Name: ______________________________

netID: ____________________@wisc.edu

Exam 2

CS/ECE 252 Section-2 (MWF 11:00)

Monday, October 26

Write legibly, especially for your name and netID.

Read all questions carefully.

Showing work for number calculations is optional for partial credit if the final answer wrong.

Annotating/commenting code for code snippets is required for full credit.

Good luck./! <- this is the version number. “Good luck.” is version A. “Good luck!” is version B. Version C is the Exam 2 review worksheet.

1. Convert decimal numbers to/from ~~un~~signed magnitude and 2's complement using 8-bits by filling out the table below. Ignore the underscores, they are just there for readability. **(3)**

Decimal	**Signed Magnitude**		**2's complement**	
	Binary	**Hexadecimal**	**Binary**	**Hexadecimal**
(-1) * (32 + 16 + 8 + 2) = -58 (unsigned) 186	0b1011_1010	0xBA	**Inv and add 1 -> 1100_0110 (unsigned) 0_1011_1010**	**0xC6 (unsigned) 0xBA**
-128 + 64 + 8 + 4 + 1 = -51	removed		0b1100_1101	0xCD

Decimal	**Signed Magnitude**		**2's complement**	
	Binary	**Hexadecimal**	**Binary**	**Hexadecimal**
(-1) * (32 + 8 + 2 + 1) = -43 (unsigned) 171	0b1010_1011	0xAB	**Inv and add 1 -> 1101_0101 (unsigned) 0_1010_1011**	**0xD5 (unsigned) 0xAB**
-128 + 64 + 8 + 4 + 2 = -50	removed		0b1100_1110	0xCE

Decimal	**Signed Magnitude**		**2's complement**	
	Binary	**Hexadecimal**	**Binary**	**Hexadecimal**
-113	**-113 = (-1) * (64 + 32 + 16 + 1) = 0b1111_0001**	**0xF1**	**113 = 64 + 32 + 16 + 1 = 0b0111_0001 -113 = 0b1000_1111**	**0x8F**
(-1) * (64 + 8 + 4 + 2) = -78	0b1100_1110	0xCE	removed	
-128 + 32 + 16 + 8 + 2 = -70	removed		0b1011_1010	0xBA

2. Convert 83.625 / 52.375 / 121.78125 from decimal to unsigned fixed point notation. **(2)**

```
83/2 = 41 R1
41/2 = 20 R1
20/2 = 10 R0
10/2 =  5 R0
 5/2 =  2 R1
 2/2 =  1 R0
 1/2 =  0 R1
0.625/0.5000 = 1 R0.125
0.125/0.2500 = 0 R0.125
0.125/0.1250 = 1 R0
83.625 = 1010011.101

52/2 = 26 R0
26/2 = 13 R0
13/2 =  6 R1
 6/2 =  3 R0
 3/2 =  1 R1
 1/2 =  0 R1
0.375/0.50000 = 0 R0.375
0375/0.25000 = 1 R0.125
0.125/0.12500 = 1 R0.0
52.375 = 110100.011

121/2 = 60 R1
 60/2 = 30 R0
 30/2 = 15 R0
 15/2 =  7 R1
  7/2 =  3 R1
  3/2 =  1 R1
  1/2 =  0 R1
0.78125/0.50000 = 1 R0.28125
0.28125/0.25000 = 1 R0.03125
0.03125/0.12500 = 0 R0.03125
0.03125/0.06250 = 0 R0.03125
0.03125/0.03125 = 1 R0
121.78125 = 1111001.11001
```

3. What is the decimal value of the floating point number ECE00000 / ACE00000 / 0ECE2520? You do not need to compute the exact answer, just set up the formula in decimal. **(3)**

```
ECE00000
   = 1110_1100_1110_0000_0000_0000_0000_0000
   = 1_11011001_11000000000000000000000
   sign = 1
   exponent =  128 + 64 + 16 + 8 + 1 = 217
   fraction = .11000000000000000000000 = 0.5 + 0.25 = 0.75
```

$(-1)^{sign} \times 2^{exponent-127} \times (1 + fraction)$

$= (-1)^1 \times 2^{217-127} \times (1 + 0.75)$ `# getting to this step is full credit`

