What is ARM architecture?

- The *ARM* architecture is the intellectual property of *ARM Holdings*, based in Cambridge, England.
- The company was *founded in 1990* as *Advanced RISC Machines* (*ARM*) by
  - Acorn Computers,
  - Apple Computers, and
  - VLSI Technology.
- The *1st-generation* of *ARM* was *8-bit* microprocessors.
- The *2nd-generation* of *ARM* was *32-bit* microprocessors.
  - In *ARM* terminology, *16 bits is a half-word*, and *32 bits is a word*.
- *There has been **remarkably little change** in **instruction set architecture** between today's high performance machines and 1st-generation microprocessors.*

- Unlike other microprocessor manufactures, e.g., Intel, AMD, and Freescale, *ARM* does ***NOT** build chips*, but ***licenses to semiconductor companies*** its core processors for use in *systems on chips* and *microcontrollers*.
- *ARM* successfully targeted the world of mobile devices, e.g., netbooks, tablets, and cell phones.

- *ARM* is a machine with register-to-register architecture, as well as load/store instructions that move data between memory and registers.
- *ARM **operand values*** are *32 bits wide*, except for several multiplication instructions that generate a 64-bit product stored in two 32-bit registers.

ARM Register Set

- The *ARM* processor has
  - 16 *32-bit* registers (r0, r1, r2, …, r12, r13, r14, r15)
    - r0, r1, r2, …, r12 are general-purpose registers
    - r15 is the program counter
    - r14 is the link register—holds subroutine return addresses
    - r13 is *reserved* for ***use by the programmer*** as the stack pointer

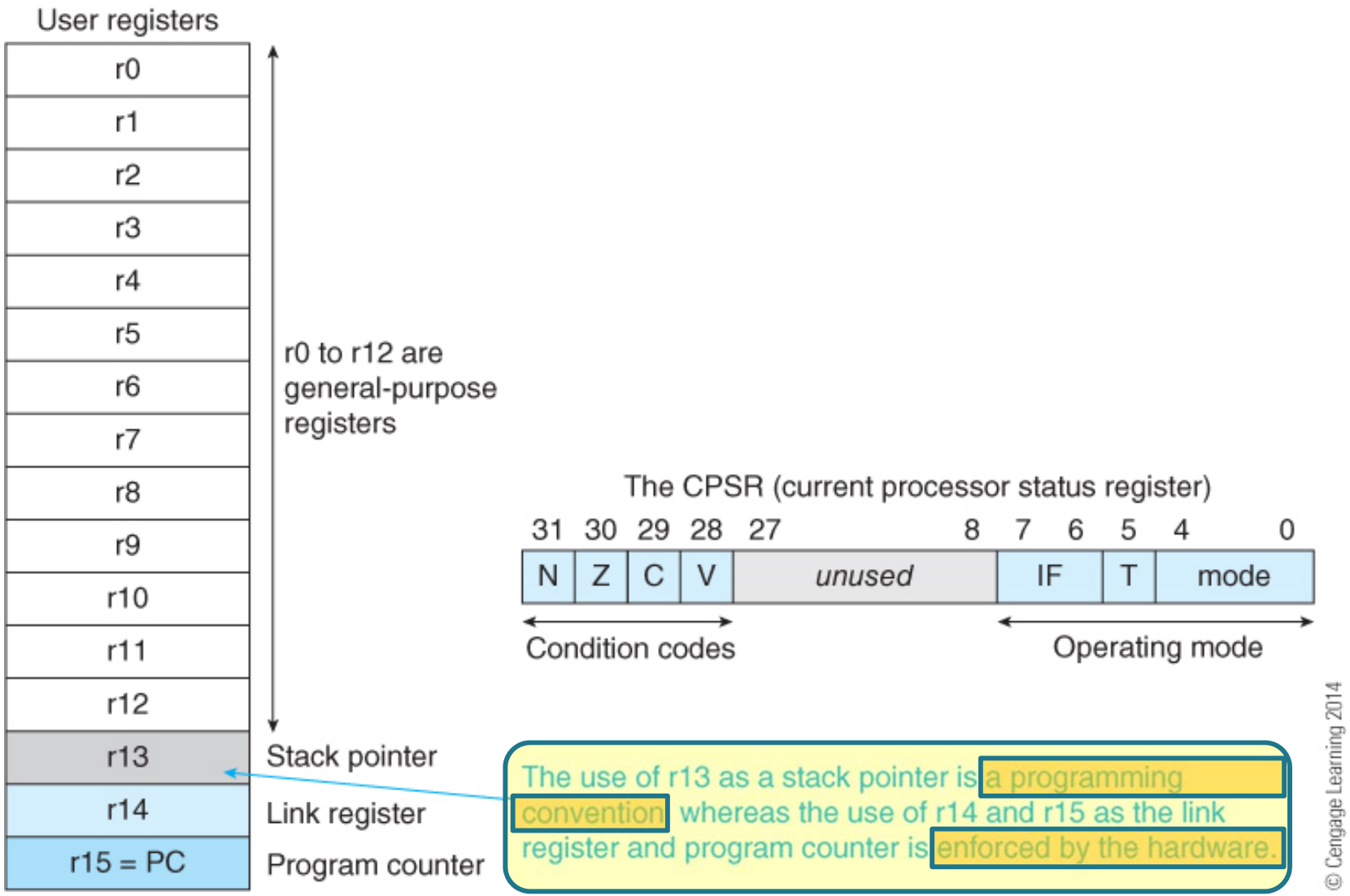
- Sixteen registers require a 4-bit address
  - saves three bits per instruction over RISC processors with 32-register architectures (5-bit address).

