CHAPTER 2: AVR ARCHITECTURE & ASSEMBLY LANGUAGE PROGRAMMING

SECTION 2.1: THE GENERAL PURPOSE REGISTERS IN THE AVR

1. 8
2. 8
3. 8
4. 0xFF
5. $28 – in R20
6. (a), (c), (d), (e), (g)
7. (c)
8. This is an illegal instruction since the arguments of ADD should be register. If they instruction was valid 0x44 would be stored in R19
9. This is an illegal instruction since the arguments of ADD should be register. If they instruction was valid 0xFF would be stored in R21
10. True

SECTION 2.2: THE AVR DATA MEMORY

11. internal SRAM
12. True
13. True
14. False
15. False
16. data memory
17. data memory = general purpose register + I/O registers + SRAM. All AVRs have 32 general purpose registers. AVRs with less than 32 I/O pins, which are not a member of special purpose AVR, have 64 I/O registers. The size of SRAM can be found in figures of Chapter 1.
    (a) 32 + 64 + 2048 = 2144
    (b) 32 + 64 + 1024 = 1120
    (c) 32 + 64 + 256 = 352

18. EEPROM does not lose its data when power is off, whereas SRAM does. So, the EEPROM is used for storing data that should rarely be changed and should not be lost when the power is off (e.g., options and settings); whereas the SRAM is used for storing data and parameters that are changed frequently.
19. Yes
20. No, each microcontroller should have general purpose registers and I/O registers.
21. From $60 to $FFFF
22. 65,536 bytes

SECTION 2.3: USING INSTRUCTIONS WITH THE DATA MEMORY

23.

```
LDI   R20, $30
STS   $105, R20
LDI   R20, $97
STS   $106, R20
```

24.

```
LDI   R20, $55
STS   $300, R20
STS   $301, R20
STS   $302, R20
STS   $303, R20
STS   $304, R20
STS   $305, R20
STS   $306, R20
STS   $307, R20
STS   $308, R20
```

25.

```
LDI R16, $5F
OUT PORTB, R16
```

26. True

27.

```
LDI R30, $11
STS $100, R30
STS $101, R30
STS $102, R30
STS $103, R30
STS $104, R30
STS $105, R30

LDS R20, $100
LDS R16, $101
ADD R20, R16
LDS R16, $102
ADD R20, R16
LDS R16, $103
ADD R20, R16
LDS R16, $104
ADD R20, R16
LDS R16, $105
ADD R20, R16
```

28.

```
LDI R30, $11
STS $100, R30
STS $101, R30
STS $102, R30
STS $103, R30
STS $104, R30
STS $105, R30
```

```
LDS R20, $100
LDS R16, $101
ADD R20, R16
LDS R16, $102
ADD R20, R16
LDS R16, $103
ADD R20, R16
LDS R16, $104
ADD R20, R16
LDS R16, $105
ADD R20, R16

STS $105, R20
```

29.

```
LDI R16, $15
STS $67, R16

LDI R19, 0
LDS R20, $67
ADD R19, R20
ADD R19, R20
ADD R19, R20
ADD R19, R20
ADD R19, R20
```

30.

```
LDI R16, $15
STS $67, R16

LDI R19, 0
LDS R20, $67
ADD R19, R20
ADD R19, R20
ADD R19, R20
ADD R19, R20
ADD R19, R20

STS $67, R19
```

31.

```
LDS R27, $68
COM R27
```

32.

```
LDS R19, $68
OUT PORTC, R19
```

SECTION 2.4: AVR STATUS REGISTER

33. 8
34. 0, 5

35. 3, 2
36. When there is a carry beyond the D7 bit.
37. When there is a carry from the D3 to the D4 bit.
38. C = 1 because there is a carry beyond the D7 bit.
    Z = 1 because the R20 (the result) has value 0 in it after the addition.
39.
    (a)

	$54	0101 0100	
+	$C4	1100 0100	
	$118	10001 1000	R20 = $18

    C = 1 because there is a carry beyond the D7 bit.

    (b)

	$00	0000 0000	
+	$FF	1111 1111	
	$FF	1111 1111	R23 = $FF

    C = 0 because there is no carry beyond the D7 bit.