$= (-1) \times 2^{90} \times 1.75$

$= -2.166E27$

```
ACE00000
   = 1010_1100_1110_0000_0000_0000_0000_0000
   = 1_01011001_11000000000000000000000
   sign = 1
   exponent = 64 + 16 + 8 + 1 = 89
   fraction = .11000000000000000000000 = 0.5 + 0.25 = 0.75
```

$(-1)^{sign} \times 2^{exponent-127} \times (1 + fraction)$

$= (-1)^1 \times 2^{89-127} \times (1 + 0.75)$ `# getting to this step is full credit`

$= (-1) \times 2^{-38} \times 1.75$

$= -6.366E-12$

```
0ECE2520
   = 0000_1110_1100_1110_0010_0101_0010_0000
   = 0_00011101_10011100010010100100000
   sign = 0
   exponent = 16 + 8 + 4 + 1 = 29
   fraction = .10011100010010100100000 ≈ 0.5 + 0.0625 + 0.03125 +
        0.015625 + ... = 0.61051
```

$(-1)^{sign} \times 2^{exponent-127} \times (1 + fraction)$

$\approx (-1)^0 \times 2^{29-127} \times (1 + 0.61051)$ `# getting to this step is full credit`

$= 1 \times 2^{-98} \times 1.61051$

$= 5.08187E-30$

4. Assuming two's complement notation, perform the following calculations using 8-bits two's complement notation. Express your answer in two's complement binary. Due to time constraints, it is not required to convert your answer to decimal to check. **(0.5 + 1.5)**

```
    11  11
a.  10101010   (-86)
   +01100110  +(102)
```

```
    00010000   (16)

b.  10101010   (-86)
   -10011101 -(-99)
    00001101   (13)
```

or equivalently by taking the 2's complement and then adding

```
    11   1
    10101010 (-86)
   +01100011 +(99)
    00001101  (13)

    1111
a.  10101010 (-86)
   +01011001 +(89)
    00000011   (3)

b.  11101010   (-22)
   -10100110 -(-90)
    01000100   (68)
```

or equivalently by taking the 2's complement and then adding

```
    1111 1
    11101010 (-22)
   +01011010 +(90)
    01000100  (68)

    1111
a.  10101010 (-86)
   +11111001 +(-7)
    10100011 (-93)

b.  11001100   (-52)
   -10110110 -(-74)
    00010110   (22)
```

or equivalently by taking the 2's complement and then adding

```
    1111 1
    11001100 (-52)
   +01001010 +(74)
    00010110  (22)
```

