ECE 375 LAB 8

Treasure Hunt

Lab Time: Wednesday 12-2

Bradley Martin

INTRODUCTION

The purpose of this lab is to use all use the all the knowledge and tools we have learned in this class to implement a treasure hunt. Taking and expanding routines from past labs and creating new arithmetic functions are required to successfully complete this lab. A skeleton code file is provided that comes with an example to test our program with. The program should find the closes treasure and the avg distance to the treasure.

PROGRAM OVERVIEW

This program takes in 8 bytes of data that have concatenated 10-bit signed numbers embedded. The program first reads those numbers out from program memory and sorts them into 3 cartesian coordinates of unsigned values. All three treasures follow the same method for finding the distance. First the program with find the square of the X coordinate, then the Y coordinate. Both values are stored in 32-bit numbers and added together. This value is saved to memory. The sqrt function is then called to find the distance that value is also stored to memory. The three distances are compared to each other to find the closes one to the origin. Depending on what treasure is closes depends on what value is stored in memory. For example, is distance1 has the smallest distance then $01 will be stored to memory. Next the average distance is found for the three distances. The average is stored to memory. The last routine the program performs is the store all the collected data in the specified orientation and destination provided in the lab handout. There are 3 results of 5 bytes each that follow X^2_plus_Y^2, square root, orientation. Then one byte for best choice, and 2 bytes for the average distance.

Besides the standard INIT and MAIN routines within the program, many additional routines were created and used. The Treasure1, Treasure 2, and Treasure 3 routines are used to gather the coordinates. The sqXn/sqYn routines are used to square a particular coordinate. The ADD32 routine is used to add two 32-bit registers. The sqrt routine finds the square root / distance of a treasure. The SortBestChoice routine finds the closes treasure. The CalAvg routine finds the average distance between treasures. Finally, the Store_Result routine to store all the values in the correct orientation and destination.

INITIALIZATION ROUTINE

The initialization routine Is very short for this program. The program clears the zero flag, and the Stack Pointer is initialized.

MAIN ROUTINE

The Main routine executes the sequence described in the program overview for. First the address of Treasureinfo is initialized to a pointer and the first byte is also pointed to. A small loop is used to get all 8 bytes out of program memory. Then Treasure1, Treasure2, and Treasure 3 are called to get the coordinate points from the Treasureinfo bytes. The next process is then repeated by all three treasures: First the X coordinate is squared, then the Y coordinate. The two numbers are added together, and the result is copied into X1_Y1, X2_Y2, or X3_Y3. The sqrt function is called to find the distance. That number is stored in SqrtCount and copied in distance1, distance2, or distance3. Finally, a call to clrSqrtCount will clear the SqrtCount memory slot so there is no overflow. The SortBestChoice is then called to find the shortest distance. Then CalAvg is called to find the average distance and Store_Result to store the results.

TREASURE1 ROUTINE

The Treasure1 routine first loads the Z Y and X registers with the First byte, second byte and X1 pointers. The low byte of X1 is found first by swapping the bits in the second byte, shifting by two bitsto the right, Performing and with 0b00000011, then shifting the bits in the first byte by two to the left and peroforming OR between first and second byte. That value is stored in the lower byte of X1. The higher byte of X1 is found by re-loading the first and second bytes, swapping the first bytes bits, shifting the bits by 2 to the right, and Performing AND with 0b00000011. That is stored in the higher bytes of X1. Next the value of X1 is temporarily store in Check_Neg_addr and run through Check_neg. This first checks the negative bit to see if it is set, if it is then all the bits past the negative bit are set to 1. A ones commplent is performed on the lower and upper bytes and +1 is added with carry. The new value is then store back into the coordinate.

To find Y1, the second byte and third byte are now loaded. The third byte bits are swapped, logical AND with 0b00001111 is performed. Then the second bytes bits are swaped and the logical AND with 0b11110000 is performed. The second and third byte are then logically ORed together and store in low byte of Y1. For the High byte, the second and third bytes are loaded again. Second bytes bits are swapped and logical AND with 0b00000011 is performed. This is stored in the high byte of Y1. A negative check is again performed.

TREASURE2 ROUTINE

The Treasure2 routine is similar to Treasure1 but for X2 and Y2. For the low byte of X2 the third byte and fourth byte are loaded. The fourth byte is shifted to the right by two bits. Then the third byte bits are swapped and shifted to the left by two. The third byte is then logically ANDed with 0b11000000. The third byte and fourth byte are then ORed together and store in the low byte of X2. For the high byte, the third and fourth bytes are loaded again. The Third byte is shifted to the right 2 bits and then logicalled ANDed with 0b00000011 and stored in the high byte of X2. A negative check is performed.

