Coding Exercise

- `Xarr` is stored in the data memory. Write a program to reverse the order of array values in `Xarr` without using the stack and using the stack. Assume the length of the array is stored in `len`.

```
            .DSEG
Xarr   .BYTES 100
.EQU   len = 100

            .CSEG
            ...

            LDI XH, HIGH(Xarr)              ;INDEX POINTS TO THE BEGIN
            LDI XL, LOW(Xarr)

            LDI YH, HIGH(Xarr+len-1)        ;INDEX POINTS TO THE END
            LDI YL, LOW(Xarr+len-1)

            LDI R16, len/2                        ;COUNTER = length/2

L1:         LD R17, X                                   ;R17 = Xarr[I]. I starts at 0  -> len/2
            LD R18, Y                                   ;R18 = Xarr[J]. J starst at 99 -> len/2

            ST X, R18                                   ;SWAP
            ST Y, R17

            INC X
            DEC Y

            DEC R16
            TST R16
            BRNE L1
```

```
            .DSEG
Xarr   .BYTES 100
.EQU   len = 100

            .CSEG
            ...

            LDI XH, HIGH(Xarr)  ;INDEX
            LDI XL, LOW(Xarr)

            LDI R16, len              ;COUNTER

L1:         LD R17, X+                      ;R17=Xarr[I], X=I, X++
            PUSH R17                        ;STORE IN STACK

            DEC R16
            TST R16
            BRNE L1

            ;TOP OF STACK = Xarr[99]; BOTTOM OF STACK = Xarr[0]

            LDI XH, HIGH(Xarr)  ;INDEX START AT BEGINNING OF ARRAY
            LDI XL, LOW(Xarr)

            LDI R16, len              ;COUNTER

L2:         POP R17                                 ;POP THE TOP OF THE STACK
            ST X+, R17                      ;STORE AT Xarr[I], X=I, X++

            DEC R16
            TST R16
            BRNE L2
```

Temp/Local Variables

- Temporary storage on the stack can be used to create temp/local variables
- Steps:
  1. Copy: SP to an index register (X, Y, Z)
     - Use IN to copy
     - Copy SPH, SPL
  2. Add: Index = Index + variable size
  3. Access: Use ST and LD to access variable

```
   LDI XH, HIGH(Xmin)
   LDI XL, LOW(Xmin)
   LD R16, X           ;xmin

   LDI YH, HIGH(Xmax)
   LDI YL, LOW(Xmax)
   LD R17, X           ;xmax

   CP R16,R17
   BRGE DONE           ;R17>R16 => DO NOTHING

   ST X, R17           ;SWAP
   ST Y, R16

DONE:
```

```
LDI XH, HIGH(Xmin)
LDI XL, LOW(Xmin)
LD R16, X           ;xmin

LDI YH, HIGH(Xmax)
LDI YL, LOW(Xmax)
LD R17, Y           ;xmax

CP R16,R17
BRGE DONE           ;R17>R16 => DO NOTHING

IN Z, SP            ;Y = SP
DEC Z               ;CREATES A BYTE ON THE STACK

ST Z, R16           ;TEMP = R16
MOV R16,R17
LD R17, Z           ;R17 =
ST X, R16           ;SWAP
ST Y, R17
```

- Write a subroutine that adds 2 words. R2:R1=address of 1st word and R4:R3 = address of 2nd word, R6:R5 = address of sum.

```
        LDI R1,..
        LDI R2,..
        LDI R3,..
        LDI R4,..

        CALL SUBEX1
        ...


SUBEX2:

        ;PUSH ALL REGISTERS USED BY THIS SUB:
        PUSH R16
        PUSH R17
        PUSH R18
        PUSH R19
        PUSH ZH
        PUSH ZL
        PUSH YH
        PUSH YL
        PUSH XH
        PUSH XL

        MOV Xh,R2
        MOV Xl,R1        ;MEM[X] = WORD1

        MOV Yh,R4
        MOV Yl,R3        ;MEM[Y] = WORD2

        MOV ZH,R6        ;MEM[Z] = SUM
        MOV ZL,R5

        LD R16,X+
        LD R17,X         ;R17:R16 = WORD1

        LD R18,Y+
        LD R19,Y         ;R6:R5 = WORD2


        ADD R18,R16
        ADC R19,R17      ;R19:R18 = SUM

        ST Z+, R18       ;SUM -> MEM[Z]
        ST Z, R19

        ;POP ALL REGS USED BY SUB
        POP XL
        POP XH
        POP YL
        POP YH
        POP ZL
        POP ZH
        POP R19
        POP R18
        POP R17
        POP R16

        RET
```

