HW 2 Solution Key

1. Given an unsigned byte integer constant x.
   y = 2x - 5
   Write a program that performs the calculation, then stores it in the variable y.

```
        .CSEG
        LDI ZH, HIGH(VARX*2)      ;be careful X and Y are reserved words
        LDI ZL, LOW(VARX*2)

        LPM R16, Z                ;R16 = X

        LDI R17, 2
        MUL R16, R17              ;X*2
        MOV R24, R0
        MOV R25, R1
        SBIW R25:R24, 5

        LDI XH, HIGH(VARY)
        LDI XL, LOW(VARY)

        ST X+, R24
        ST X, R25

VARX:   .DB     255

        .DSEG
VARY:   .BYTE   2
```

2. Given 2 byte integer constants: x (signed) and y (unsigned).
   z = x^2 - 2xy.
   Write a program that performs the calculation, then stores it in the variable z.

```
        .CSEG
        LDI ZH, HIGH(VARX*2)    ;be careful X and Y are reserved words
        LDI ZL, LOW(VARX*2)
        LPM R16, Z              ;R16 = X

        LDI ZH, HIGH(VARY*2)
        LDI ZL, LOW(VARY*2)
        LPM R17, Z              ;R17 = Y

        MULS R16, R16           ;X^2
        MOV R2, R0
        MOV R3, R1              ;R3:R2 = X^2

        MULSU R16, R17          ;X*Y => R1:R0

        SUB R2, R0
        SBC R3, R1              ;R5:R4 - R3:R2
        SUB R2, R0
        SBC R3, R1              ;R5:R4 - R3:R2 - R3:R2

        LDI XH, HIGH(VARZ)
        LDI XL, LOW(VARZ)

        ST X+, R2
        ST X, R3

VARX:   .DB     -10
VARY:   .DB     250
        .DSEG
VARZ:   .BYTE   2
```

3. Given an unsigned integer x.
   The variable even = 1 if x is even, otherwise even = 0.
   Write a program that computes even.

Simple Algorithm:

1. Shift X to the right once. This will put the least significant bit into the CF. Odd numbers will have 1 in the LSB that will be shifted into the CF.	Example odd: X = 15 = 0b00001111 X >> 1 = 00000111, CF = 1	Example even: X=14=0b00001110 X >> 1 = 00000111, CF = 1
2. Add with carry 2 zeros to get the carry bit out into a register.	0 + 0 + CF = 1	0 + 0 + CF = 0
3. Complement the LSB by Ex-ORing with 1. Since CF=1 means odd, when need to complement that value (Ex-ORing with 1 will complement bit).	1 Ex-OR 1 = 0	1 Ex-OR 1 = 1

```
        .CSEG
        LDI ZH, HIGH(VARX*2)    ;be careful X and Y are reserved words
        LDI ZL, LOW(VARX*2)
        LPM R16, Z              ;R16 = X

        LSR R16                 ;CF = 1 if odd
        CLR R16
        ADC R16, R16            ;R16 = CF

        SET R17                 ;R17 = 1
        EOR R16, R17            ;Complement the CF

        LDI XH, HIGH(even)
        LDI XL, LOW(even)

        ST X+, R16

VARX:   .DB     10

        .DSEG
even:   .BYTE   1
```

4. Code the following C code into AVR Assembly:
   for (i=0, sum=0; i < 10; i++)
      sum += i*i;

   Treat **i** and **sum** as variables.

   sum is a data memory WORD variable.

```
            CLR R16             ;R16 = i
            CLR R17             ;R18:R17 = SUM
            CLR R18

FOR:        CPI R16, 10
            BRSH ENDFOR

            MUL R16,R16         ;i^2 => R1:R0

            ADD R17, R0
            ADC R18, R1         ;SUM += i^2

            INC R16             ;I++
            RJMP FOR

ENDOR:      LDI XH, HIGH(sum)
            LDI XL, LOW(sum)
            ST X+, R17
            ST X, R18
```

5. Code the following C code into AVR Assembly:

```
switch (a): {
    case(1): option = 1; break;
    case(10): option = 2; break;
    case(100): option = 3; break;
    default: option = 0;
}
```

