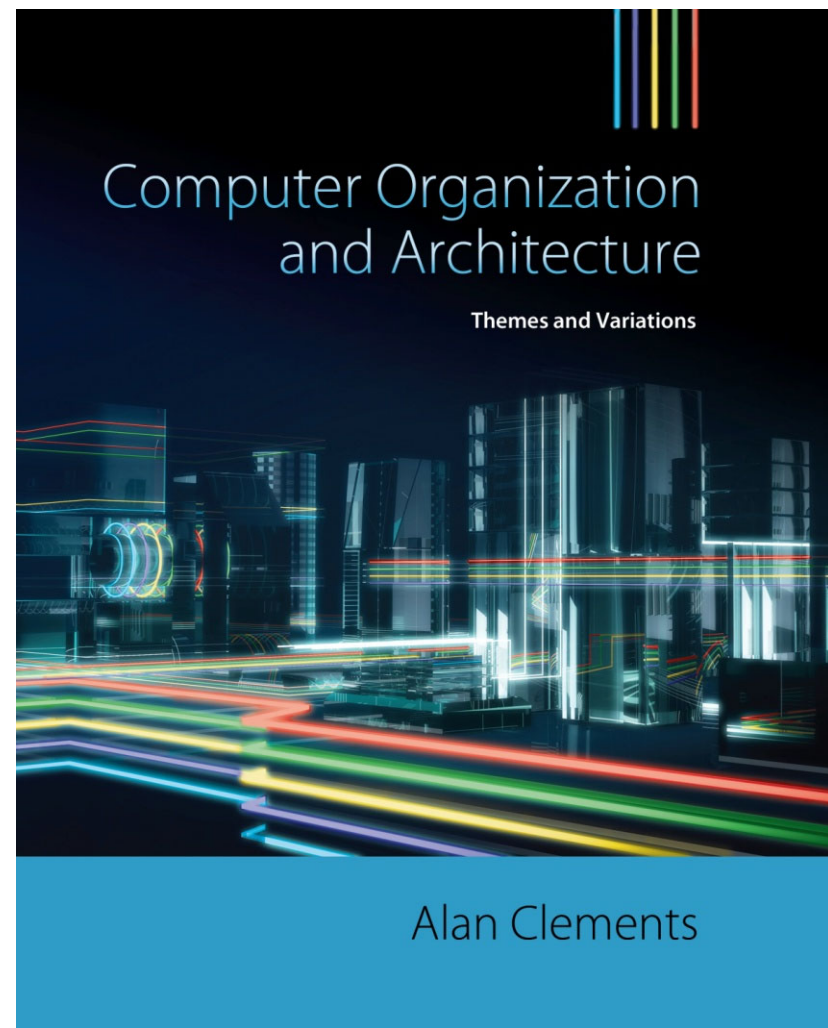


Part 3

CHAPTER 3

Architecture and Organization

1



These slides are being provided with permission from the copyright for CS2208 use only. The slides must not be reproduced or provided to anyone outside of the class.

All download copies of the slides are for personal use only.
Students must destroy these copies within 30 days after receipt of final course evaluations.