- The *ARM*'s *current program status register* (***CPSR***) contains
  - Condition codes (bits number 31, 30, 29, and 28)
    N (negative), Z (zero), C (carry) and V (overflow) flag bits
  - Operating mode (bits number 0–7)
    Will talk about them later

- *ARM* processors have a rich instruction set

ARM Register Set

FIGURE 3.12 ARM register set



Typical ARM Instructions

TABLE 3.1 ARM Data Processing, Data Transfer, and Compare Instructions

Instruction	ARM Mnemonic	Definition
Addition	`ADD r0,r1,r2`	[r0] ← [r1] + [r2]
Subtraction	`SUB r0,r1,r2`	[r0] ← [r1] - [r2]
AND	`AND r0,r1,r2`	[r0] ← [r1] · [r2]
OR	`ORR r0,r1,r2`	[r0] ← [r1] + [r2]
Exclusive OR	`EOR r0,r1,r2`	[r0] ← [r1] ⊕ [r2]
Multiply	`MUL r0,r1,r2`	[r0] ← [r1] x [r2]
Register-to-register move	`MOV r0,r1`	[r0] ← [r1]
Compare	`CMP r1,r2`	[r1] - [r2]
Branch on zero to label	`BEQ label`	[PC] ← label (jump to label)

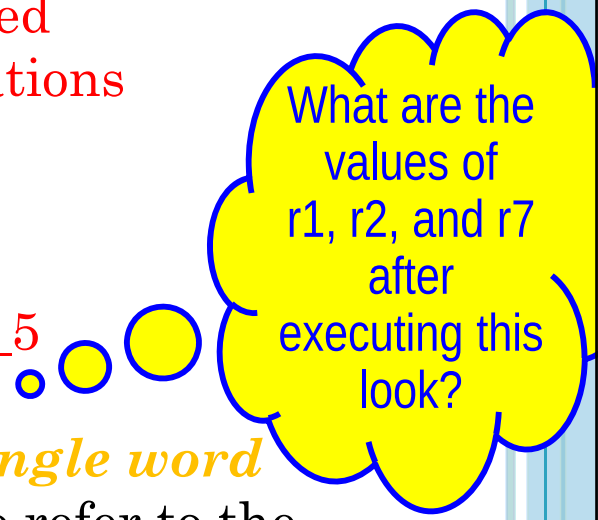
ARM Assembly Language

- ARM instructions are written in the form

  {Label} Op-code **operand1**, operand2, operand3 {;comment}

- Consider the following example of a loop.

```
        MOV   r1,#0       ;initialize the total
        MOV   r2,#10      ;initialize the value to be added
        MOV   r7,#20      ;initialize the number of iterations
Test_5  ADD   r1,r1,r2    ;increment total by the value
        SUBS  r7,#1       ;decrement loop counter
                          ;same as SUBS r7, r7, #1
        BNE   Test_5      ;IF not zero THEN goto Test_5
```



- The Label field is a user-defined label (***case-sensitive single word without space***) that can be used by other instructions to refer to the address of that line.

- Any text following a semicolon is regarded as a ***comment*** field which is ignored by the assembler.

