2 of the non-short answers are incorrect.

1. (Margush textbook) Page 66, Exercises #4

What is the usual length of an AVR machine instruction (in bits)?

16 bits, page 28

2. (Margush textbook) Page 66, Exercises #6

How many general purpose registers are present in the AVR core? What is the size of each register (in bytes)?

32 registers, 1 byte each – page 28

3. (Margush textbook) Page 66, Exercises #7

What is the name of the special register that always contains the address of the next instruction to be executed?

Program counter/PC, page 29

4. (Margush textbook) Page 66, Exercises #8

If an AVR processor is running with a clock speed of 4 MHz, what is the maximum

number of instructions it can execute in a single second?

4 million, page 30

5. (Margush textbook) Page 66, Exercises #10

Why does the LDI instruction only work with half of the available general purpose registers? Which half can it affect?

LDI can only work with half of the available registers because it has only 4 bits dedicated to representing the destination register in its machine code, allowing for only 16 (24) different locations to be specified. LDI can affect registers 16-31 inclusive.

(Margush, Page 32)

6. (Margush textbook) Page 66, Exercises #11

The ADD instruction can utilize any general purpose register as the destination of the result (sum). What is different about the encoding of the ADD and LDI instructions that makes this possible?

Since LDI needs 8 bits to encode a constant value and 4 more for the opcode, only 4 are left to specify the destination register. In contrast, the ADD instruction needs 6 bits for the opcode, but it then has 10 bits with which to specify the operands (which are *both* registers). Giving 5 bits to each allows all 32 (25)distinct registers to be used with the ADD instruction.

(Margush, Page 34)

7. (Margush textbook) Page 67, Exercises #14

How similar are these two instructions:

LSL R2

ADD R2, R2

Explain.

The two instructions are nearly identical.

LSL effectively multiplies the contents of R2 by 2, since it shifts all the bits left by one. It also loads the most significant bit into the carry flag.

ADD R2, R2 adds the content of R2 to its self, also effectively multiplying the value by 2, and stores the result in R2. It also sets the carry flag to the appropriate value (which happens to be the most significant bit of the contents of R2 prior to the instruction.

(AVR Instruction Set Manual, Pages 17, 99)

8. (Margush textbook) Page 67, Exercises #17

Hand assemble the following instructions. Express each machine language instruction in hexadecimal (4 hex digits)

a. LDI R20, 45

b. LDI R31, $C2

c. ADD R18, R19

d. ADD R0, R16

e. RJMP 27 (Jumps to PC + 1 + 27)

a. 0xE24D

b. 0xECF2

c. 0x0F23

d. 0x0E00

e. 0xC01B

want little endean so

1. 4DE2
2. F2EC
3. 230F
4. 000E
5. 1BC0

9. (Margush textbook) Page 100, Exercises #7

Use a directive to define a symbol named isDefined as the number 0. This symbol may change value later in the program.

.set isDefined=0

Page 76

10. (Margush textbook) Page 101, Exercises #14

What is the numeric value of each expression truncated to a single byte? Answer in hex or unsigned decimal.

a. !2

b. ~0

c. “/45”

d. 2 + 7/3

e. 0xFF << 0x04

1. 0x00 – Page 84
2. 0xFF – Page 84
3. \_ this should just be the ascii value of “5”ungraded
4. 0x04
5. 0xF0 – Page 84

11. (Margush textbook) Page 102, Exercises #22

The instructions that use labels for operands can also accept numbers (after all, labels represent numeric values). Write an instruction that copies the byte at address 0x00F0 of the data space into register 2.

Lds R2, 0x00F0

Try 0xF000

Question: Don’t we have to restrict our data access to addresses after 0x200?

12. (Margush textbook) Page 102, Exercises #23

Some might call the nop instruction an oxymoron. Why? Why would a processor have such an instruction (do a little research if necessary)?

The nop instruction means “no instruction”, so it is an oxymoron since it is an instruction which is by definition not an actual instruction. Processors have such an instruction to allow the programmer to “do nothing” for a period of time without clunky or strange code. By repeatedly calling nop, we can effectively halt the execution of our programs for as long as desired. This can be useful in many scenarios. For example we may wish to pause long enough for the user to be able to interpret program output, or to pause until the user gives necessary input.

13. (Margush textbook) Page 102, Exercises #24

Lookup the addresses of the I/O registers in the ATMEGA16A for the digital I/O port A. Write directives that correctly define the symbols PORTA, PINA, and DDRA.

PORTA is 0x1B (0x3B)

PINA is 0x19 (0x39)

DDRA is 0x1A (0x3A)

.equ DDRA = 0x1A

.equ PINA = 0x19

.equ PORTA = 0x1B

14. (Margush textbook) Page 138, Exercises #8

What is found in R18 after thus sequence of instructions?

Ldi r18, 0x80  
add r18, r18  
adc r18, r18

0x01

15. (Margush textbook) Page 139, Exercises #17

If r16 contains $E2, a two’s compliment code for an integer, and the instruction inc r16 is executed, what is the new value in r16 and what integer does it represent?

E3 would be in it?

If so, this represents -29

Still unsure as to whether this is correct