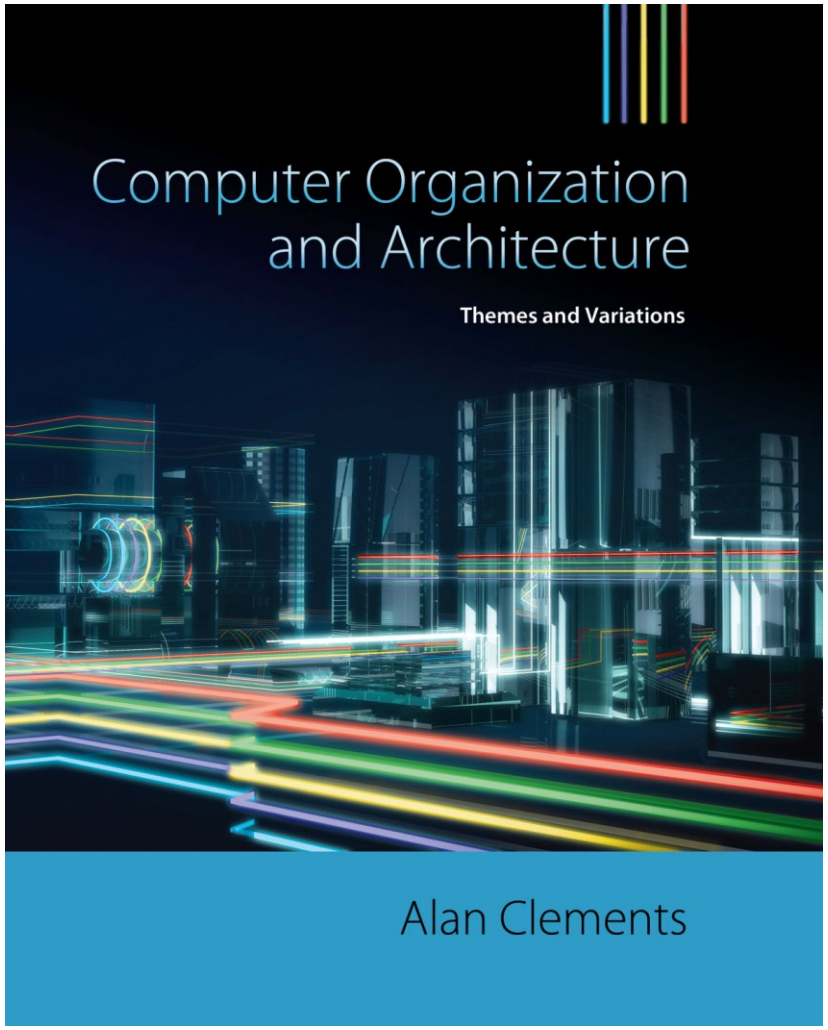
Part 3

Chapter 3

Architecture and Organization

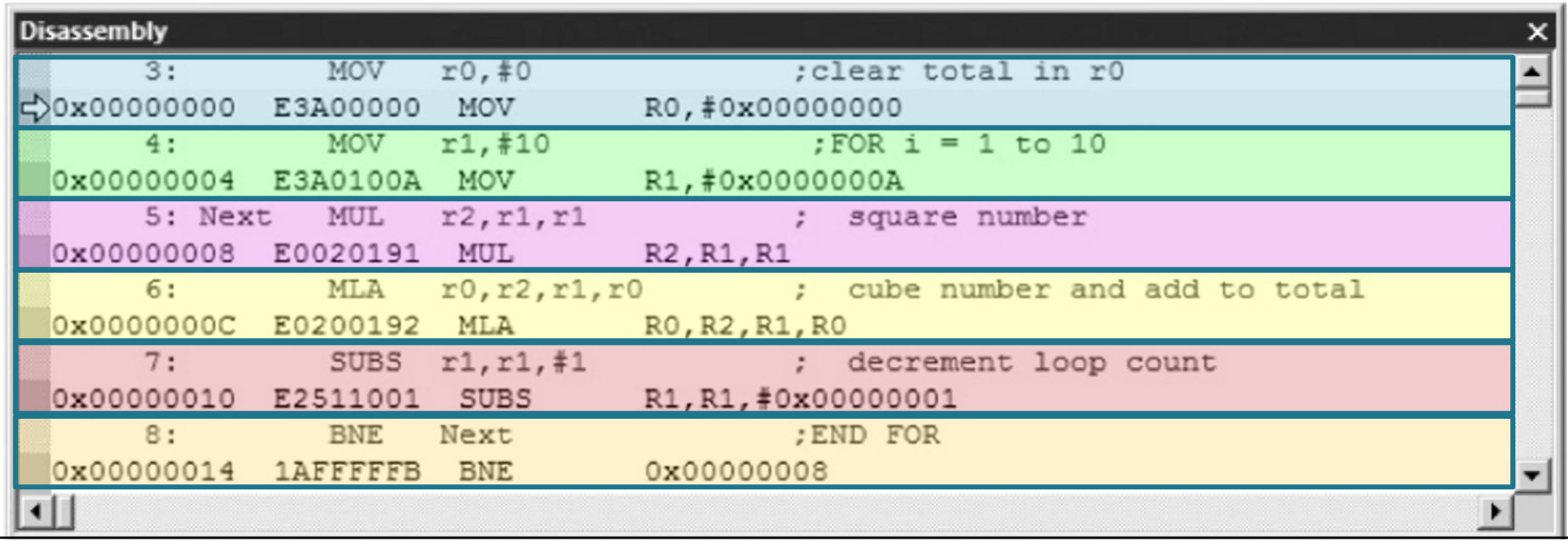


CENGAGE Learning

Snapshot of the Display of an ARM Development System