To find Y2, the fourth and fifth byte are loaded. The fifth byte is immediately load into the low byte of Y2. To find the high byte we load the fourth and fifth byte the fourth byte is logically ANDed with 0b00000011 and set to high byte of Y2. A negative check is performed.

TREASURE3 ROUTINE

The Treasure3 routine is similar to Treasure1 and Treasure2 but for X3 and Y3. For the low byte of X3 with load the sixth byte and the seventh byte. The seventh bytes bits are first swapped, then shifted to the right by two bits and logically ANDed with 0b00000011. The sixth byte is then shifted two bits to the left. The sixth and Seventh byte are then logically ORed together and stored in the low byte of X3. For the high byte, the sixth and seventh bytes are loaded again. The sixth bytes bits are swapped, then shifted to the right by 2 bits, and logically ANDed with 0b00000011. This is stored in the high byte of X3. A negative check is performed.

To find Y3, the seventh and eight bytes are loaded. The eight bytes bits are swapped and so are the seventh bytes bits. Then the seventh byte is logically ANDed with 0b11110000 and logically ORed with the eighth byte. This is stored in the low byte of Y3. For the high byte of Y3 the seventh and eighth bytes are loaded again. The seventh bytes bits are swapped and logically ANDed with 0b00000011. This is stored in the high byte of Y3. A negative check is performed.

SQXN/SQYN ROUTINE

The Square routine will square the coordinate that is passed into it. First the X, Y, and Z registers are loaded with pointers to the coordinate, the first operand and the second operand. Both operands are then loaded with the value of the coordinate. The mul16 function is called to square the number. This number is then added to one of the operands of the add32 to find the squared distance. The LaddrP memory slot must be clear at the end.

ADD32 ROUTINE

The ADD32 routine adds two 32-bit registers to find the squared distance. The operands for this function should already be loaded with the treasure coordinates from the square function. The addition is done in 4 steps, first a regular add between the two lowest bytes, then 3 more additions using adc point towards the next three bytes. Clear the carry at the end.

SQRT ROUTINE

The Sqrt routine will take the square root of the squared distance. It does this by looping an incCounter function that increments a 16-bit counter by 1. The counter is then put through the MUL16 function to get a squared value. The value is then check using one cp for the lowest bytes and 3 more cpc for the next 3 bytes. If the value is the same as our original squared value then we have found the sqrt, otherwise we can still go another step. Make sure to clear LaddrP memory after each step.

SORTBESTCHOICE ROUTINE

The SortBestChoice Routine finds the shortest distance from the origin. The three distances are first loaded in. then the first distance is tested with the second, if they are equal then it will test with the third and if that is equal it will placed -1 in BestChoice. If distance1 is smaller than distance2, if is it will see if distance 1 is smaller than distance, if so, then then 1 is placed in best choice. But if distance2 is smaller then it will test with distance3, if distance 2 is still smaller than 2 is placed in BestChoice. Otherwise a 3 is placed in BestChoice

CALAVG ROUTINE

The CAlAvg Routine first adds up the first two distances. The third distance is then padded with zeros to be added with the other 2. A similar routine is used to find the division of 3 as in the square root. We use the Inccounter function and increment the counter every time we subtract 3 until we are either at 0 or below 0.

STORE_RESULT ROUTINE

The Store_Result routine simply takes the store values we have been collecting and places them is reults1/2/3 with the squared distance first and then distance,

CONCLUSION

In this Lab we implemented a treasure hunt to find the distances to 3 treasures, which one was the closes and what the average distance between them are. This tested my knowledge of AVR and took a lot longer than I thought but was very rewarding to complete.

