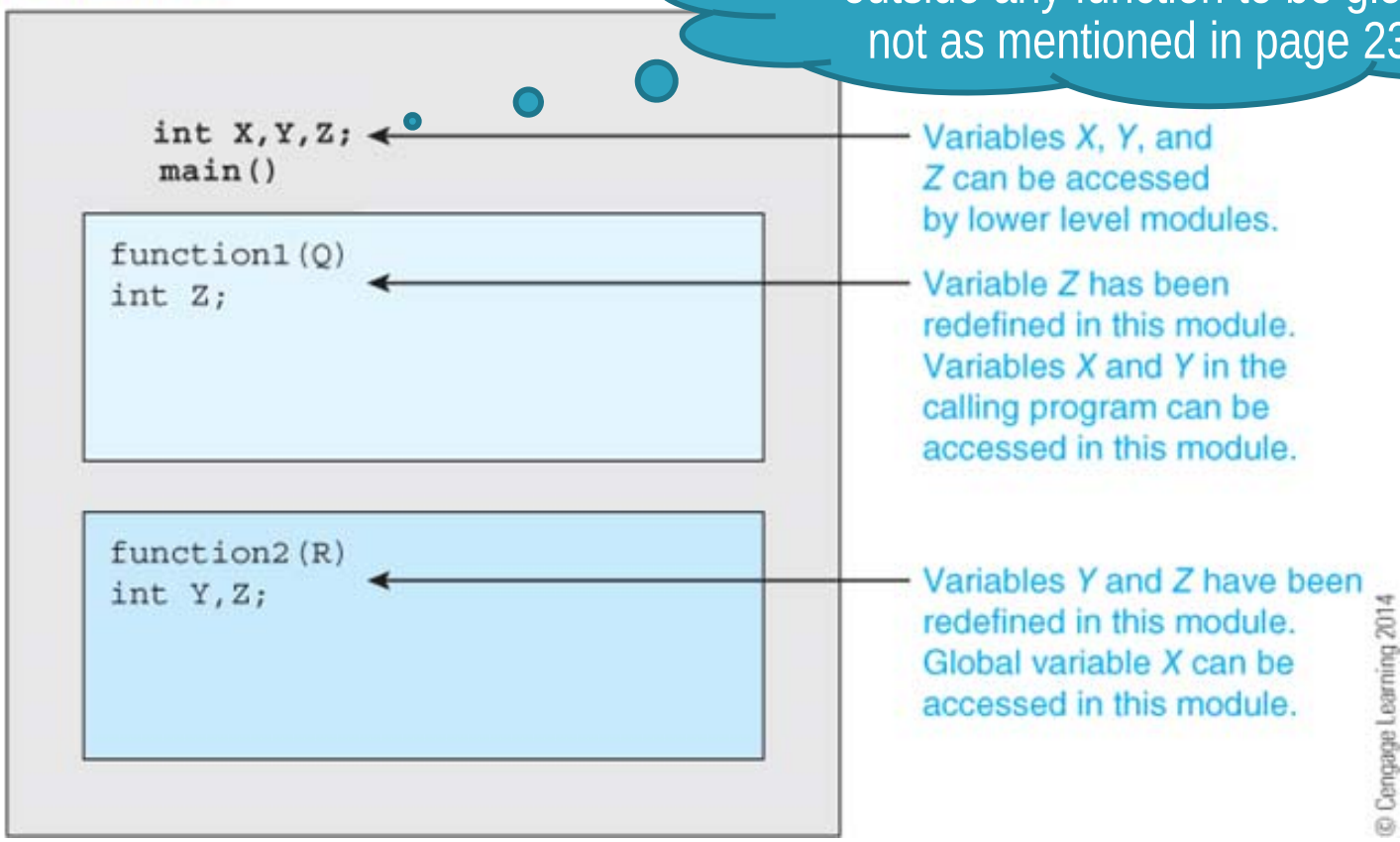
- Figure 4.1 illustrates the scope of variables

The *duration* of local variables and parameters are "*automatically*"
- *allocated* when the enclosing *function is called*

and
- *deallocated* when the *function returns*

The `int X,Y,Z;` should be outside any function to be global, not as mentioned in page 231.

FIGURE 4.1 The concept of scope



Storage and the Stack

- When a language invokes a procedure, it is said to *activate* the procedure.
- Associated with each invocation (activation) of a procedure, there is an ***activation record*** containing all the information necessary to execute the procedure, including
  - parameters,
  - local variables, and
  - return address,