Homework 5 [TOTAL POINTS: 42]

1. Convert the following 32 bit floating point hexadecimal number to its decimal value (as per the scheme mentioned in the book): 3EA00000

Note that these conversions cannot be performed directly: hexadecimal and decimal numbers must first be converted into intermediate binary in order to do the floating point conversion. Also, this is a 32 bit floating point number. Refer to section 4.5 in the textbook for the scheme of the bits (1 bit for the sign, 8 bits for the exponent, and the remaining 23 bits for the fraction).

Show your work for each step of the conversion for full credit.

(2)

Hexadecimal to binary:

0 01111101 010 0000 0000 0000 0000 0000

sign exponent fraction

\+ 125 0.25

32-bit floating point number = (sign) * $2^{\text{exponent - 127}}$ * (1+fraction)

1 * 2^(125-127) * (1+0.25)

0.3125

2. There are three parts of an ISA. What are they? Explain each in **detail**. **(3)**

architecture: basic physical components

instruction set: instructions computer understands

encoding: instructions using a human-readable notation

3. For the following types of memory on the AVR chip, describe their type (persistent or non-persistent), size (depth and width), speed, and function(s).

(3)

(a) program memory

Type: Persistent

Size: 65536 by 16 (width) = 131072 bytes

Speed: Slow

Function: Storing the actual instructions that are being executed

(b) RAM

Type: Non-persistent

Size: 65536 by 8 (width) = 65536 bytes

Speed: Slow

Function: Storing large amounts of data that is not all needed immediately.

(c) SREG (status register)

Type: Non-persistent

Size: 1 by 8 (width) = 1 byte

Speed: Very fast

Function: Storing information about the most recent arithmetic operation the computer performed

4. What is assembly language? What improvements does it make to ISA? **(1)**

   Assembly language is a very slight layer over ISA. The main improvements are 1) specifying offsets for jumps and branches, 2) initializing ram with specified data, and 3) including string syntax.

5. What is non-persistent memory and why is it useful? (**1)**
   Non-persistent memory erases after you turn the power off. It is useful if you do not want memory saved after shutting down the power, like rage quitting video games.

6. What does an assembler do? **(1)**
   Assembler converts ISA into numbers for storage into the program memory.

7. What registers can be used in the ldi instruction? **(1)**
   reg 16 through 31

8. There are 7 control flow instructions that alter the program counter: **breq, brne, brsh, brlo, rjmp, rcall,** and **ret**. We can roughly categorize these as branching, jumps, and return instructions. When would you use one type of instruction over the other? **(2)**

   branching: **when a comparison is done, conditional**
   rjmp: general movement throughout code, unconditional
   rcall/return: to go back once we finish running a block of code, used in functions

9. For the following assembly code, trace through it by hand and write the N, C and Z flag and PC values after each line for the first 8 lines executed. Assume the initial values of all registers to be 0. Note that AVR effectively uses an 8-bit representation, and the effect of this fact on the N-flag. Check your answer with the browser-based AVR simulator.
   (NOT GRADED)

   C flag is set according to non-negative value
   N flag is set according to two's complement value

Instruction	**Negative**	**Carry**	**Zero**	**PC**
`ldi r16,100 ; r16 = 100`	0	0	0	1
`ldi r17,244 ; r17 = 244`	**0**	**0**	**0**	**2**

`add r17,r16 ; r17 = 344, overflows; r17 = 88`	**0**	**1**	**0**	**3**
`add r17,r17 ; r17 = 176, which can't be represented in 8 bits; r17 = -80`	**1**	**0**	**0**	**4**
`sub r16,r17 ; set if op1 < op2 (unsigned); r16 = 100; r17 = 176, r16 = -76`	**1**	**1**	**0**	**5**
`ldi r18, 1`	**1**	**1**	**0**	**6**
`cp r16, r18`	**1**	**0**	**0**	**7**
`breq -3 ; branch does not modify status register`	**1**	**0**	**0**	**8**
`ldi r19, 0 ; load immediate does not modify status register`	**1**	**0**	**0**	**9**

10. Consider the code snippet below.

(a) Trace through the code by hand and fill out the table below. Assume all registers are initialized at 0. Check your answer with the browser-based AVR simulator. **(2)**

```
ldi r16, 10      ; line i
ldi r17, 7       ; line ii
inc r17          ; line iii
cp r16, r17      ; line iv
brne -3          ; line v
halt             ; line vi
```

Current line	**Calculation performed by this line**	**Registers**		**Next line:**
		r16	**r17**	
i	load 10 into r16	10	0	ii
ii	load 8 into r17	10	8	iii
iii	r17 = 9	10	9	iv
iv	compare 9 and 10	10	9	v

v	jump back to line 3	10	9	iii
iii	r17 = 10	10	10	iv
iv	compare r16 and r17	10	10	v
v	since they are equal, we ignore this line	10	10	vi
vi	halt			

(b) Hopefully, you were able to fill out the table and not run into an infinite loop. Will you get an infinite loop if the second line of the program was "ldi r17, 11" instead? If yes, what makes it infinite? If no, at what point will the loop stop.