    (c)

	$FF	1111 1111	
+	$05	0000 0101	
	$FF	10000 0100	R30 = $04

    C = 1 because there is a carry beyond the D7 bit.

40.

```
LDI R16, $55
LDI R20, $55
ADD R16, R20
ADD R16, R20
ADD R16, R20
ADD R16, R20
ADD R16, R20
```

SECTION 2.5: AVR DATA FORMAT AND DIRECTIVES

41.

```
.EQU MYDAT_1 = $37
.EQU MYDAT_2 = $62
.EQU MYDAT_3 = $47
.EQU MYDAT_4 = $50
.EQU MYDAT_5 = $C8
.EQU MYDAT_6 = $41
.EQU MYDAT_7 = $AA
.EQU MYDAT_8 = $FF
.EQU MYDAT_9 = $90
.EQU MYDAT_10 = $7E
.EQU MYDAT_11 = $0A
.EQU MYDAT_12 = $0F
```

42.

```
.EQU DAT_1 = $16
.EQU DAT_2 = $56
.EQU DAT_3 = $99
.EQU DAT_4 = $20
.EQU DAT_5 = $F6
.EQU DAT_6 = $FB
```

43.

```
.EQU TEMP0 = $60
.EQU TEMP1 = $61
.EQU TEMP2 = $62
.EQU TEMP3 = $63
.EQU TEMP4 = $64
.EQU TEMP5 = $65

      LDI R16, $11
      STS TEMP0, R16
      STS TEMP1, R16
      STS TEMP2, R16
      STS TEMP3, R16
      STS TEMP4, R16
      STS TEMP5, R16

      LDS R20, TEMP0
      LDS R21, TEMP1
      ADD R20, R21
      LDS R21, TEMP2
      ADD R20, R21
      LDS R21, TEMP3
      ADD R20, R21
      LDS R21, TEMP4
      ADD R20, R21
      LDS R21, TEMP5
      ADD R20, R21
```

SECTION 2.6: INSTRUCTION TO AVR ASSEMBLY PROGRAMMING AND

SECTION 2.7: ASSEMBLING AN AVR PROGRAM

44. Low, High
45. Assembly
46. Assembler
47. True
48. False
49. False
50. No
51. Because they do not produce machine instructions. They just give directions to the assembler.
52. True
53. hex
54. hex, eep, lst, map, and obj

SECTION 2.8: THE PROGRAM AND PROGRAM ROM SPACE IN THE AVR

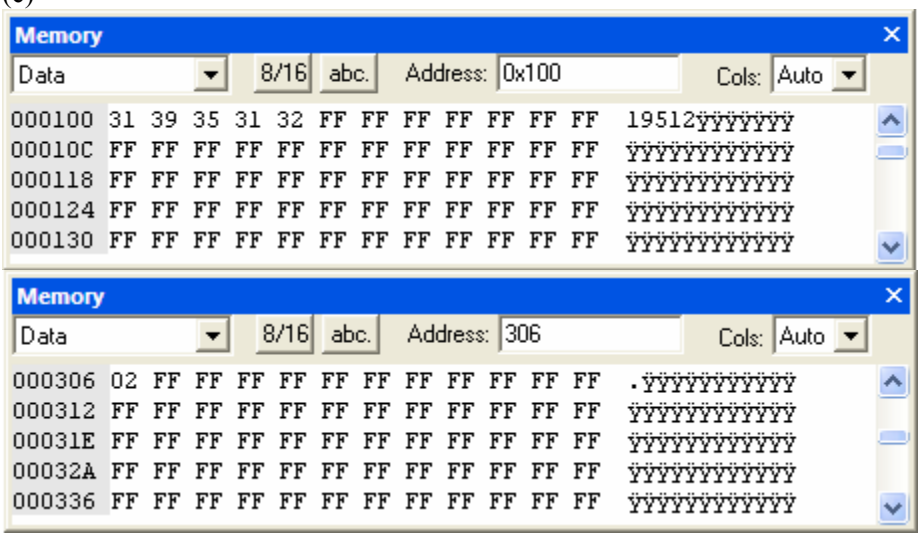
55. 0
56. It executes whatever is at location 0 which could be garbage in this case.
57. a) 2 bytes b) 2 bytes c) 2 bytes d) 2 bytes
    e) 2 bytes f) 2 bytes g) 2 bytes h) 4 bytes
58. (a)

```
LDI R20,'1'
STS 0x100, R20
LDI R20,'9'
STS 0x101, R20
LDI R20,'5'
STS 0x102, R20
LDI R20,'1'
STS 0x103, R20
LDI R20,'2'
STS 0x104, R20
```