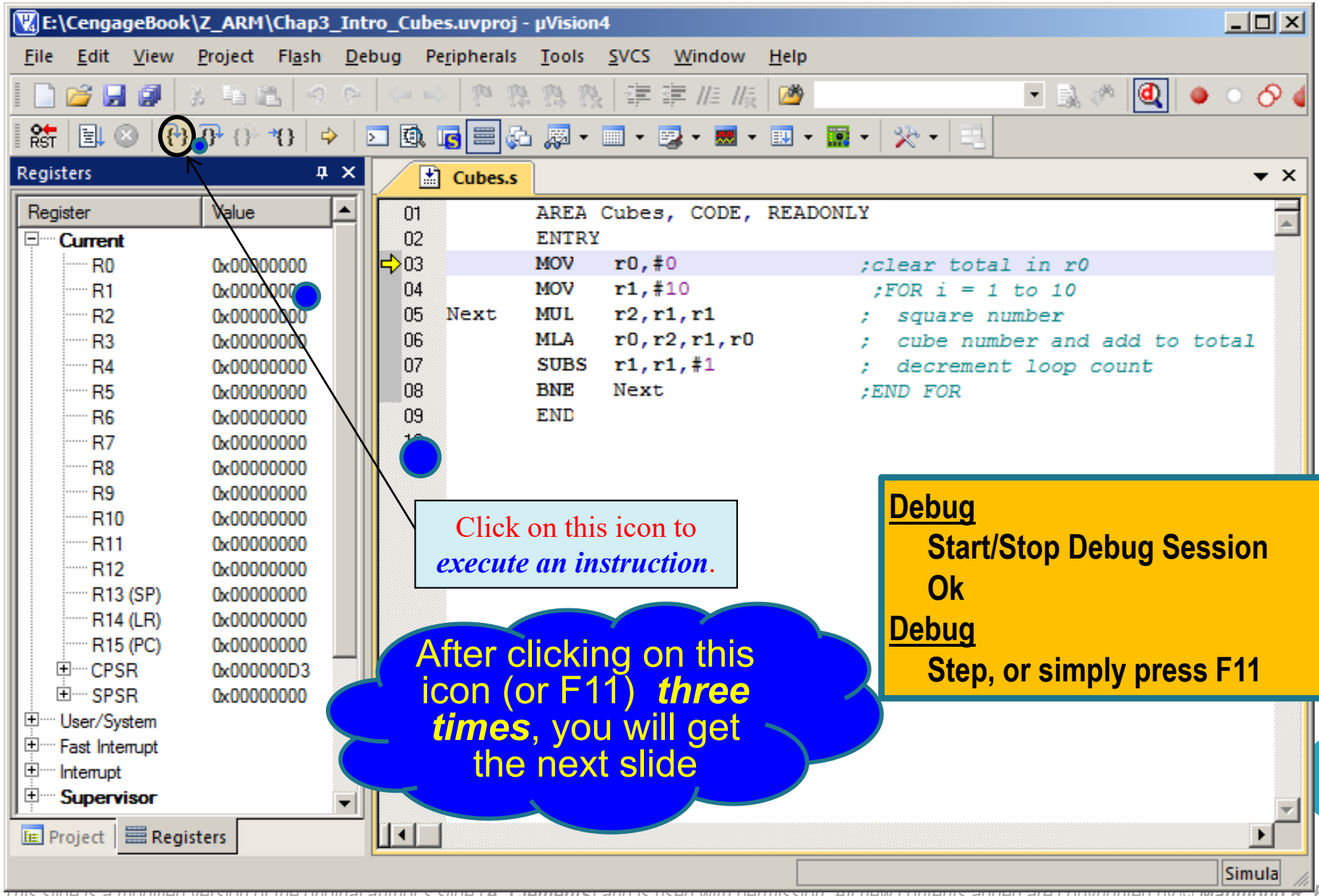
- This is the Disassembly Window that shows memory contents as both
  - hexadecimal values (machine language) and
  - assembly code.