- Write a subroutine that adds the elements of a byte array. The PM address of the array is in Z, length of the array in r16. The byte sum is returned in R0.

```
            LDI R1,..
            LDI R2,..
            LDI R3,..
            LDI R4,..

            CALL SUBEX1
            ...


SUBEX2:

            CLR R0              ;SUM=0

L1:         LPM R17, Z+
            ADD R0, R17

            DEC R16
            TST R16
            BRNE L1

            RET
```

- Write a subroutine that adds 2 words. R2:R1=address of 1st word and R4:R3 = address of 2nd word. Returns their word sum in R6:R5.

```
        LDI R1,..
        LDI R2,..
        LDI R3,..
        LDI R4,..

        CALL SUBEX1
        ...


SUBEX1:
        MOV R6,R2
        MOV R5,R1

        ADD R5,R3
        ADC R6,R2

        RET
```

- Write a subroutine that adds 2 words. R2:R1=address of 1st word and R4:R3 = address of 2nd word, R6:R5 = address of sum.

```
            CALL SUBEX1
            ...


SUBEX2:

            PUSH R16          ;THIS SUB WILL USE R16
            PUSH R17          ;THIS SUB WILL USE R17

            MOV Xh,R2
            MOV Xl,R1         ;MEM[X] = WORD1

            MOV Yh,R4
            MOV Yl,R3         ;MEM[Y] = WORD2

            LD R16,X+
            LD R17,X          ;R17:R16 = WORD1

            LD R5,Y+
            LD R6,Y           ;R6:R5 = WORD2


            ADD R5,R16
            ADC R6,R17 ;R6:R5 = WORD1 + WORD2

            POP R17           ;RETREIVE R16,R17 (REVERSE ORDER)
            POP R16

            RET
```

- Create a macro called ADD2W that works as follows:
  ADD2W result, num1, num2
  where result is a DM variable;
  num1, num2 are PM constants.

```
                .CSEG
.MACRO ADD2W                                ;@0=RESULT; @1=NUM1; @2=NUM2
                LDI ZH, HIGH(2*@1)
                LDI ZH, LOW(2*@1)
                LPM R16, Z+                 ;R17:R16 = NUM1
                LPM R17, Z

                LDI ZH, HIGH(2*@2)
                LDI ZH, LOW(2*@2)
                LPM R18, Z+                 ;R19:R18 = NUM2
                LPM R19, Z

                ADD R18,R16
                ADC R19,R17                 ;SUM = R19:R18

                LDI ZH, HIGH(@0)
                LDI ZL, LOW(@0)
                ST X+, R18
                ST X, R19

.ENDM



N1:             .DW 0x1234
N2:             .DW 0x5678
N3:             .DW 0x90AB

                ADD2W S, N1, N3
```

1. [30 pts] Given VarX is a signed byte and VarY is an unsigned byte defined in the program memory; Varz is defined in the data memory (they are not arrays). Write a program to perform the following calculation: $Varz = 2\times(VarX)^2 + (VarY/4)$.

```
        .CSEG
        LDI ZH, HIGH(2*VarX)
        LDI ZL, LOW(2*VarX)

        LPM R16, Z

        LDI ZH, HIGH(2*VarY)
        LDI ZL, LOW(2*VarY)

        LPM R17, Z

        LDI R19, Q7(1./4)

        FMUL R17, R19           ;Y/4
        MOV R20, R1
        CLR R21
        MULS R16,R16            ;R1:R0 = X^2

        ADD R0, R0              ;2*X^2
        ADC R1,R1

        ADD R0, R20
        ADC R1, R21

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

        ST X+, R0
        ST X, R1

        .DSEG
VarZ    .BYTE 2
```