Treat **a** and **option** as variables.

```
                LDI XH, HIGH(a)
                LDI XL, LOW(a)
                LD R16, X               ;R16 = A

DEFAULT:        CLR R17                 ;R17 = OPTION = 0

CASE1:          CPI R16, 1
                BRNE CASE2
                LDI R17, 1              ;OPTION = 1
                RJMP ENDSWITCH
CASE2:          CPI R16, 10
                BRNE CASE2
                LDI R17, 2              ;OPTION = 2
                RJMP ENDSWITCH
CASE3:          CPI R16, 100
                BRNE CASE2
                LDI R17, 3              ;OPTION = 3
ENDSWITCH:
                LDI XH, HIGH(option)
                LDI XL, LOW(option)
                ST X, R16
```

6. **Grade** is an <u>unsigned</u> byte stored in the program memory (PM). **Letter** is a byte stored in the data memory. **Letter** = "A", "B", "C", "D", or "F" based on the value in **Grade**.
(89 < A; 79 < B < 90; 69 < C < 80; 59 < D < 60; F < 60)

```
                LDI ZH, HIGH(2*GRADE)
                LDI ZL, LOW(2*GRADE)
                LPM R16, Z              ;R16=GRADE

AA:             CPI R16, 90
                BRLO BB
                LDI R17, 'A'            ;R17=LETTER= 'A'
                RJMP ENDIF

BB:             CPI R16, 80
                BRLO CC
                LDI R17, 'B'            ;R17=LETTER= 'B'
                RJMP ENDIF

CC:             CPI R16, 70
                BRLO DD
                LDI R17, 'C'            ;R17=LETTER= 'C'
                RJMP ENDIF

DD:             CPI R16, 60
                BRLO FF
                LDI R17, 'D'            ;R17=LETTER= 'D'
                RJMP ENDIF

FF:             LDI R17, 'F'            ;R17 = LETTER = 'F'

ENDIF:
                LDI ZH, HIGH(LETTER)
                LDI ZL, LOW(LETTER)
                ST Z, R17               ;STORE LETTER GRADE
```

7. **Hex_ascii** is an unsigned ASCII byte that can be one the following values: '0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'A', 'B', 'C', 'D', 'E', 'F'. Note that '0'-'9' = 0x30-0x39 and 'A'-'F' = 0x41-0x46. Write a program that will convert Hex_ascii from ASCII to a number (0-15) and store it in **digit**. Hint: simply subtract 0x30 for ASCII '0'-'9', and subtract 0x37 for ASCII 'A'-'F'.

```
                LDI ZH, HIGH(2*Hex_ascii)
                LDI ZL, LOW(2* Hex_ascii)
                LPM R16, Z              ;R16=HEX_ASCII

                CPI R16, 'A'
                BRSH ADD37              ;"A"-"F"
                ADD R16, 0x30           ;"0"-"9"
                RJMP STORE

ADD37:          ADD R17, 0x37

STORE:          LDI ZH, HIGH(digit)
                LDI ZL, LOW(digit)
                ST Z, R16               ;STORE DIGIT
```

8. **Num** is an <u>unsigned</u> byte stored in the program memory (PM). Write a program that will check if **Num** is divisible by 4. Idea: keep subtracting by 4 until what is left is < 4. If what is left = 0 then it is divisible, otherwise it is not divisible. The variable **div4** = 1 if **Num** is divisible by 4.

```
                LDI ZH, HIGH(2*NUM)
                LDI ZL, LOW(2*NUM)
                LPM R16, Z              ;R16=R16

                LDI R17, 0              ;DIV4 = 0 BY DEFAULT

WHILE:          CPI R16, 4              ;N>4
                BRLO ENDW

                SUBI R16,4

                RJMP WHILE
ENDW:
                CPI R16, 0              ;REMAINDER = 0 ?
                BRNE STORE
                LDI R17, 1              ;DIV4 = 1

STORE:          LDI ZH, HIGH(div4)
                LDI ZL, LOW(div4)
                ST Z, R17               ;STORE DIGIT
```