SOURCE CODE

```
;***********************************************************
;*
;*       Author: Bradley Martin
;*         Date: 11/29/2020
;*
;***********************************************************
.include "m128def.inc"                  ; Include definition file
;***********************************************************
;*      Internal Register Definitions and Constants
;*      (feel free to edit these or add others)
;***********************************************************
.def    rlo = r0                        ; Low byte of MUL result
.def    rhi = r1                        ; High byte of MUL result
.def    zero = r2                               ; Zero register, set to zero in INIT, useful for
calculations
.def    A = r3                                  ; A variable
.def    B = r4                                  ; Another variable
.def    mpr = r16                               ; Multipurpose register
.def    oloop = r17                             ; Outer Loop Counter
.def    iloop = r18                             ; Inner Loop Counter
.def    temp1 = r19                             ; temporary register
.def    temp2 = r20                             ; temporary register
.def    temp3 = r21                             ; temporary register
.def    temp4 = r22                             ; temporary register
.def    temp5 = r23                             ; temporary register
.def    temp6 = r24                             ; temporary register
.def    lpcounter = r25                 ; loop counter

.equ    numBytes = 8                    ; number of bytes to parse initial array

;***********************************************************
;*      Data segment variables
;*      (feel free to edit these or add others)
;***********************************************************
.dseg
.org    $0100                                   ; data memory allocation for operands
FirstB:                 .byte 1                 ; allocate 1 byte
.org    $0101
SecondB:        .byte 1                         ; allocate 1 byte
.org    $0102
ThirdB:                 .byte 1                 ; allocate 1 byte
.org    $0103
FourthB:        .byte 1                         ; allocate 1 byte
.org    $0104
FifthB:                 .byte 1                 ; allocate 1 byte
.org    $0105
SixthB:                 .byte 1                 ; allocate 1 byte
.org    $0106
SeventhB:               .byte 1                 ; allocate 1 byte
.org    $0107
EighthB:        .byte 1                         ; allocate 1 byte

.org    $0108
X1:                             .byte 2                 ; allocate 2 byte
.org    $0110
Y1:                             .byte 2                 ; allocate 2 byte

.org    $0112
X2:                             .byte 2                 ; allocate 2 byte
.org    $0114
Y2:                             .byte 2                 ; allocate 2 byte

.org    $0116
X3:                             .byte 2                 ; allocate 2 byte
.org    $0118
Y3:                             .byte 2                 ; allocate 2 byte

.org    $0120
addrA:                  .byte 2                 ; allocate 2 byte
```

```
.org    $0122
addrB:                  .byte 2                         ; allocate 2 byte
.org    $0124
LaddrP:                 .byte 4                         ; allocate 4 byte

.org    $0128
ADD32_op1:              .byte 4                         ; allocate 4 byte
.org    $0132
ADD32_op2:              .byte 4                         ; allocate 4 byte
.org    $0136
ADD32_Res:              .byte 5                         ; allocate 5 byte

.org    $0141
SqrtCount:              .byte 2                         ; allocate 2 byte
.org    $0143
IncResult:              .byte 2                         ; allocate 2 byte
.org    $0145
CheckCount_Res: .byte 4                         ; allocate 4 byte
.org    $0150
distance1:              .byte 2                         ; allocate 2 byte
distance2:              .byte 2                         ; allocate 2 byte
distance3:              .byte 2                         ; allocate 2 byte

.org    $0156
ADD_D1D2:               .byte 3                         ; allocate 3 byte
Padded_D3:              .byte 3                         ; allocate 3 byte
ADD24_Res:              .byte 4                         ; allocate 4 byte
const_3:        .byte 4                         ; allocate 4 byte
SUB32_Res:              .byte 4                         ; allocate 4 byte
const_0:        .byte 4                         ; allocate 4 byte

.org    $0180
Check_Neg_addr: .byte 2                         ; allocate 2 byte
const_1:        .byte 2                         ; allocate 2 byte
ADD16_result:   .byte 2                         ; allocate 2 byte

.org    $0190
X1_Y1:                  .byte 3                         ; allocate 3 byte
X2_Y2:                  .byte 3                         ; allocate 3 byte
X3_Y3:                  .byte 3                         ; allocate 3 byte

;***********************************************************
;*      Start of Code Segment
;***********************************************************
.cseg                                                   ; Beginning of code segment
;-----------------------------------------------------------
; Interrupt Vectors
;-----------------------------------------------------------
.org    $0000                                   ; Beginning of IVs
                rjmp    INIT                    ; Reset interrupt
.org    $0046                                   ; End of Interrupt Vectors
;-----------------------------------------------------------
; Program Initialization
;-----------------------------------------------------------
INIT:   ; The initialization routine
                clr             zero

                ldi             mpr, low(RAMEND)        ;Initilize stack pointer
                out             SPL, mpr
                ldi             mpr, high(RAMEND)
                out             SPH, mpr
; To do
; your code goes here

;               jmp     Grading

;***********************************************************
;*      Procedures and Subroutines
;***********************************************************
```

```
; your code can go here as well
MAIN:
                ldi             ZL, Low(TreasureInfo<<1)        ; initilize pointer to start of array
                ldi             ZH, High(TreasureInfo<<1)

                ldi             YL, Low(FirstB)                                 ; initilize pointer to
first byte