2. [35 pts] Given ArrayX is a signed byte array defined in the program memory. ArrayY is a word array defined in the data memory. Write a program to copy the elements from ArrayX into ArrayY. The symbolic constant len contains the length of the 2 arrays. Use a loop.

```
        LDI R16, len

        LDI ZH, HIGH(2* ArrayX)
        LDI ZL, LOW(2* ArrayX)
        LDI R19, 1

        LDI XH, HIGH(ArrayY)
        LDI XL, LOW(ArrayX)

L1:     LPM R18, Z+
        MULS R18,R19


        ST X+, R0
        ST X+, R1

        DEC R16
        TEST R16
        BRE L1
```

3. [35 pts] Given ArrayX is an unsigned byte array defined in the program memory; Sum is a single unsigned word variable defined in the data memory. Write a program to sum all of the elements of ArrayX and store it in Sum. E.g. if ArrayX = {1,5,2,3,4,6} then Sum = 1+5+2+3+4+6 = 21. The symbolic constant len contains the length of the ArrayX.

```
        LDI R16, len
        CLR R17
        CLR R18
        CLR R19
        CLR R20

        LDI ZH, HIGH(2* ArrayX)
        LDI ZL, LOW(2* ArrayX)

L1:     LPM R17, Z+         ;X[I]

        ADD R18, R17        ;SUM = SUM + X[I]
        ADC R19, R20        ;SUM = R19:R18

        DEC R16
        TST R16
        BRNE L1

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

        ST X+, R18
        ST X, R19
```

Given an alphanumeric array of length N. Write a code to count how many alphabetic characters are in the array.

```
        LDI ZH, HIGH(2* alphanum)
        LDI ZL, LOW(2* alphanum)

        LDI R16, N

        CLR R18         ;CLEAR COUNTER = 0

L1:     LPM R17, Z+
        CPI R17, "A"
        BRLT skip
        CPI R17, "z"
        BREQ Alph
        BRGE skip

Alph:   INC R18

skip:   DEC R16
        TST
        BRNE L1


alphanum: .BYTE   "09ABCDefgh1234"
```

Convert the above code to a MACRO. Macro usage:
alphacount @0, @1, @2 ;@0 = count; @1 = array; @2 = length of array
Push/pop registers used inside the MACRO.

```
.MACRO    alphacount
          PUSH R16
          PUSH ZH
          PUSH ZL
          LDI ZH, HIGH(2* @1)
          LDI ZL, LOW(2* @1)

          LDI R16, @2

          CLR @0 ;CLEAR COUNTER = 0

L1:       LPM R17, Z+
          CPI R17, "A"
          BRLT skip
          CPI R17, "z"
          BREQ Alph
          BRGE skip
Alph:     INC @0
skip:     DEC R16
          TST
          BRNE L1
```

Given an integer array of length N. Write a MACRO to count how many integers are odd. Macro usage: oddcount @0, @1, @2 ;@0 = count; @1 = array; @2 = length of array

```
.MACRO      oddcount
            PUSH R16
            PUSH R17
            PUSH ZH
            PUSH ZL

            LDI ZH, HIGH(2* @1)
            LDI ZL, LOW(2* @1)

            LDI R16, @2

            CLR @0      ;CLEAR COUNTER = 0

L1:         LPM R17, Z+
            LSR R17     ;C = lsb. Lsb = 1 => Odd
            BRCC skip

            INC @0

skip:       DEC R16
            TST
            BRNE L1

            POP ZL
            POP ZH
            POP R17
            POP R16

            .ENDM
```

Given a null terminated ASCII string. Write a code to convert flip case (lower becomes upper and upper becomes lower). Write the converted array into DM.

```
            LDI ZH, HIGH(2* alphanum)
            LDI ZL, LOW(2* alphanum)

            LDI XH, HIGH(2* FLIPPED)
            LDI XL, LOW(2* FLIPPED)

            LDI R18,0x20      ;TO HELP FLIP

L1:         LPM R17, Z+
            TST R17           ;NULL ?
            BREQ Done

Chkup:      CPI R17, "A"
            BRLT L1
            CPI R17, "Z"
            BREQ flip2Lo
            BRGE Chklow

flip2Lo:    ADD R17, R18
            ST X+, R17
            RJMP L1

Chklow:     CPI R17, "a"
            BRLT L1
            CPI R17, "z"
            BREQ flip2up
            BRGE L1

flip2up:    SUB R17, R18
            ST X+, R17
            RJMP L1
Done:

alphanum:   .DB       "09ABCDefgh1234", 0

            .DSEG
FLIPPED:    .BYTE     15
```