(1)

It would hit the max (overflow) and return to 0. r17 would then continuing incrementing from 0 until it equals 10 and break out of the loop.

11. Write assembly code to configure the **complete** PORT D as an output port and write 134 on this port.
    (2)

```
ldi r30, 255
out 17, r30
ldi r30, 134
out 18, r30
```

12. Using the stack, write a simple **assembly language** function labeled *myfunction* that compares two values ~~in the top two entries of the stack~~ and prints the higher number to the output LED. If they are equal, print either value.

Initialize your stack to RAM location 3000. ~~Push values 4 and 3 onto the stack and then call this function.~~ Run your function with the values 3 and 4. **Annotate your code with comments for full credit. Submit using the browser-based simulator. (4)**

Sample answer:

```
rjmp myprogram
```

```
myfunction:     ; pass inputs via r19 and r20
push r16        ; push r16 because want it to use to set output
ldi r16, 255    ; set r16 to all high bits
out 17, r16     ; all high bits IO 17 for output
cp r19, r20     ; compare values passed into myfunction
brlo 2          ; if r19 < r20, skip to out 18, r20
out 18, r19     ; output the contents of r17
rjmp 1          ; jump over the next instruction
out 18, r20     ; output the contents of r18
pop r16         ; pop registers used
ret             ; end of myfunction

myprogram:
; initialize stack pointer to 3000 = 11*256 + 184
ldi r31, 184
out 61, r31
ldi r31, 11
out 62, r31
; prepare 3 and 4 to pass into myfunction
ldi r19,3
ldi r20,4
rcall myfunction ;jump to myfunction
```

13. Convert the following Python code into **assembly language**. Assume x is stored in r16, y in r17, and z in r18, and a in r19. You may store values in other registers if necessary. **Annotate your code with comments.** **(4)**

```
x = 2
y = 1
z = x + y
if ( z >= 2):
     a = 1
else:
     a = 2
```

Sample answer:

```
ldi r16, 2      ; load x
ldi r17, 1      ; load y
mov r18,r16     ; z = x
add r18,r17     ; z = x + y
```

```
ldi r20, 2      ; our comparison value
cp r18, r20     ; if z > 2
brsh 2          ; jump to ldi r19, 1
ldi r19, 2      ; a = 2
rjmp 1          ; jump over the next instruction
ldi r19, 1      ; a = 1
```

14. Repeat Homework 2 Question 11, but this time use **assembly language** instead of Python. **(6)**
    A hard coded solution will result in 0 points.
    Annotate your code with comments, it is worth point(s).
    Use the browser-based simulator to write and submit.

    For reference, Homework 2 Question 11:

    Write a program to print the first n Fibonacci numbers. (Initialize n as 15). Start with 0 and 1 as the first two numbers. The next number is created by adding the previous two numbers. Thus, the series would go like this: 0 1 1 2 3 5 8 13 21 34 55 89 144 233 377.

    Note: Due to overflow, it is okay for your program to instead print 0 1 1 2 3 5 8 13 21 34 55 89 16 105 121

    For reference, answer to Homework 2 Question 11:

```
n = 15
counter = 1
a = 0
b = 1
print(a)
while(counter<n):
    print(b)
    temp = b
    b = b + a
    a = temp
    counter = counter + 1
```

    For reference, answer to Homework 2 Question 11 with a hard coded solution:

```
print(0)
print(1)
print(1)
print(2)
```

```
print(3)
print(5)
print(8)
print(13)
print(21)
print(34)
print(55)
print(89)
print(144)
print(233)
print(377)
```

For reference, answer to this question with a hard coded solution:

```
; this will earn you 0 points because it is hard coded
; this is provided to give you a sense of what we are
;    looking for and how the Output LCD should behave
;    when we run your code
; this also demonstrates how to print to Output LCD
ldi r16, 255 ; set r16 to all high bits
out 17, r16  ; all high bits IO 17 for output
ldi r17, 0   ; 1st number
out 18, r17  ; print 1st number
ldi r17, 1   ; 2nd number
out 18, r17  ; print 2nd number
ldi r17, 1   ; 3rd number
out 18, r17  ; print 3rd number
ldi r17, 2   ; 4th number
out 18, r17  ; print 4th number
ldi r17, 3   ; 5th number
out 18, r17  ; print 5th number
ldi r17, 5   ; 6th number
out 18, r17  ; print 6th number
ldi r17, 8   ; 7th number
out 18, r17  ; print 7th number
ldi r17, 13  ; 8th number
out 18, r17  ; print 8th number
ldi r17, 21  ; 9th number
out 18, r17  ; print 9th number
ldi r17, 34  ; 10th number
out 18, r17  ; print 10th number
ldi r17, 55  ; 11th number
```

```
out 18, r17  ; print 11th number
ldi r17, 89  ; 12th number
out 18, r17  ; print 12th number
ldi r17, 144 ; 13th number
out 18, r17  ; print 13th number
ldi r17, 233 ; 14th number
out 18, r17  ; print 14th number
ldi r17, 121 ; 15th number, wrap around, 377-256=121
out 18, r17  ; print 15th number
halt         ; end program
```