                ldi             YH, High(FirstB)

                ldi             lpcounter, numBytes                             ; set loop counter to
number of bytes

arrLp:
                lpm             mpr, Z+                                                 ; load a byte
from program memory
                st              Y+, mpr                                                 ; store it in
the correct byte
                dec             lpcounter                                               ; dec loop
counter
                brne    arrLp                                                   ; continue until all
byttes loaded

                rcall   Treasure1                                               ; get the cordinates for
first treasure
                rcall   Treasure2                                               ; get the cordinates for
second treasure
                rcall   Treasure3                                               ; get the cordinates for
third treasure

                rcall   sqX1                                                    ; square X1
                rcall   sqY1                                                    ; square Y1
                rcall   ADD32                                                   ; add Square X1 and
square Y1

                ldi             XL, low(ADD32_Res)                              ; set a pointer to
X1^2+Y1^2
                ldi             XH, high(ADD32_Res)

                ldi             YL, low(X1_Y1)                                  ; set pointer to store
X1^2+Y1^2
                ldi             YH, high(X1_Y1)

                ld              temp1, X+                                               ; store the
first byte
                st              Y+, temp1
                ld              temp1, X+                                               ; store second
byte
                st              Y+, temp1
                ld              temp1, X                                        ; store third byte
                st              Y, temp1

                rcall   sqrt                                                    ; find the square root
of treasure 1

                ldi             XL, low(SqrtCount)                              ; pointer to treasure 1
sqrt
                ldi             XH, high(SqrtCount)

                ldi             YL, low(distance1)                              ; pointer to place to
store treasure 1 distance
                ldi             YH,     high(distance1)

                ld              temp1, X+                                               ; store first
byte
                st              Y+, temp1
                ld              temp1, X                                        ; store second byte
                st              Y, temp1
                rcall   clrSqrtCount                                    ; clear counter
```

```
                rcall   sqX2                                    ; Find X2^2
                rcall   sqY2                                    ; Find Y2^2
                rcall   ADD32                                   ; add X2^2 and Y2^2

                ldi             XL, low(ADD32_Res)              ;set a pointer to
X2^2+Y2^2
                ldi             XH, high(ADD32_Res)

                ldi             YL, low(X2_Y2)                  ; set pointer to store
X2^2+Y2^2
                ldi             YH, high(X2_Y2)

                ld              temp1, X+                               ; store first
byte
                st              Y+, temp1
                ld              temp1, X+                               ; store second
byte
                st              Y+, temp1
                ld              temp1, X                        ; store third byte
                st              Y, temp1

                rcall   sqrt                                    ; find treasure 2 sqrt

                ldi             XL, low(SqrtCount)              ; pointer to treasure 2
sqrt
                ldi             XH, high(SqrtCount)

                ldi             YL, low(distance2)              ; pointer to place to
store treasure 2 distance
                ldi             YH,     high(distance2)

                ld              temp1, X+                               ; store first
byte
                st              Y+, temp1
                ld              temp1, X                        ; store second byte
                st              Y, temp1
                rcall   clrSqrtCount                    ; clear counter

                rcall   sqX3                                    ; find X3^2
                rcall   sqY3                                    ; find Y3^2
                rcall   ADD32                                   ; add X3^2 and Y3^2

                ldi             XL, low(ADD32_Res)              ;set a pointer to
X3^2+Y3^2
                ldi             XH, high(ADD32_Res)

                ldi             YL, low(X3_Y3)                  ; set pointer to store
X3^2+Y3^2
                ldi             YH, high(X3_Y3)

                ld              temp1, X+                               ; store first
byte
                st              Y+, temp1
                ld              temp1, X+                               ; store second
byte
                st              Y+, temp1
                ld              temp1, X                        ; store third byte
                st              Y, temp1

                rcall   sqrt                                    ; find treasure 3 sqrt

                ldi             XL, low(SqrtCount)              ; pointer to treasure 3
sqrt
                ldi             XH, high(SqrtCount)

                ldi             YL, low(distance3)              ; pointer to place to
store treasure 3 distance
                ldi             YH,     high(distance3)
```

```
                ld              temp1, X+                                       ; store first
byte
                st              Y+, temp1
                ld              temp1, X                                        ; store second byte
                st              Y, temp1
                rcall   clrSqrtCount                                    ; clear counter

                rcall   SortBestChoice                                  ; find the nearest treasure

                rcall   CalAvg                                                  ; calculate the average
distance

                rcall   Store_Result                                    ; store the results for grading