(b)

```
LDI R19,0
LDS R16, 0x100
ADD R19,R16
LDS R16, 0x101
ADD R19,R16
LDS R16, 0x102
ADD R19,R16
LDS R16, 0x103
ADD R19,R16
LDS R16, 0x104
ADD R19,R16
STS 0x306, R19
```

(c)



59. In AVR, each location of the program memory holds 2 bytes; therefore:
    a) Memory locations = 32 K / 2 = 16 K = 16 * 1024 = 16384 ➔ Last location = 16383 = $3FFF

b) Memory locations = 8 K / 2 = 4 K = 4 * 1024 = 4096 ➔ Last location = 4095 = $FFF
c) Memory locations = 64 K / 2 = 32 K = 32 * 1024 = 32768 ➔ Last location = 32767 = $7FFF
d) Memory locations = 16 K / 2 = 8 K = 8 * 1024 = 8192 ➔ Last location = 8191 = $1FFF
e) Memory locations = 128 K / 2 = 64 K = 64 * 1024 = 65536 ➔ Last location = 65535 = $FFFF

60. In ATmega32, the program memory is 32K bytes. Since the 32K is organized as 16K x 2 Bytes, the last location has the address of $3FFF. Therefore the program counter can have values between 0 and $3FFF.
61. $7FFF = 32767 ➔ the program memory has 32767 + 1 = 32768 locations. Therefore, it has 65536 = 64K bytes.
62. $3FF = 1023 ➔ the program memory has 1024 locations. Therefore, the size of program memory is 2 Kbytes.
63. (a) $1FFF + 1 = 8,192 words = 16,384 bytes = 16 KB
    (b) $3FFF + 1 = 16,384 words = 32 KB
    (c) $7FFF + 1 = 32,768 words = 64 KB
    (d) $FFFF + 1 = 65,536 words = 128 KB
    (e) $1FFFF + 1 = 131,072 words = 256 KB
    (f) $3FFFF + 1 = 262,144 words = 512 KB
    (g) $FFF + 1 = 4096 words = 8 KB
    (h) $1FF + 1 = 512 words = 1 KB

64. (a) $3FF + 1 = 1024 words = 2 KB
    (b) $7FF + 1 = 2048 words = 4 KB
    (c) $7FFFF + 1 = 524,288 words = 1,048,576 KB
    (d) $FFFFF + 1 = 1,048,576 words = 2048 KB = 2 MB
    (e) $1FFFFF + 1 = 2,097,152 words = 4096 KB = 4 MB
    (f) $3FFFFF + 1 = 4,194,304 words = 8192 KB = 8 MB
    (g) $5FFF + 1 = 24,576 words = 49152 bytes = 48 KB
    (h) $BFFFF + 1 = 786,432 words = 1,572,864 bytes = 1536 KB = 1.5 MB

65. 2 bytes
66. 2 bytes
67. As shown in Figure 2-14, 8 bits are set aside for K. Therefore, K can be between 0 and 255.
68. $0C01 = 0000 1100 0000 0001. According to the figures of page 92, it is the machine code for the ADD instruction.
69. It is a 4-byte instruction. 16 bits of it are set aside for K. Therefore, K can be between 0 and 65535. In AVR, the data memory is 64 KB; as a result, STS can address the entire memory space.
70. It is a 4-byte instruction. 16 bits of it are set aside for K. Therefore, K can be between 0 and 65535. In AVR, the data memory is 64 KB; as a result, LDS can address the entire memory space.
71. In the JMP instruction, 22 bits are set aside for K. Therefore K can be between 0 and 4,194,303.

SECTION 2.9: RISC ARCHITECTURE IN THE AVR

72. RISC is reduced instruction set computer; CISC stands for complex instruction set computer.
73. CISC
74. RISC
75. RISC
76. CISC
77. False