Test 1 Review

Test Organization

- Paper/pen test
- Coding test:
    - no conceptual questions
    - no number conversions to and from binary (calculators aren't allowed)
    - no questions asking what is the output of this code
    - no questions asking fill the tables, … etc.
    - All questions will be "write an assembly code that does …"
    - You don't have to comment your code. It's a good idea to do at the end.
    - You will not going to be graded on code efficiency
    - You will be penalized for wrong syntax
        - Instruction names
        - Proper operands: which registers are permitted

Question Types:

- Copy an array into another array
- Apply an arithmetic formula on variables/array
- Apply logical condition(s) to variables/arrays

Material Review

Defining Symbolic Constants

```
.EQU  name = value
```

Data Transfer:

1. Register to Register copy:
    - MOV
    - MOVW
2. Immediate to Register copy:
    - Copies a constant value into a register
    - LDI
3. (Memory) to Register copy:
    - Load constants in program memory into a register
    - Typical code:

```
LDI Zh, HIGH(2*varname)
LDI Zl, LOW(2*varname)
LPM reg, Z              ;Z -> constant
```

    - Can use +/- index
    - When loading word/doubleword: first byte is LSB and last byte is MSB
        - Example: load a word into R3:R2 (MSB:LSB):

```
LPM R2, Z+      ;LSB
LPM R3, Z       ;MSB
```

        - Example: load a doubleword into R3:R2:R1:R0 (MSB: : :LSB):

```
LPM R0, Z+      ;LSB
LPM R1, Z+
LPM R2, Z+
LPM R3, Z       ;MSB
```

4. Register to (Memory) copy:
    - Store a byte in data memory
    - Typical code:

```
LDI Xh, HIGH(varname)
LDI Xl, LOW(varname)
ST X, reg               ;X-> variable
```

    - Can use X, Y, Z as index
    - Can use +/- on index
    - When storing word/doubleword: first byte is LSB and last byte is MSB
        - Example: Store the word in R3:R2 (MSB:LSB):

```
ST X+, R2       ;LSB
ST X, R3        ;MSB
```

        - Example: load a doubleword into R3:R2:R1:R0 (MSB: : :LSB):

```
ST X+, R0       ;LSB
ST X+, R1
ST X+, R2
ST X+, R3       ;MSB
```

Arithmetic:

1. Adding:
    - Byte - Byte: `ADD reg, reg`
    - Word - Word:
```
ADD reg3, reg1       ;reg3:reg2 + reg1:reg0
ADC reg2, reg0
```
    - Word + Imm : `ADIW reg3:reg2, K    ;reg3:reg2 + K`
    - Byte + Imm: `SUBI reg, -K         ;reg = reg -(-K) = reg + K`

2. Subtracting:
    - Bytes - Byte, Word – Imm, Byte – Imm: `SUB, SBIW, SBI`
    - Word – Word:
```
SUB reg3, reg1       ;reg3:reg2 + reg1:reg0
SBC reg2, reg0
```
3. Multiplying Integers:
    - MUL (UU), MULS (SS), MULSU (SU)
    - Byte * Byte => Word (R1:R0)

4. Fractional Multiplication:
    - FMUL (UU), FMULS (SS), FMULSU (SU)
    - Range of 1.7 Fraction multiplication:
        - Unsigned: 0 to <2
        - Signed: -1 to <1
    - Q7() used to convert fractions to 1.7 Format.
        - Must use a decimal inside argument
        - Only does fractions (-1.0 < fraction < 1.0). Can't convert integer parts.
    - Integer Division as fractional multiplication:

```
LDI reg1, M
LDI reg2, Q7(1./N)
FMUL/FMULS/FMULSU reg1, reg2   ; M/N
```

5. Loops:
    - Use the following code for loops:

```
      LDI  reg, count      ;reg = counter
      …
LX:   …
   DEC reg           ;reg--
   TST reg
   BRNE LX
```

    - Always use loops with arrays

6. Jumps/Branches
    - No theoretical questions about flags
    - Must know how flags and stacks work to be able to use them in a code
    - Unconditional jumps: JMP, IJMP, RJMP
        - RJMP is typically the only one needed
    - Conditional branches:
        - Based on specific flags
        - Following a comparison:
            - CP: compares 2 regs

        - CPI: compares a reg with a cons/imm
        - TST: compares a re with 0
    - Following signed comparison:
        - BREQ/BRNE, BRLT, BRGE
        - BRLE/BRGT are no directly supported
            - We can use 2 branches to create
            - BRLE can be implemented as:
              BREQ label
              BRLT label
            - BRGT can be implemented as:
              BREQ skip ;PC+2
              BRGE label
    - Following unsigned comparison:
        - BREQ/BRNE, BRLO, BRSH
        - BRLE/BRHI are no directly supported
            - We can use 2 branches to create them
    - When trying to code a condition it is typically better to code the negative (false) condition
        - Condition Negative

==	!=
>	<=
<	>=
>=	<
<=	>

Practice:

- Given an array add all the elements (cumulative sum) in an array and store it in a variable
- Given an array determine the number of elements that are odd
- Given an x and y, calculate x^y
- Given an array create the mirror of the array, e.g. {1,2,3,4,5} => {1,2,3,4,5,4,3,2,1}
- Implement X/Y as successive division:

```
NUMER = X, DENOM=Y, QUOT=0, REM=0, count =0
While (NUMER >DENOM)
{       NUMER = NUMER – DENOM
        count++}
QUOT = count, REM = NUMER
```

- Given an array, determine the min/max value of the array.
- A signed 1.7 fractional number,X, is to be rounded down to the nearest integer. (Hint: if X < 0, => floor(X)=-1; X>0 => floor(X)=0).