                rjmp    MAIN                                                    ; infinite loop

;-----------------------------------------------------------
; Func: Treasure1
; Desc: Parese out the Coordinates for treasure 1.
;-----------------------------------------------------------
Treasure1:
;X1
                ldi             ZL, low(FirstB)                 ; pointer to first byte
                ldi             ZH, high(FirstB)

                ldi             YL, low(SecondB)        ; pointer to second byte
                ldi             YH, high(SecondB)

                ldi             XL, low(X1)                             ; pointer to X1
                ldi             XH, high(X1)

                ld              temp1, Z                        ; load temp1 with Z
                ld              temp2, Y                        ; load temp2 with Y

                swap    temp2                                   ; swap bits of temp2
                lsr             temp2                                   ; shift bits right twice
                lsr             temp2
                ldi             mpr, 0b00000011                 ; perform logical AND
                and             temp2, mpr
                lsl             temp1                                   ; shift bits left twice
                lsl             temp1
                or              temp2, temp1                    ; perform logical OR

                st              X+, temp2                               ; store low byte

                ld              temp1, Z                        ; load temp1 with next byte
                ld              temp2, Y                        ; load temp2 with next byte

                swap    temp1                                   ; swap temp1 bits
                lsr             temp1                                   ; shift bits right twice
                lsr             temp1
                ldi             mpr, 0b00000011                 ; perform logical AND
                and             temp1, mpr

                st              X+, temp1                               ;store high byte

                ldi             XL, low(X1)                             ; pointer to X1
                ldi             XH, high(X1)

                ldi             YL, low(Check_Neg_addr)  ; Pointer to Check neg address
                ldi             YH, high(Check_Neg_addr)

                ld              temp1, X+                               ; copy the contents of X1 to
Check_neg_addr
                st              Y+, temp1
                ld              temp1, X
                st              Y, temp1
```

```
                rcall Check_Neg                                 ; Check if signed negative number

                ldi             XL, low(Check_Neg_addr)  ; pointer to check_neg_addr
                ldi             XH, high(Check_Neg_addr)

                ldi             YL, low(X1)                             ; pointer to X1
                ldi             YH, high(X1)

                ld              temp1, X+                               ; Copy contents of
check_neg_addr to X1
                st              Y+, temp1
                ld              temp1, X
                st              Y, temp1

;Y1
                ldi             ZL, low(SecondB)        ; pointer to second byte
                ldi             ZH, high(SecondB)

                ldi             YL, low(ThirdB)                 ; pointer to thrid byte
                ldi             YH, high(ThirdB)

                ldi             XL, low(Y1)                             ; pointer to Y1
                ldi             XH, high(Y1)

                ld              temp1, Z                        ; load temp registers with lower bytes
                ld              temp2, Y

                swap    temp2                                   ; swap temp2 bits
                ldi             mpr, 0b00001111                 ; Perform logical AND
                and             temp2, mpr
                swap    temp1                                   ; swap temp1 bits
                ldi             mpr, 0b11110000                 ; Perform logical AND
                and             temp1, mpr
                or              temp1, temp2                    ; Perform logical OR with both registers

                st              X+, temp1                               ; store in low byte

                ld              temp1, Z                        ; load temp registers with next byte
                ld              temp2, Y

                swap    temp1                                   ; swap temp1 bits
                ldi             mpr, 0b00000011                 ; Perform logical AND
                and             temp1, mpr

                st              X+, temp1                               ; store in high byte

                ldi             XL, low(Y1)                             ; pointer to Y1
                ldi             XH, high(Y1)

                ldi             YL, low(Check_Neg_addr)  ; pointer to Check_neg_addr
                ldi             YH, high(Check_Neg_addr)

                ld              temp1, X+                               ; copy contents of y1 to
check_neg_addr
                st              Y+, temp1
                ld              temp1, X
                st              Y, temp1

                rcall Check_Neg                                 ; check if signed negative value

                ldi             XL, low(Check_Neg_addr)  ; pointer to check_neg_addr
                ldi             XH, high(Check_Neg_addr)

                ldi             YL, low(Y1)                             ; pointer to Y1
                ldi             YH, high(Y1)

                ld              temp1, X+                               ; copy contents of
check_neg_addr to Y1
                st              Y+, temp1
```

```
                ld              temp1, X
                st              Y, temp1

                ret                                             ; End a function with RET

;-----------------------------------------------------------
; Func: Treasure2
; Desc: Parse out the coordinates for treasure 2
;-----------------------------------------------------------
Treasure2:
;X2
                ldi             ZL, low(ThirdB)                 ; pointer to third byte
                ldi             ZH, high(ThirdB)

                ldi             YL, low(FourthB)        ; pointer to fourth byte