Sample answer:

```
; r16 used for n
; r17 used for counter
; r18 used for a
; r19 used for b
; r20 used to initialize IO 17
; r21 used for temp
ldi r16, 15     ; n = 15
ldi r17, 1      ; counter = 1
ldi r18, 0      ; a = 0
ldi r19, 1      ; b = 1
ldi r20, 255    ; set r20 to all high bits
out 17, r20     ; print(a), all high bits IO 17 for output
out 18, r18     ; print(a), actually display a
whileLoop:      ; while(counter<n):, while loop label to jump to
cp r17, r16     ; while(counter<n), compare counter to n
brsh halt       ; while(counter<n), exit loop if counter >= n
out 18, r19     ; print(b)
mov r21, r19    ; temp = b
add r19, r18    ; b = b + a
mov r18, r21    ; a = temp
inc r17         ; counter = counter + 1
rjmp whileLoop ; end of while loop
halt:           ; halt, end of program
halt
```

15. Repeat Homework 2 Question 12, but this time use **assembly language** instead of Python, and go from 32 (due to overflow and wrap around) to 50 instead. **(6)**

Annotate your code with comments, it is worth point(s).

Use the browser-based simulator to write and submit.

You may have to write your own multiply by doing multiple adds.

You may have to write your own divide by using a quotient counter, and subtracting the divisor from the dividend while the dividend is greater than the divisor.

For reference, Homework 2 Question 12:

Say you wanted to print out the Celsius equivalent for all integer Fahrenheit temperatures from **32** degrees F to 50 degrees F. Write a program to print out this conversion information. The pseudocode for implementing this is given below.

The equation for converting Fahrenheit (F) to Celsius (C) is: $C = (F - 32) * \frac{5}{9}$

i. Set F's initial value to **32** (lower bound)
ii. While F is less than or equal to 50 (upper bound)
iii. Convert Fahrenheit to Celsius.
iv. Print the number of degrees in Fahrenheit
v. Print the number of degrees in Celsius
vi. Increment F

For reference, answer to Homework 2 Question 12:

```
F = 32
while(F <= 50):
    C = (F - 32) * 5 / 9
    print(F)
    print(C)
    F = F + 1
```

Sample output:

```
32
0
33
0
34
1
35
1
36
2
37
2
38
3
39
```

```
3
40
4
41
5
42
5
43
6
44
6
45
7
46
7
47
8
48
8
49
9
50
10
```

Sample answer:

```
; r16 is used for F
; r17 is used to  initialize IO 17
; r18 is used for F-32
; r19 is used for (F-32)*5
; r20 is used for C, (F-32)*5/9
; r21 is used for 51, since cpi is removed
; r22 is used for 9, since cpi is removed
ldi r21, 51     ; r21 = 51 (constant)
ldi r22, 9      ; r22 = 9 (constant)
ldi r16,32      ; F = 32
ldi r17, 255    ; set r16 to all high bits
out 17, r17     ; all high bits IO 17 for output
whileLoop:      ; label for while(F <= 50):
; cpi r16,51    ; while(F < 51):, compare F to 51
cp r16, r21     ; while(F < 51):, compare F to 51
brsh halt       ; while(F < 51):, break loop and halt if F >= 51
mov r18, r16    ; r18 = F
```

```
subi r18, 32    ; r18 = F - 32
ldi r19, 0      ; r19 = 0, could have optimize multiplication better
add r19, r18    ; r19 = (F - 32) * 1
add r19, r18    ; r19 = (F - 32) * 2
add r19, r18    ; r19 = (F - 32) * 3
add r19, r18    ; r19 = (F - 32) * 4
add r19, r18    ; r19 = (F - 32) * 5, our dividend
ldi r20, 0      ; r20 = 0, this is our quotient counter
divideLoop:     ; label to loop while dividend >= divisor
; cpi r19, 9    ; dividend >= 9 ?
cp r19, r22     ; dividend >= 9 ?
brsh incrementQuotient ; if so, then branch to incrementQuotient
; else C = (F - 32) * 5 / 9
out 18, r16     ; print(F)
out 18, r20     ; print(C)
inc r16         ; F = F + 1
rjmp whileLoop ; end of while loop, jump back to check condition again
incrementQuotient: ; label to inc quotient and subtract from dividend
subi r19, 9     ; subtract divisor, 9, from dividend
inc r20         ; increment quotient counter
rjmp divideLoop; end of incrementQuotient
; jump to check dividend >= divisor again
halt:           ; label to jump to if F >= 51
halt            ; halt to end program
```