Given a null terminated ASCII string with only alphabetic characters. Write a code to convert flip case (lower becomes upper and upper becomes lower). Write the converted array into DM.

```
        LDI ZH, HIGH(2* alphanum)
        LDI ZL, LOW(2* alphanum)

        LDI XH, HIGH(2* FLIPPED)
        LDI XL, LOW(2* FLIPPED)

        LDI R18,0x20      ;TO HELP FLIP

L1:     LPM R17, Z+
        TST R17           ;NULL ?
        BREQ Done

        EOR R17, R18
        ST X+, R17
        RJMP L1
Done:

alphanum: .DB     "ABCDefgh", 0

        .DSEG
FLIPPED:  .BYTE   15
```

Given an integer array of length N. Chksum = sum of all the binary 1's in all the elements. Write the code to determine Chksum. E.g. array = {4,5,6,7,8} = {00000100, 00000101, 00000110, 00000111, 00001000} => chksum = 9

```
                LDI ZH, HIGH(2*ARRAY)
                LDI ZL, LOW(2*ARRAY)

                LDI R16, N
                CLR R18
                CLR R19             ;CHKSUM

L1:             LPM R17, Z+

SHIFT:          LSR R17             ;C = LSB
                ADC R19, R18        ;ADDS THE CARRY FALG
                TST R17
                BRNE SHIFT

                DEC R16
                TST R16
                BRNE L1
```

1. Given an array of length N. Create in the DM a mirrored copy of the array with a length 2N. Eg array = {1,2,3,4} => mirror copy = {1,2,3,4,4,3,2,1}

```
1   .DSEG
2   Mirror: .BYTE 8
3
4           .CSEG
5           LDI ZH, HIGH(2*Array)
6           LDI ZL, LOW(2*Array)
7           LDI XH, HIGH(Mirror)
8           LDI XL, LOW(Mirror)
9
10          LDI R16, Len
11  L1:     LPM R17, Z+
12          ST X+, R17
13          DEC R16
14          TST R16
15          BRNE L1
16
17          LDI R16, Len
18          LDI ZH, HIGH(2*Array+len-1)
19          LDI ZL, LOW(2*Array+len-1)
20
21          LDI XH, HIGH(Mirror+len)
22          LDI XL, LOW(Mirror+len)
23
24
25  L2:     LPM R17, Z
26          ST X+, R17
27          SBIW R31:R30, 1
28          DEC R16
29          TST R16
30          BRNE L2
31
32  END:    RJMP END
33
34  Array: .DB 1,2,3,4
35  .EQU    Len = 4
```

Given an integer array of length N. Create in the DM a cumulative sum array of length N. E.g. array = {1,2,3,4} => cumsum = {1,3,6,10}

```
.MACRO      CSUM            ;@0 = PMarray, @1=DMarray, @2=N

            LDI ZH, HIGH(2*PMarray)
            LDI ZL, LOW(2* PMarray)

            LDI XH, HIGH(2* DMarray)        ;X->DMarray[0]
            LDI XL, LOW(2* DMarray)

            LDI R16, @2
            CLR R18                         ;CSUM
L1:         LPM R17, Z+

            ADD R18, R17
            ST X+, R18

            DEC R16
            TST R16
            BRNE L1
```

Given an signed integer byte, N, stored in PM. Determine the binary representation of the abs(N) and store it as an array of bits. E.g. N = -127 => Array = {0,1,1,1,1,1,1,1}

```
                LDI ZH, HIGH(2*N)
                LDI ZL, LOW(2* N)

                LDI XH, HIGH(2* DMarray)        ;X->DMarray[0]
                LDI XL, LOW(2* DMarray)

                LDI R16, 8
                LDI R18, 1
                LDI R19, 0
                LPM R17, Z+

                TST R17
                BRGE POS
                NEG R17

POS:            LSL R17

                BRCS WRITE1
                ST X+, R19
                RJMP NEXT
WRITE1:         ST X+, R18

NEXT:           DEC R16
                TST R16
                BRNE L1
```