                ldi             YH, high(FourthB)

                ldi             XL, low(X2)                             ; pointer to X2
                                ldi             XH, high(X2)
                Ld              temp1, Z                        ; load third byte
                ld              temp2, Y                        ; load fourth byte

                lsr             temp2                                   ; shift bit to the right twice
                lsr             temp2
                swap    temp1                                   ; swap fourth byte
                lsl             temp1                                   ; shift to the left twice
                lsl             temp1
                ldi             mpr, 0b11000000
                and             temp1,mpr                               ; Perform logical AND
                or              temp1, temp2                    ; Perform Locical OR

                st              X+, temp1                               ; store in low byte

                ld              temp1, Z                        ; Load third byte
                ld              temp2, Y                        ; load fourth byte

                lsr             temp1                                   ; shift two bits to the right
                lsr             temp1
                ldi             mpr, 0b00000011
                and             temp1, mpr                              ; Perform logical AND

                st              X+, temp1                               ; store in high byte

                ldi             XL, low(X2)                             ; pointer to X2
                ldi             XH, high(X2)

                ldi             YL, low(Check_Neg_addr)  ; pointer check_neg_addr
                ldi             YH, high(Check_Neg_addr)

                ld              temp1, X+                               ; load X2
                st              Y+, temp1                               ; store into Check_neg_addr
                ld              temp1, X
                st              Y, temp1

                rcall Check_Neg                                 ; check if bytes are negative

                ldi             XL, low(Check_Neg_addr)  ; pointer to check_neg_addr
                ldi             XH, high(Check_Neg_addr)

                ldi             YL, low(X2)                             ; pointer to X2
                ldi             YH, high(X2)

                ld              temp1, X+                               ; copy Check_neg _addr into X2
                st              Y+, temp1
                ld              temp1, X
                st              Y, temp1


;Y2
```

```
                ldi             ZL, low(FourthB)        ; pointer to fourth byte
                ldi             ZH, high(FourthB)

                ldi             YL, low(FifthB)                 ; pointer to fifth byte
                ldi             YH, high(FifthB)

                ldi             XL, low(Y2)                             ; pointer to Y2
                ldi             XH, high(Y2)


                ld              temp1, Z                        ; load third byte
                ld              temp2, Y                        ; load fourth byte

                st              X+, temp2                               ; store low byte

                ld              temp1, Z                        ; load third byte
                ld              temp2, Y                        ; load fourth byte

                ldi             mpr, 0b00000011
                and             temp1, mpr                              ; Perform logical AND

                st              X+, temp1                               ; store high byte

                ldi             XL, low(Y2)                             ; pointer to Y2
                ldi             XH, high(Y2)

                ldi             YL, low(Check_Neg_addr)  ; pointer to Check_neg_addr
                ldi             YH, high(Check_Neg_addr)

                ld              temp1, X+                               ; copy Y2 into Check_neg_addr
                st              Y+, temp1
                ld              temp1, X
                st              Y, temp1

                rcall Check_Neg                                 ; check if bytes are negative

                ldi             XL, low(Check_Neg_addr)  ; pointer to check_neg_addr
                ldi             XH, high(Check_Neg_addr)

                ldi             YL, low(Y2)                             ; pointer to Y2
                ldi             YH, high(Y2)

                ld              temp1, X+                               ; copy Check_neg_addr into Y2
                st              Y+, temp1
                ld              temp1, X
                st              Y, temp1


                ret                                             ; End a function with RET

;-----------------------------------------------------------
; Func: Treasure3
; Desc: Parse out the coordinates for treasure 3
;-----------------------------------------------------------
Treasure3:
;X3
                ldi             ZL, low(SixthB)                         ; pointer to sixth byte
                ldi             ZH, high(SixthB)

                ldi             YL, low(SeventhB)                       ; pointer to seventh byte
                ldi             YH, high(SeventhB)

                ldi             XL, low(X3)                                     ; pointer to X3
                ldi             XH, high(X3)

                ld              temp1, Z                                ; load sixth byte
                ld              temp2, Y                                ; load seventh byte

                swap    temp2                                           ; swap seventh byte
```

```
                lsr             temp2                                   ; shift to the right
twice
                lsr             temp2
                ldi             mpr, 0b00000011
                and             temp2, mpr                              ; Perform Logical AND
                lsl             temp1                                   ; shift sixth twice to
the left
                lsl             temp1
                or              temp1, temp2                    ; Perform logical OR

                st              X+, temp1                               ; store in low byte