ARM Assembly Language

- Suppose we wish to generate the sum of the cubes of numbers from 1 to 10. We can use the *multiply and accumulate* instruction;

```
        MOV   r0,#0          ;clear total in r0
        MOV   r1,#10         ;FOR i = 10 to 1 (count down)
Next    MUL   r2,r1,r1       ;  square the number (i × i)
        MLA   r0,r2,r1,r0    ;  cube the number and add it to total
        SUBS  r1,r1,#1       ;  decrement counter (set condition flags)
        BNE   Next           ;END FOR (branch back on count not zero)
```

- This fragment of assembly language is *syntactically* correct.
  - But, it is not yet a program that we can run.
- We have to specify the environment to make it a standalone program.
- There are two types of statement:
  - ***executable instructions*** that are executed by the computer and
  - ***assembler directives*** that tell the assembler something about the environment.

Structure of an ARM Program

```
        AREA Cubes, CODE, READONLY
        ENTRY

        MOV   r0,#0          ;clear total in r0
        MOV   r1,#10         ;FOR i = 10 to 1
Next    MUL   r2,r1,r1       ;  square number
        MLA   r0,r2,r1,r0    ;  cube number and add to total
        SUBS  r1,r1,#1       ;  decrement loop count
        BNE   Next           ;END FOR

        END
```