5. Fill in the blanks to convert the following piece of code from Python to AVR assembly. **(4)**

```
n = 1
s = 0
b = 0
while(n<=5):
   s = s + n
   if(s<3):
      b = b + 1
   else:
      b = b + 2
   n = n + 1

; r16 is used for n
; r17 is used for s
; r18 is used for b
; up to you to figure out the rest of the register usage
ldi r16, 1             ; r16 = n = 1
ldi r17, 0             ; r17 = s = 0
ldi r18, 0             ; r18 = b = 0

whileLoopBegin:        ; label for while(n<=5):
ldi r19, 6___________; which immediate value to load for compare?
cp r16, r19__________; which 2 registers to compare?
brsh whileLoopEnd____; which label to jump to? or 9
add r17, r16           ; s = s + n
ldi r20, 3             ; load 3 for compare immediate
cp r17, r20            ; if(s>=3):
brsh elseBlockBegin    ;    then jump to else block
inc r18______________; which register to increment?
rjmp ifElseEnd_______; which label to jump to?; 1

elseBlockBegin:        ; label for else block
subi r18, -2_________; which operation?
; and if that operation takes 2 operands, then what is the second operand?

ifElseEnd:             ; label for end of if-else block
inc r16                ; n = n + 1
rjmp whileLoopBegin    ; jump to check condition of while loop again

whileLoopEnd:          ; label for end of while loop
halt                   ; halt to end program
```

```
n = 2
s = 0
b = 0
while(n<=6):
   s = s + n
```

```
    if(s<3):
        b = b + 1
    else:
        b = b + 3
    n = n + 1

; r16 is used for n
; r17 is used for s
; r18 is used for b
; up to you to figure out the rest of the register usage
ldi r16, 2           ; r16 = n = 2
ldi r17, 0           ; r17 = s = 0
ldi r18, 0           ; r18 = b = 0

whileLoopBegin:      ; label for while(n<=~~5~~6):
ldi r19, 7___________; which immediate value to load for compare?
cp r16, r19__________; which 2 registers to compare?
brsh whileLoopEnd____; which label to jump to?
add r17, r16         ; s = s + n
ldi r20, 3           ; load 3 for compare immediate
cp r17, r20          ; if(s>=3):
brsh elseBlockBegin  ;     then jump to else block
inc r18______________; which register to increment?
rjmp ifElseEnd_______; which label to jump to?; or 1

elseBlockBegin:      ; label for else block
subi r18, -3_________; which operation?
; and if that operation takes 2 operands, then what is the second operand?

ifElseEnd:           ; label for end of if-else block
inc r16              ; n = n + 1
rjmp whileLoopBegin  ; jump to check condition of while loop again

whileLoopEnd:        ; label for end of while loop
halt                 ; halt to end program
```

```
x = 20
y = 30
n = 1
s = 0
while(n<5):
    s = s + n
    if(x<=y):
        x = x + y
    else:
        y = x - y
    n = n + 1
```

```
; r16 is used for x
; r17 is used for y
; r18 is used for n
; r19 is used for s
; up to you to figure out the rest of the register usage
ldi r16, 20          ; r16 = x = 20
ldi r17, 30          ; r17 = y = 30
ldi r18, 1           ; r18 = n = 1
ldi r19, 0           ; r18 = s = 0

whileLoopBegin:      ; label for while(n<5):
ldi r22, 5           ; which immediate value to load for compare?
cp r18, r22          ; which 2 registers to compare?
brsh whileLoopEnd    ; which label to jump to?
add r19, r18         ; s = s + n
mov r20, r17         ; which register to move/copy to r20?
inc r20
cp r16, r20          ; which 2 registers to compare?
brsh elseBlockBegin  ;    if x > y, then jump to elseBlockBegin
add r16, r17         ;    else fall through and execute if block; x = x + y
rjmp ifElseEnd       ; jump to end of if-else block

elseBlockBegin:      ; label for else block
mov r21, r17
mov r17, r16
sub r17, r21         ; some statement(s)

ifElseEnd:           ; label for end of if-else block
inc r18              ; n = n + 1
rjmp whileLoopBegin  ; jump to check condition of while loop again

whileLoopEnd:        ; label for end of while loop
halt                 ; halt to end program
```

6. Write 4 code snippets to finish the AVR assembly program that eor (exclusive-or) the first 10 numbers in the RAM address labeled "*array*", and print the final result to output LCD. For simplicity, you can assume that all 10 numbers are stored in the same HI memory pointer. In other words, the LO memory will not overflow when incrementing it to loop through all 10 number.

   Write 4 code snippets to finish the AVR assembly program that add (ignoring overflow) the first 10 numbers in the RAM address labeled "*array*", and print the final result to output LCD. For simplicity, you can assume that all 10 numbers are stored in the same HI memory pointer. In other words, the LO memory pointer will not overflow when incrementing it to loop through all 10 number. **(4)**

   You can initialize the values at label "array" to whatever you want.
   Annotate your code with comments for full credit.

```
.byte(array) 1,2,5,-7,-4,-6,-9,8,1,-3 ; assembler directive to load RAM
ldi r16, 10              ; r16 is counter for first 10 numbers
```

```
; write a code snippet to initialize answer
; YOUR_1st_CODE_SNIPPET_BELOW_HERE
ldi r17, 0              ; r17 holds the results

; load the address of the "array" in X (X = r26 + 256*r27)
ldi r27, hi8(array)     ; set up HI memory pointer
ldi r26, lo8(array)     ; set up LO memory pointer

loopBegin:              ; label to loop
; write code snippet to complete computation portion of loop
; YOUR_2nd_CODE_SNIPPET_BELOW_HERE
ld r18, X               ; load from X to r18
eor r17, r18            ; eor or add, depending on version

inc r26                 ; increment LO memory pointer
dec r16                 ; decrement counter
; write code snippet to finish flow control of loop
; YOUR_3rd_CODE_SNIPPET_BELOW_HERE
; ldi & cp can also be used here
; SAMPLE SOLUTION 1
brne loopBegin          ; continue loop if not done with all 10 numbers
                        ; else fall through and print output to LCD
; SAMPLE SOLUTION 2
; breq printToOutputLCD; break loop if done with all 10 numbers
; rjmp loopBegin        ; else go back to loop again

; SAMPLE SOLUTION 3
; ldi r20, 0; break loop if done with all 10 numbers
; cp r16, r20
; brne loopBegin        ; continue loop if not done with all 10 numbers
                        ; else fall through and print output to LCD

; SAMPLE SOLUTION 4
; ldi r20, 0; break loop if done with all 10 numbers
; cp r16, r20
; breq printToOutputLCD; break loop if done with all 10 numbers
; rjmp loopBegin        ; else go back to loop again


printToOutputLCD:       ; label to jump to after done with all 10 numbers
; write code snippet to display the answer to output LCD
; YOUR_4th_CODE_SNIPPET_BELOW_HERE
ldi r18, 255            ; set r18 to all high
out 17, r18             ; set IO reg 17 to all high for output
out 18, r17             ; display value of r17 to output LCD
```