                ld              temp1, Z                        ; load sixth byte
                ld              temp2, Y                        ; load seventh byte

                swap    temp1                                   ; swap sixth byte
                lsr             temp1                                   ; shift to the right
twice
                lsr             temp1
                ldi             mpr, 0b00000011
                AND             temp1, mpr                              ; Perform logical AND

                st              X+, temp1                               ; store in high byte

                ldi             XL, low(X3)                             ; pointer to X3
                ldi             XH, high(X3)

                ldi             YL, low(Check_Neg_addr)         ; Pointer to Check_neg_addr
                ldi             YH, high(Check_Neg_addr)

                ld              temp1, X+                               ;copy X3 into
check_neg_addr
                st              Y+, temp1
                ld              temp1, X
                st              Y, temp1

                rcall Check_Neg                                 ; check if bytes are negative

                ldi             XL, low(Check_Neg_addr)         ; pointer to check_neg_addr
                ldi             XH, high(Check_Neg_addr)

                ldi             YL, low(X3)                             ; pointer to X3
                ldi             YH, high(X3)

                ld              temp1, X+                               ; copy check_neg_addr to
X3
                st              Y+, temp1
                ld              temp1, X
                st              Y, temp1

;Y3
                ldi             ZL, low(SeventhB)                       ; pointer to seventh byte
                ldi             ZH, high(SeventhB)

                ldi             YL, low(EighthB)                ; pointer to eighth byte
                ldi             YH, high(EighthB)

                ldi             XL, low(Y3)                             ; pointer to Y3
                ldi             XH, high(Y3)

                ld              temp1, Z                        ; load seventh byte
                ld              temp2, Y                        ; load eighth byte

                swap    temp2                                   ; swap eighth
                swap    temp1                                   ; swap seventh
                ldi             mpr,0b11110000
                and             temp1, mpr                              ; Perform Logical AND
                or              temp2, temp1                    ; Perform Logical OR
```

```
                st              X+, temp2                                       ; store in low byte

                ld              temp1, Z                                ; load seventh byte
                ld              temp2, Y                                ; load eighth byte

                swap    temp1                                           ; swap seventh
                ldi             mpr, 0b00000011
                and             temp1, mpr                                      ; Perform logical AND

                st              X+, temp1                                       ; store in high byte

                ldi             XL, low(Y3)                                     ; pointer to Y3
                ldi             XH, high(Y3)

                ldi             YL, low(Check_Neg_addr)         ; Pointer to Check_neg_addr
                ldi             YH, high(Check_Neg_addr)

                ld              temp1, X+                                       ; copy Y3 into
check_neg_addr
                st              Y+, temp1
                ld              temp1, X
                st              Y, temp1

                rcall Check_Neg                                         ; check if bytes are negative

                ldi             XL, low(Check_Neg_addr)         ; pointer to check_neg_addr
                ldi             XH, high(Check_Neg_addr)

                ldi             YL, low(Y3)                                     ; pointer to Y3
                ldi             YH, high(Y3)

                ld              temp1, X+                                       ; copy Check_neg_addr
into Y3
                st              Y+, temp1
                ld              temp1, X
                st              Y, temp1

                ret                                             ; End a function with RET
;-----------------------------------------------------------
; Func: MUL16
; Desc: An example function that multiplies two 16-bit numbers
;                       A - Operand A is gathered from address $0101:$0100
;                       B - Operand B is gathered from address $0103:$0102
;                       Res - Result is stored in address
;                                       $0107:$0106:$0105:$0104
;               You will need to make sure that Res is cleared before
;               calling this function.
;-----------------------------------------------------------
MUL16:

                clr             zero                    ; Maintain zero semantics

                ; Set Y to beginning address of B
                ldi             YL, low(addrB)   ; Load low byte
                ldi             YH, high(addrB)  ; Load high byte

                ; Set Z to begginning address of resulting Product
                ldi             ZL, low(LAddrP)  ; Load low byte
                ldi             ZH, high(LAddrP); Load high byte

                ; Begin outer for loop
                ldi             oloop, 2        ; Load counter
MUL16_OLOOP:
                ; Set X to beginning address of A
                ldi             XL, low(addrA)   ; Load low byte
                ldi             XH, high(addrA)  ; Load high byte

                ; Begin inner for loop
```

```
                ldi             iloop, 2        ; Load counter
MUL16_ILOOP:
                ld              A, X+                   ; Get byte of A operand
                ld              B, Y                    ; Get byte of B operand
                mul             A,B                             ; Multiply A and B
                ld              A, Z+                   ; Get a result byte from memory
                ld              B, Z+                   ; Get the next result byte from memory
                add             rlo, A                  ; rlo <= rlo + A