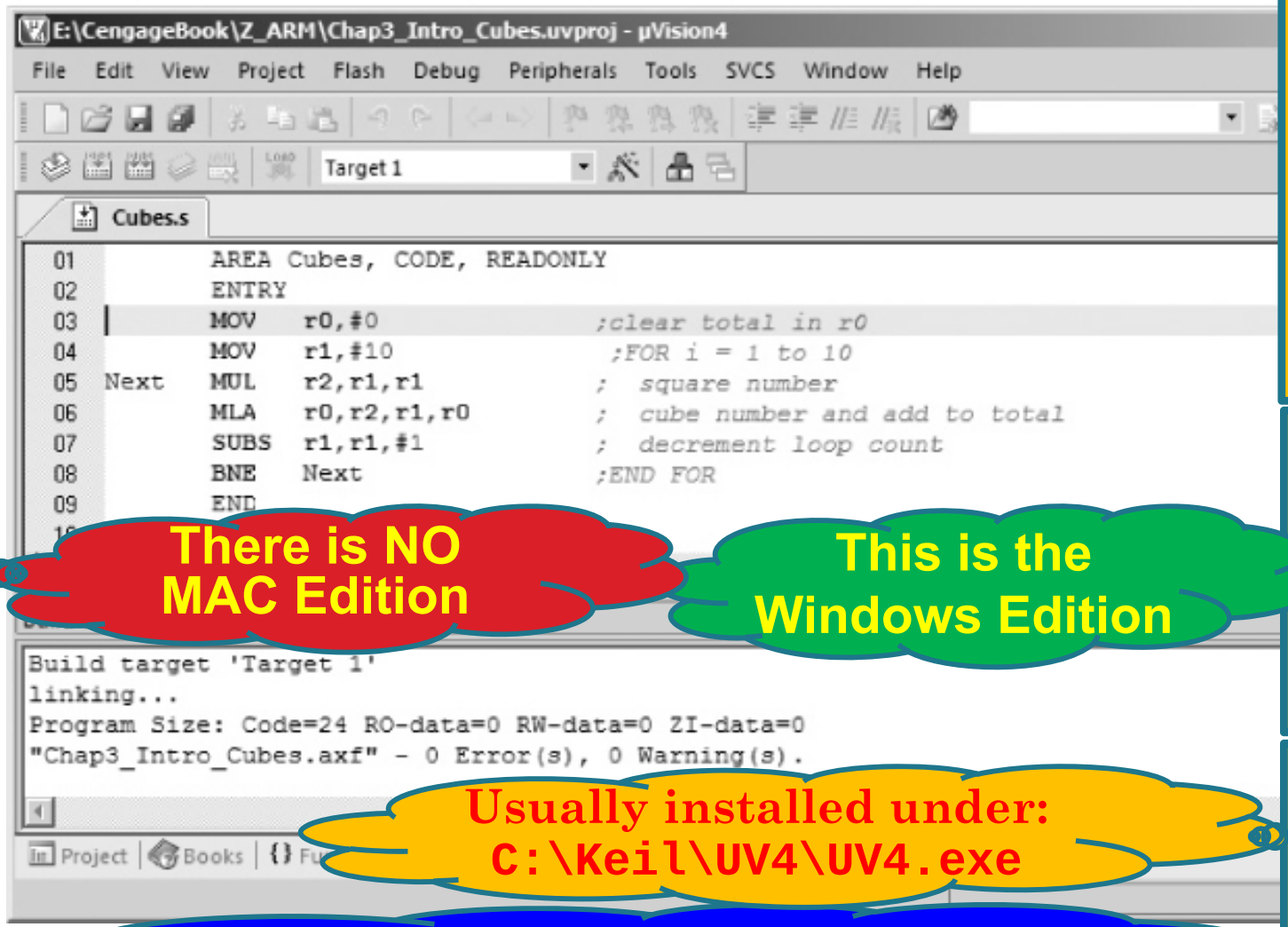
assembler directive

Assembly code

assembler directive

Snapshot of the Display of an ARM Development System

FIGURE 3.13 Assembling an assembly language program using Kiel's ARM IDE



There is NO MAC Edition

This is the Windows Edition

Usually installed under: C:\Keil\UV4\UV4.exe

This is MicorVision 4, not 5.

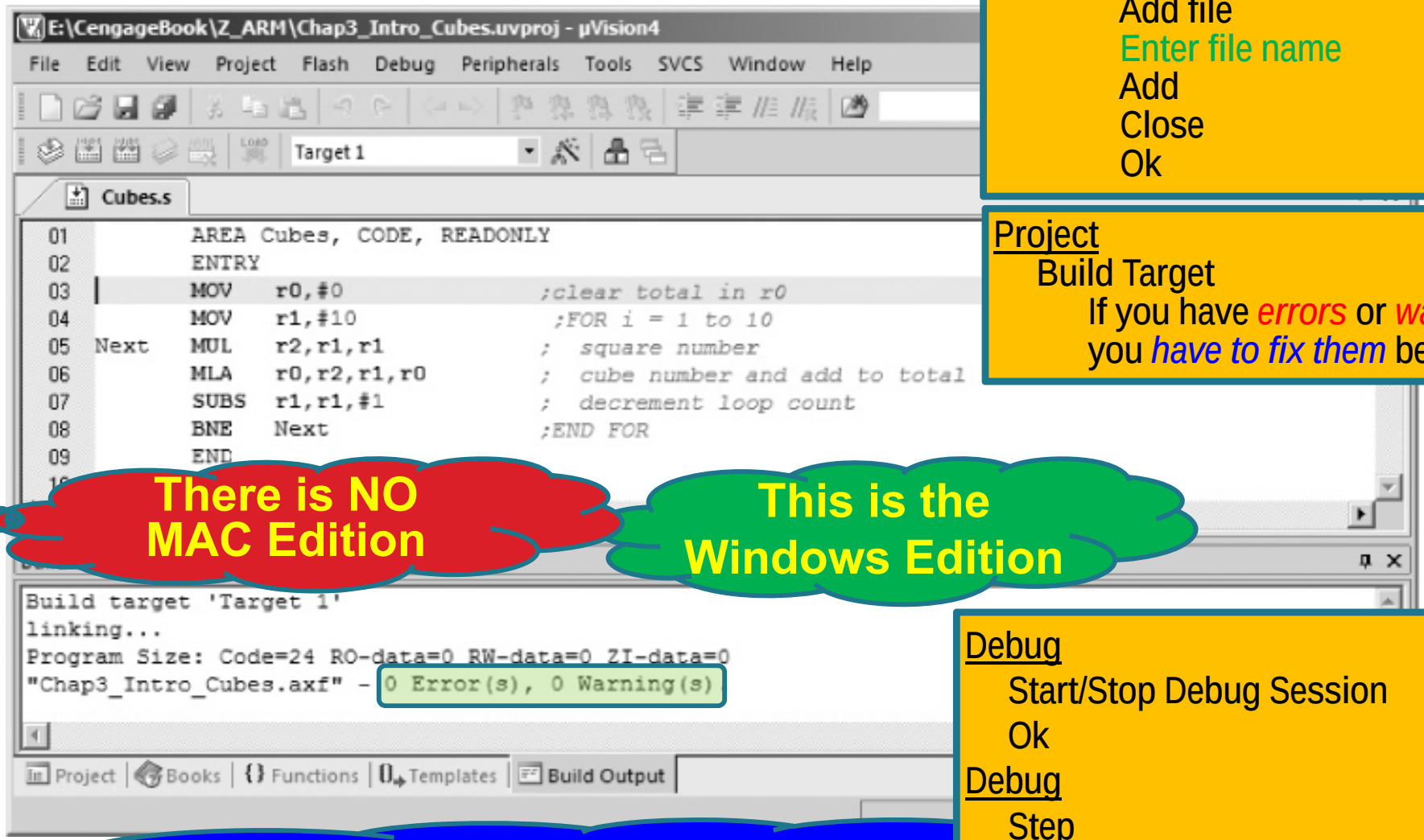
Project
- New µVision Project
  - Enter file name
  - Save
  - Select device for Target
    - ARM
      - ARM7 (Big Endian)
      - Ok