7. Fill in the contents of the register file (only r0-r3 in this case) and stack (including relevant addresses) after the following code finishes executing: **(2 for registers + 1 for memory)**

```
push r1
push r2
push r3
pop r0
pop r1
pop r3
add r1 r3
```

Structures before execution:

Register File	
r0	0
r1	1
r2	2
r3	4

Stack Pointer
399

RAM	
397	0
398	0
399	0
400	10
401	20
402	30
403	40

Fill in your answers below, showing the structures after execution:

Register File	
r0	4
r1	3
r2	2
r3	1

Stack Pointer
399

RAM	
397	4
398	2
399	1
400	10
401	20
402	30
403	40

8. Explain how function (subroutine) call and return works in AVR. **(2)**
rcall first pushes pc+1 on to the stack, then sets the PC to the PC+offset.
Ret pops the PC+1 off the stack to the PC.
1 point for control flow behaviour. 1 point for pushing/popping PC+1

Consider the code on the handout for problems 9-11

9. Describe what the function is doing **(1)**

This function is doing a divide operation (iteratively subtracting and counting how many times it has subtracted until below the divisor)

10. Is this piece of code using caller or callee saved registers? **(1)**

Callee saved

11. How would you modify the code to use the other method of saving registers? You can either describe it or write the code snippet. **(2)**

The push and pop of r25 in the function should be removed. r25 should be pushed before rcall, and popped after.

12. For this question, we will be asking you to write the missing function in the following piece of code. This function is meant to reverse the contents of an array, that is, if an array was [4,5,6], the output of the function would be [6,5,4]. For writing this function, there are a few requirements:

- Any registers or values that are modified outside of the array should be restored before the function returns
- The function assumes that X is set prior to the function being called, and is set to the location of the first element of the array
- The function assumes that r16 contains the *length* of the array, that is, the amount of elements in the array.

Hint: It might be beneficial to think of the function in two parts: reading the contents of the array, and then storing the array back in memory in reverse order.

(2 pts extra credit)

```
ldi r31, 136
out 61, r31
ldi r31, 19
out 62, r31
rjmp program

function:
;assume the size of the array is stored in r16
;and that X is set to the beginning of the array
[The code for your function]

program:
;Store the values 4, 5, and 6 in addresses 500, 501, and 502
ldi r27, 1
ldi r26, 244
ldi r25, 4
st X, r25
inc r26
ldi r25, 5
st X, r25
inc r26
ldi r25, 6
st X, r25
dec r26
dec r26
;Store the size of the array
ldi r16, 3
rcall function
```

Sample solution:

```
ldi r31, 136
out 61, r31
ldi r31, 19
out 62, r31
rjmp program
function:
;assume the size of the array is stored in r16
;and that X is set to the beginning of the array
push r17
push r18
push r19
ldi r18, 0
mov r19, r26
begin_first_while:
cp r18, r16
breq done_first_while
ld r17, X
push r17
inc r18
inc r26
rjmp begin_first_while
done_first_while:
mov r26, r19
ldi r18, 0
begin_second_while:
cp r18, r16
breq done_second_while
pop r17
st X,r17
inc r18
inc r26
rjmp begin_second_while
done_second_while:
mov r26, r19
pop r19
pop r18
pop r17
ret
program:
;Store the values 4, 5, and 6 in addresses 500, 501, and 502
ldi r27, 1
ldi r26, 244
ldi r25, 4
```

```
st X, r25
inc r26
ldi r25, 5
st X, r25
inc r26
ldi r25, 6
st X, r25
dec r26
dec r26
;Store the size of the array
ldi r16, 3
rcall function
```