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

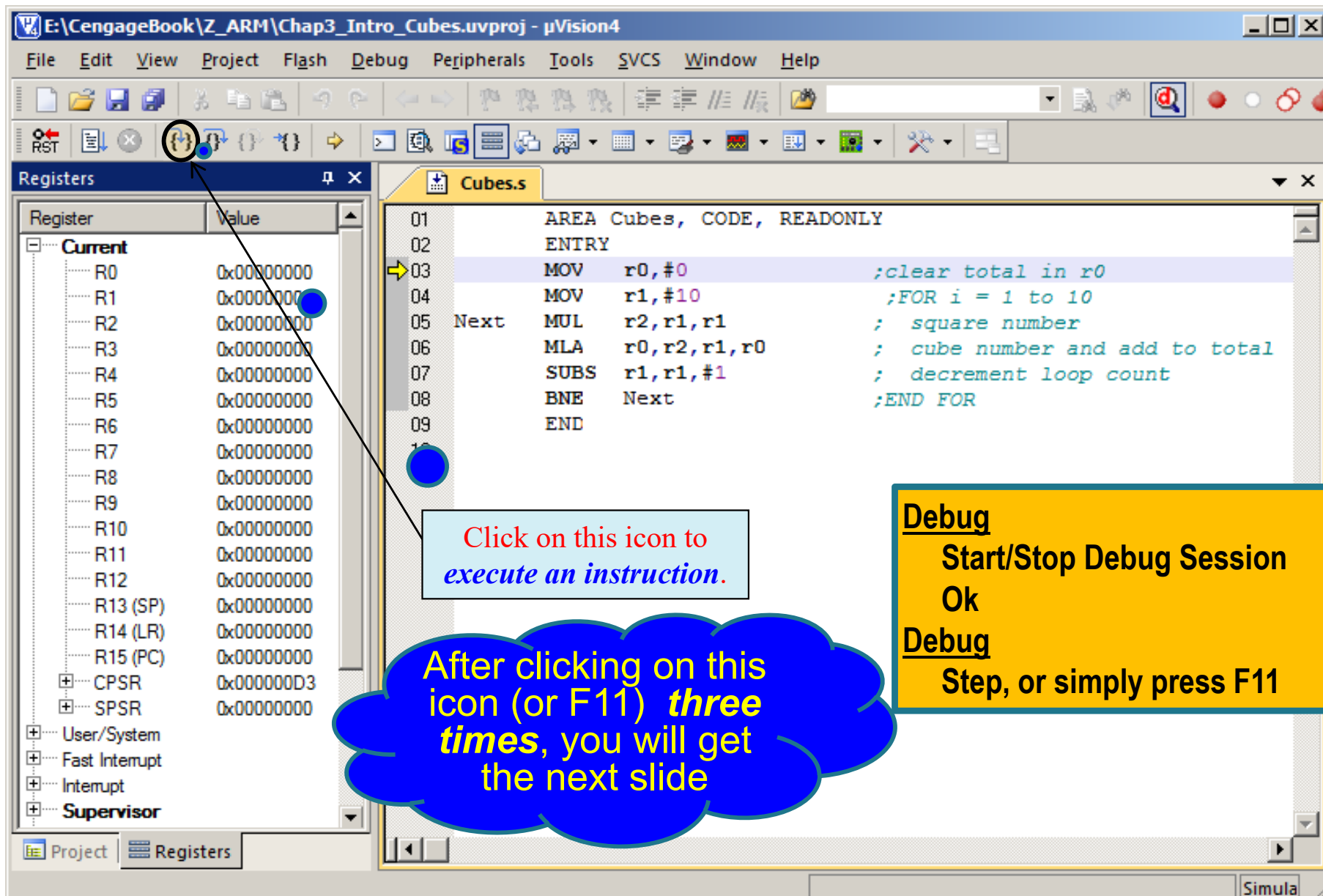
The disassembly window with the hexadecimal code generated by the program

The screenshot shows a window titled "Disassembly" with a list of instructions. Each instruction is displayed with its address, assembly code, and a comment. The instructions are color-coded: blue for MOV, green for MOV, purple for MUL, yellow for MLA, pink for SUBS, and orange for BNE.

Address	Assembly Code	Comment
3:	MOV r0,#0	;clear total in r0
0x00000000	E3A00000 MOV R0,#0x00000000	
4:	MOV r1,#10	;FOR i = 1 to 10
0x00000004	E3A0100A MOV R1,#0x0000000A	
5: Next	MUL r2,r1,r1	; square number
0x00000008	E0020191 MUL R2,R1,R1	
6:	MLA r0,r2,r1,r0	; cube number and add to total
0x0000000C	E0200192 MLA R0,R2,R1,R0	
7:	SUBS r1,r1,#1	; decrement loop count
0x00000010	E2511001 SUBS R1,R1,#0x00000001	
8:	BNE Next	;END FOR
0x00000014	1AFFFFFFB BNE 0x00000008	