File
- New
  - Enter assembly program (i.e., code and assembler directives)

File
- Save
  - Enter file name (to simplify things, use .s as an extension
  - Save

Snapshot of the Display of an ARM Development System

FIGURE 3.13 Assembling an assembly language program



Project
Manage
Components, Environment, books
Add file
Enter file name
Add
Close
Ok

Project
Build Target
If you have *errors* or *warnings*,
you *have to fix them* before continue.

There is NO MAC Edition

This is the Windows Edition

Debug
Start/Stop Debug Session
Ok

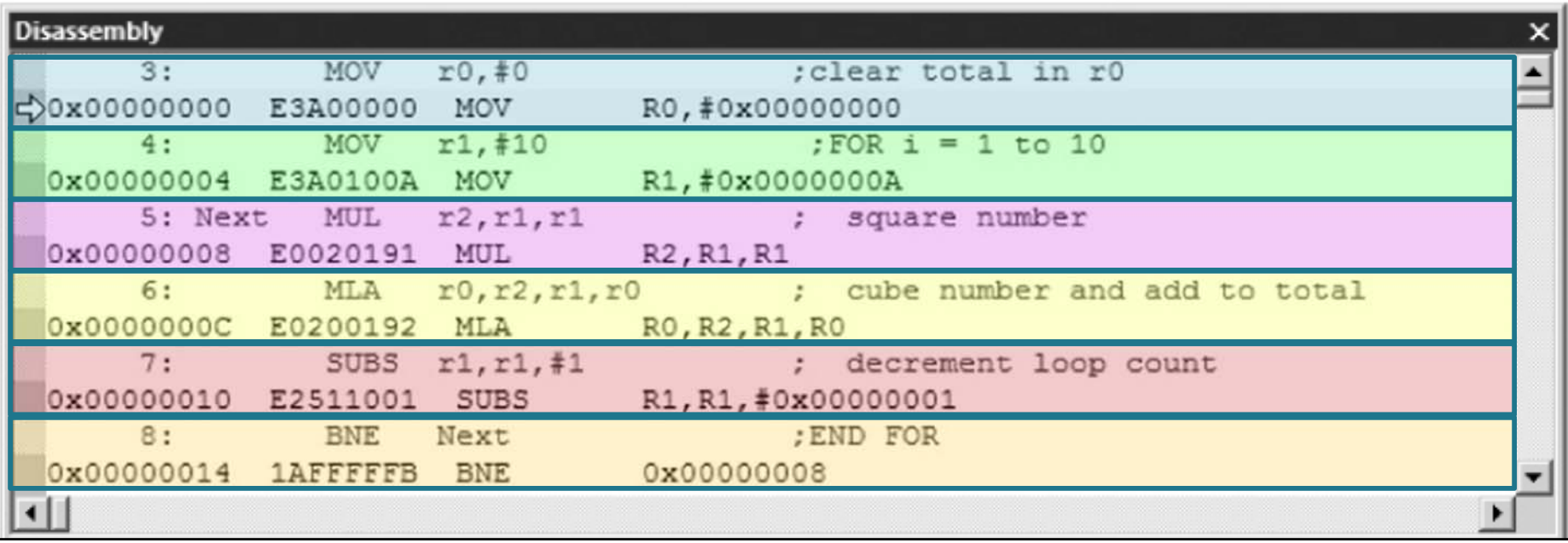
Debug
Step

This is MicorVision 4, not 5.