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



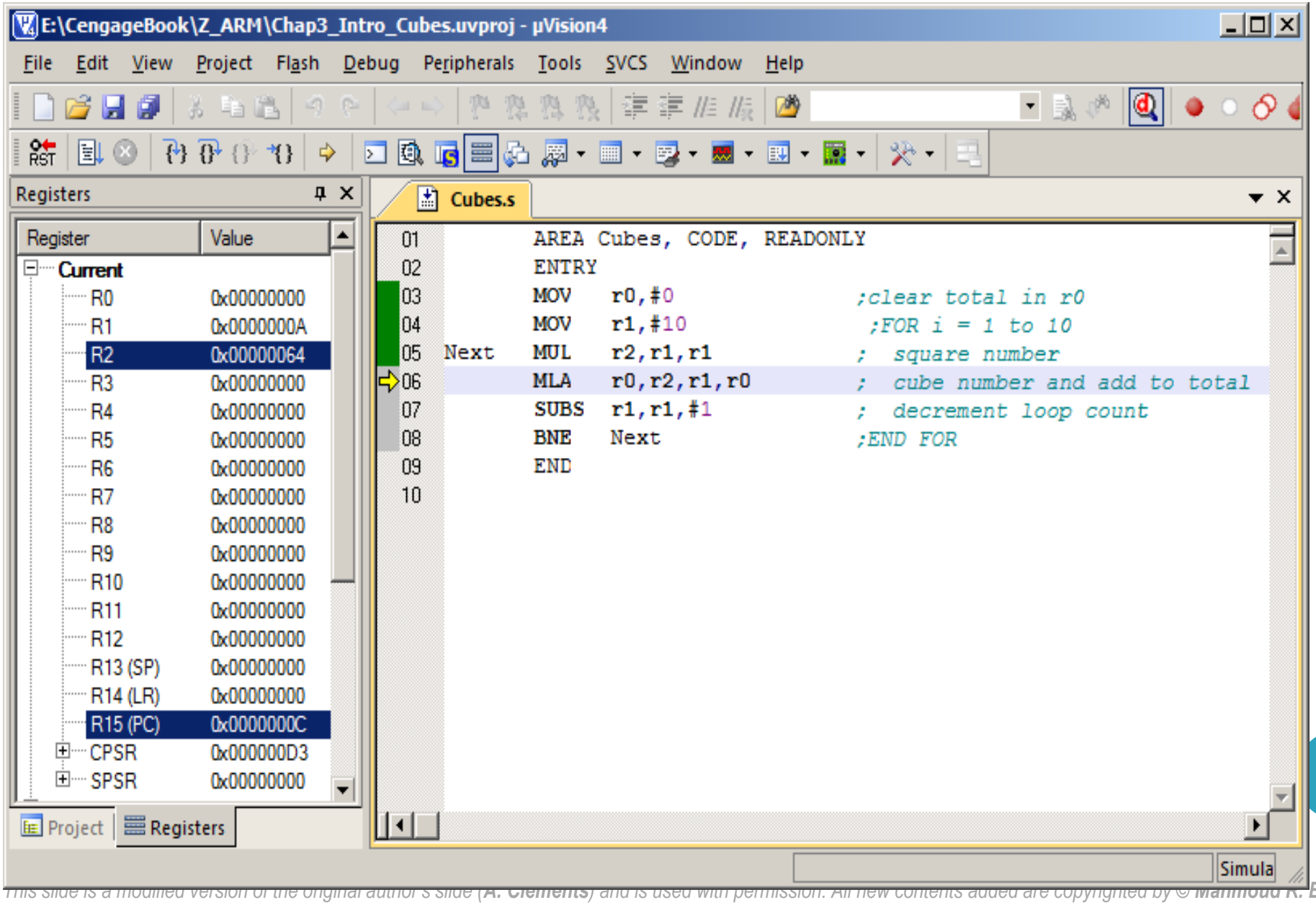
Snapshot of the Display of an ARM Development System

- 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 *(one per program)*
END	The physical end of the program *(one per program)*
name EQU v. expr	Equate a *name* to the *value* of the v. expr ***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 data item is correctly aligned on 32-bit boundaries, i.e., to start at a multiple of 4 address-location

Constant-value expression

Useless if you have an ALIGN before a DCD

The Assembler---Practical Consideration

full half byte

- The ***DCD***, ***DCW***, or ***DCB*** directives tell the assembler to
  - *reserve* one or more *32-bit*, *16-bit*, or *8-bit* of storage in memory, respectively
    - The memory-location used is the next location in sequence,
    - *In case of DCD or DCW, the used location must be on the 32-bit word boundary, or16-bit word boundary, respectively;*
      - *if not, the assembler will insert byte(s) with value of zero to ensure that the data location is on the appropriate boundary*
  - *load* whatever value(s) to the right of ***DCD***, ***DCW***, or ***DCB*** into these location(s). variable in the simulator: track the next available memory
  - *advance* the ***location-counter*** by one or more *four*, *two*, or *one* bytes, space respectively, so that the next instruction/data will be put in the next place in memory.