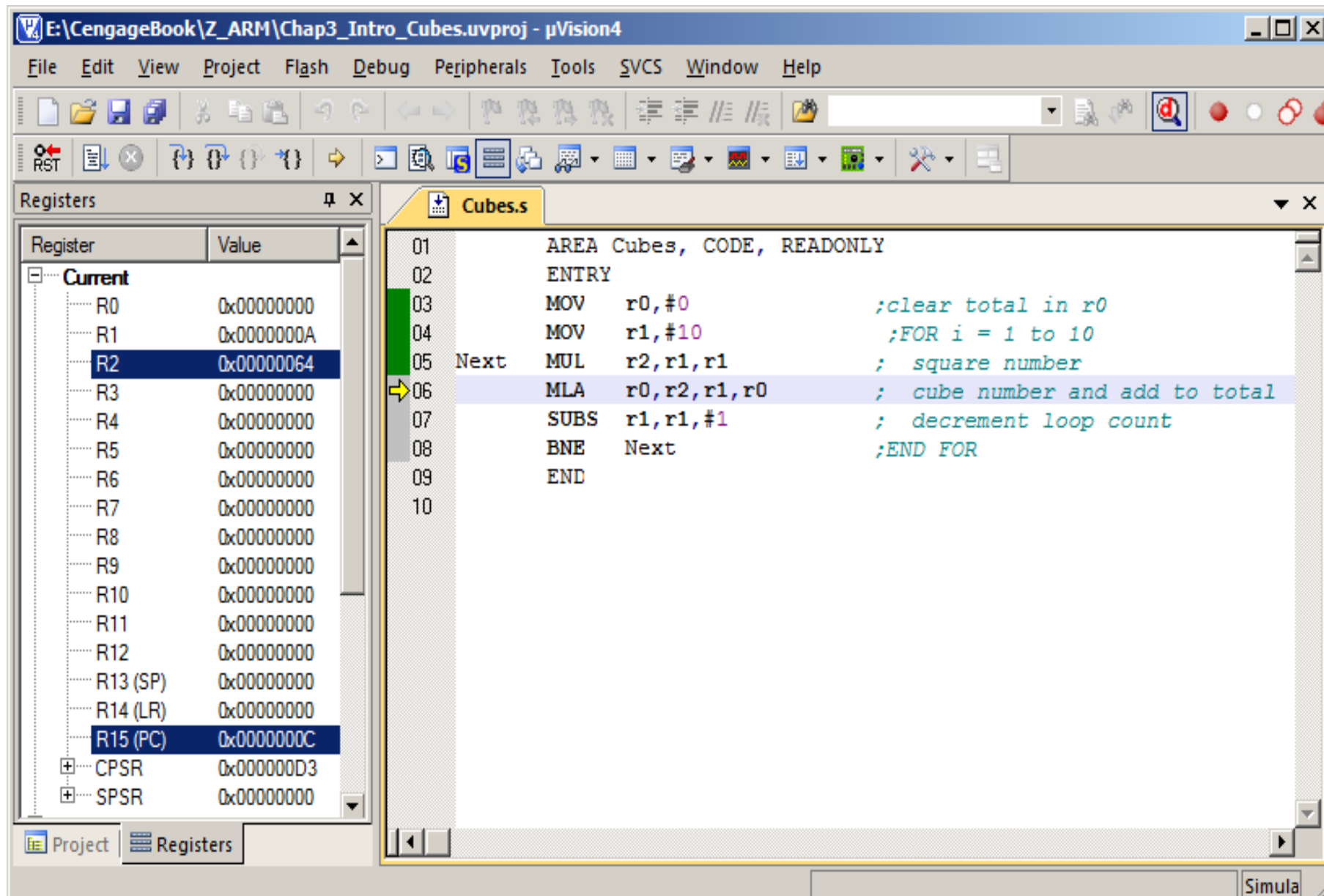
Snapshot of the Display of an ARM Development System

❑ 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 *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

Constant-value
expression

{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

The Assembler--Practical Consideration

- ❑ 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, or 16-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).
 - **advance** the **location-counter** by one or more **four**, **two**, or **one** bytes, respectively, so that the next instruction/data will be put in the next place in memory.
- ❑ 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 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 multiple of 4 address-location, (**explicit alignment**)

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

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
  
```

3.18

Allocating data to memory—the memory map

00000000000018	12	Word 0x12345678
00000000000019	34	
0000000000001A	56	
0000000000001B	78	
0000000000001C	19	Byte 25
0000000000001D	41	Byte 'A'
0000000000001E	30	Half Word 12342
0000000000001F	36	
00000000000020	H	String "Hello"
00000000000021	e	
00000000000022	1	
00000000000023	1	
00000000000024	O	
00000000000025	X	String "X2"
00000000000026	2	
00000000000027	0C	Byte 0x0C
00000000000028	0A	Byte 0x0A
00000000000029	00	Forced alignment
0000000000002A	AB	Half Word 0xABAC
0000000000002B	CD	

To be stored as
ASCII values

Strings must be
used with DCB

This is
X, not x