Snapshot of the Display of an ARM Development System

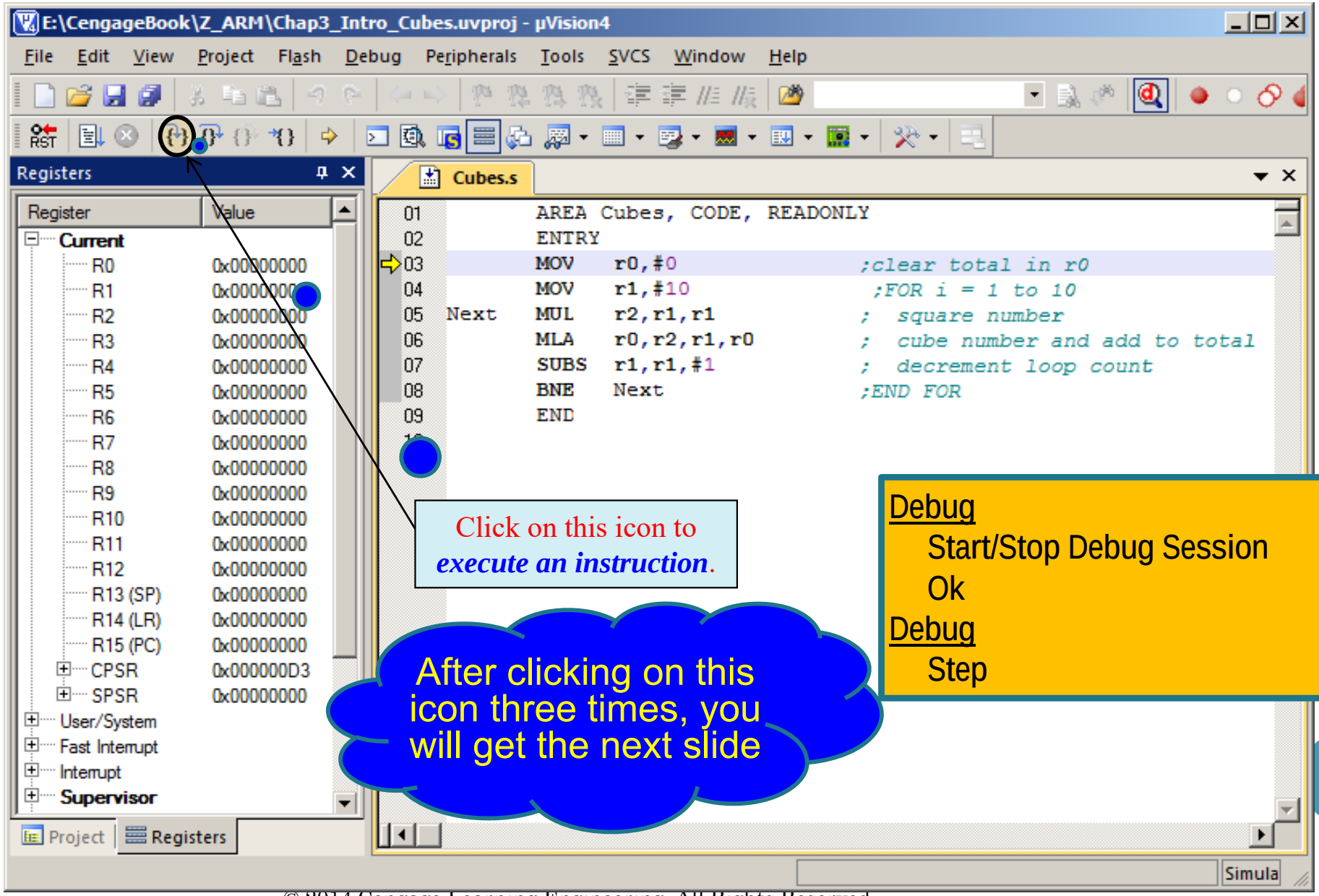
- This is the Disassembly Window that shows memory contents as both
  - hexadecimal values (machine language) and
  - assembly code.