How about SPACE?

- The ***Location-Counter*** is a ***variable inside the assembler*** to ***keep track of memory-locations during assembling a program***, whereas the ***Program-Counter*** is a ***register inside the CPU*** to ***keep track of the next instruction to be executed*** in a program ***at run time***.

- The ***ALIGN*** directive tells the assembler to *align* the current position (the ***Location-Counter***) to be on the next word boundary, i.e., to start at a dividable by 4 address-location, ***(explicit alignment)***

implicit when current location is unable to

The Assembler---Practical Consideration

```
        AREA Directives, CODE, READONLY
        ENTRY

        MOV  r6,#XX          ;load r6 with 5 (i.e., XX)
        LDR  r7,P1           ;load r7 with the contents at location P1
        ADD  r5,r6,r7        ;just a dummy instruction
        MOV  r0, #0x18       ;angel_SWIreason_ReportException
        LDR  r1, =0x20026    ;ADP_Stopped_ApplicationExit
        SVC  #0x123456       ;ARM software interrupt
XX      EQU  5               ;equate XX to 5
P1      &  0x12345678        ;store hex 32-bit value 0x1345678
P3      DCB  25              ;store the one-byte value 25 in memory
YY      DCB  'A'             ;store byte whose ASCII character is A in memory
Tx2     DCW 12342            ;store the 16-bit value 12342 in memory
        ALIGN                ;ensure code is on a 32-bit word boundary
Strg1   DCB  "Hello"
Strg2   =  "X2", &0C, &0A
Z3      DCW  0xABCD
        END
```

The & sign here is a synonym for DCD

assembler directives are in RED

The = sign here is a synonym for DCB

The & sign here is a synonym for 0x

' ' is used to define a SINGLE ascii character.

The Assembler---Practical Consideration

" " is used to define a string (a sequence of ascii characters).

```
P1      & 0x12345678
P3      DCB 25
YY      DCB 'A'
Tx2     DCW 12342
        ALIGN
Strg1   DCB "Hello"
Strg2   = "X2", &0C, &0A
Z3      DCW  0xABCD
```

FIGURE 3.17 Allocating data to memory

3.18 Allocating data to memory—the memory map

```
000000000018    12          Word 0x12345678
000000000019    34
00000000001A    56
00000000001B    78
00000000001C    19          Byte 25
00000000001D    41          Byte 'A'
00000000001E    30          Half Word 12342
00000000001F    36
000000000020    H           String "Hello"
000000000021    e
000000000022    l
000000000023    l
000000000024    O
000000000025    X           String "X2"
000000000026    2
000000000027    0C          Byte 0x0C
000000000028    0A          Byte 0x0A
000000000029    00            Forced alignment
00000000002A    AB          Half Word 0xABAC
00000000002B    CD
```

To be stored as ASCII values

Strings must be used with DCB

Disassembly

```
     4:         MOV    r6,#XX          ;load
0x00000000  E3A06005  MOV        R6,#0x00000005
     5:         LDR    r7,P1           ;load r7 with the
0x00000004  E59F700C  LDR        R7,[PC,#0x000C]
     6:         ADD    r5,r6,r7        ;just a dummy ins
0x00000008  E0865007  ADD        R5,R6,R7
     7:         MOV    r0, #0x18       ;angel_SWIreason_
0x0000000C  E3A00018  MOV        R0,#0x00000018
     8:         LDR    r1, =0x20026    ;ADP_Stopped_Appl
0x00000010  E59F1014  LDR        R1,[PC,#0x0014]
     9:         SVC    #0x123456       ;ARM semihosting
0x00000014  EF123456  SWI        0x00123456
0x00000018  12345678  EORNES     R5,R4,#0x07800000
0x0000001C  19413036  STMNEDB    R1,{R1-R2,R4-R5,R12-R13}^
0x00000020  48656C6C  STMMIDA    R5!,{R2-R3,R5-R6,R10-R11,R13-R14}^
0x00000024  6F58320C  SWIVS      0x0058320C
0x00000028  0A00ABCD  BEQ        0x0002AF64
0x0000002C  00020026  ANDEQ      R0,R2,R6,LSR #32
0x00000030  00000000  ANDEQ      R0,R0,R0
```

This is X, not x