FIGURE 3.14 The disassembly window with the hexadecimal code generated by the program



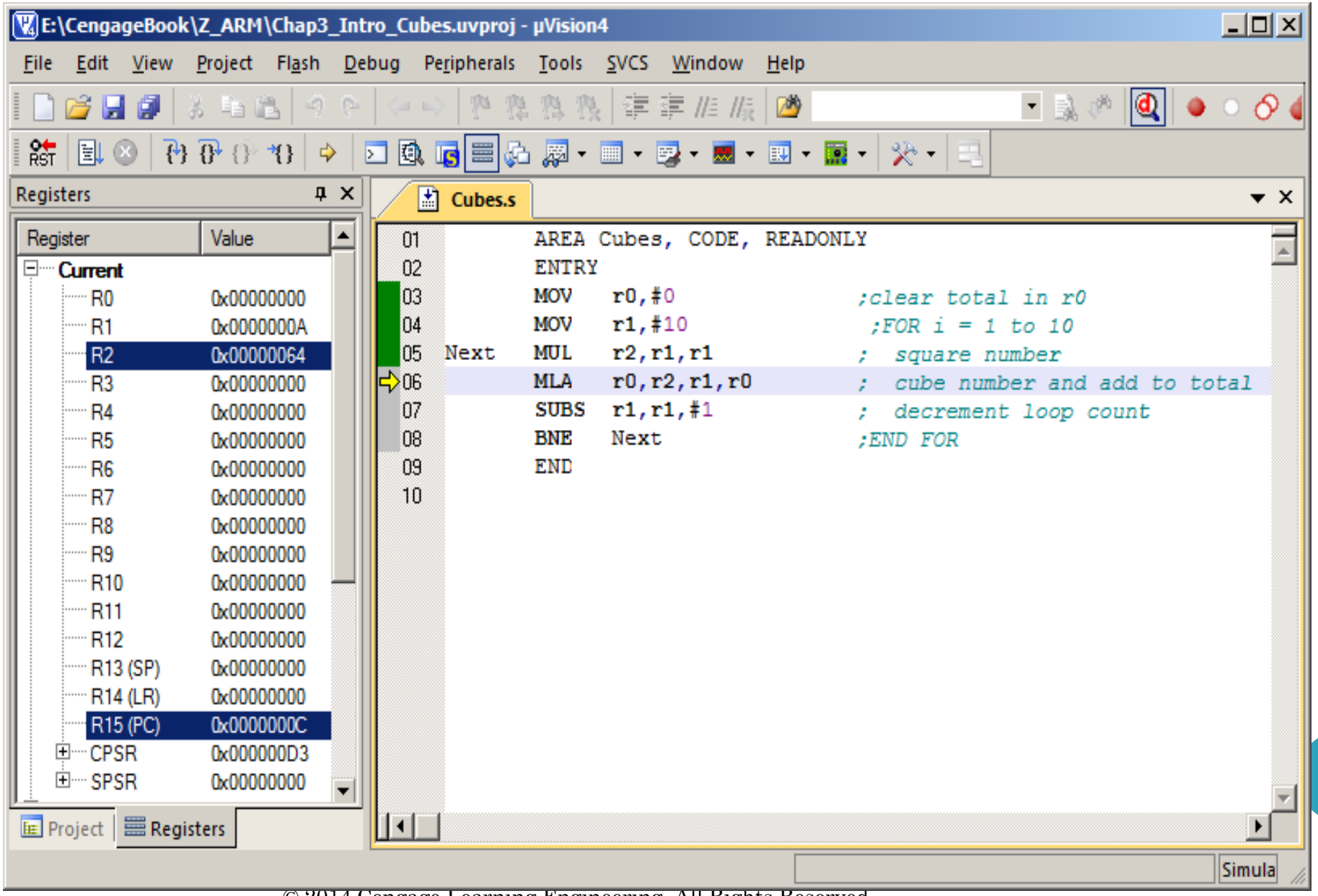
Snapshot of the Display of an ARM Development System

- Executing a program



Snapshot of the Display of an ARM Development System

- Executing a program



The Assembler---Practical Consideration

- Assembly language directives include:

AREA	To name a region of **code** or **data**
ENTRY	The execution starting point
END	The physical end of the program
name EQU *value*	Equate a *name* to a *value* ***Will not make any memory allocation, i.e. similar to* `#define` *in* `C`**
{label} DCD v. expr {, v. expr} …	Set up one or more 32-bit constant in memory *Must start at a multiple of 4 address location*
{label} DCW v. expr {, v. expr} …	Set up one or more 16-bit constant in memory *Must start at an even address location*
{label} DCB v. expr {, v. expr} …	Set up one or more 8-bit constant in memory *Can start anywhere*
{label} SPACE size expr	Reserves a zeroed block of memory *Can start anywhere*
ALIGN	Ensures that next instruction is correctly aligned on 32-bit boundaries, i.e., to start at a multiple of 4 address location

constant value expression