                adc             rhi, B                  ; rhi <= rhi + B + carry
                ld              A, Z                    ; Get a third byte from the result
                adc             A, zero                 ; Add carry to A
                st              Z, A                    ; Store third byte to memory
                st              -Z, rhi                 ; Store second byte to memory
                st              -Z, rlo                 ; Store first byte to memory
                adiw    ZH:ZL, 1        ; Z <= Z + 1
                dec             iloop                   ; Decrement counter
                brne    MUL16_ILOOP             ; Loop if iLoop != 0
                ; End inner for loop

                sbiw    ZH:ZL, 1        ; Z <= Z - 1
                adiw    YH:YL, 1        ; Y <= Y + 1
                dec             oloop                   ; Decrement counter
                brne    MUL16_OLOOP             ; Loop if oLoop != 0
                ; End outer for loop

                ret                                             ; End a function with RET

;-----------------------------------------------------------
; Func: Template function header
; Desc: Cut and paste this and fill in the info at the
;               beginning of your functions
;-----------------------------------------------------------
sqX1:
                ldi             XL, low(X1)                                     ; pointer to X1
                ldi             XH, high(X1)

                ldi             YL, low(addrA)                  ; pointer to first operand
                ldi             YH, high(addrA)

                ldi             ZL, low(addrB)                  ; pointer to second operand
                ldi             ZH, high(addrB)

                ldi             lpcounter, 2                    ; set loop counter to 2
sqX1lp:

                ld              temp1, X+                               ; load the contents of
X1
                st              Y+,     temp1                           ; and copy into Y and Z
                st              Z+, temp1
                dec             lpcounter
                brne    sqX1lp

                rcall   MUL16                                   ; Calculate square

                ldi             XL, low(LaddrP)                 ; Pointer to LaddrP
                ldi             XH, high(LaddrP)

                ldi             YL, low(ADD32_op1)              ; pointer to add32 operand 1
                ldi             YH, high(ADD32_op1)

                ldi             lpcounter, 4                    ; set loop counter to 4

sqX1_ADD32lp:
                ld              temp1, X+                               ; copy contents of
square to
                st              Y+, temp1                               ; the first operand in
ADD32
                dec             lpcounter
                brne    sqX1_Add32lp
```

```
                ldi             YL, low(LaddrP)                         ; pointer to LaddrP
                ldi             YH, high(LaddrP)

                ldi             lpcounter, 4                            ; set loop counter to 4
                ldi             mpr, $00

X1clrLaddrP:
                st              Y+, mpr                                         ; set LaddrP to 0
                dec             lpcounter
                brne    X1clrLaddrP

                ret                                             ; End a function with RET
;-----------------------------------------------------------
; Func: Template function header
; Desc: Cut and paste this and fill in the info at the
;               beginning of your functions
;-----------------------------------------------------------
sqY1:
                ldi             XL, low(Y1)                             ; pointer to Y1
                ldi             XH, high(Y1)

                ldi             YL, low(addrA)                          ; pointer to first operand
                ldi             YH, high(addrA)

                ldi             ZL, low(addrB)                          ; pointer to second operand
                ldi             ZH, high(addrB)

                ldi             lpcounter, 2                            ; set loop counter to 2
sqY1lp:

                ld              temp1, X+                                       ; load contents of X1
                st              Y+,     temp1                                   ; and copy into Y and Z
                st              Z+, temp1
                dec             lpcounter
                brne    sqY1lp

                rcall   MUL16                                           ; calculate square

                ldi             XL, low(LaddrP)                         ; pointer to LaddrP
                ldi             XH, high(LaddrP)

                ldi             YL, low(ADD32_op2)                      ; pointer to add32 operand 2
                ldi             YH, high(ADD32_op2)

                ldi             lpcounter, 4                            ; set loop counter to 4

sqY1_ADD32lp:
                ld              temp1, X+                                       ; copy contents of
square to
                st              Y+, temp1                                       ; the second operand in
add32
                dec             lpcounter
                brne    sqY1_Add32lp

                ldi             YL, low(LaddrP)                         ; pointer to LaddrP
                ldi             YH, high(LaddrP)

                ldi             lpcounter, 4                            ; set loop counter to 4
                ldi             mpr, $00

Y1clrLaddrP:
                st              Y+, mpr                                         ; set LaddrP to 0
                dec             lpcounter
                brne    Y1clrLaddrP

                ret                                             ; End a function with RET

;-----------------------------------------------------------
```

```
; Func: Template function header
; Desc: Cut and paste this and fill in the info at the
;               beginning of your functions
;-----------------------------------------------------------
sqX2:
                ldi             XL, low(X2)                                     ; pointer to X2
                ldi             XH, high(X2)

                ldi             YL, low(addrA)                          ; pointer to first operand
                ldi             YH, high(addrA)

                ldi             ZL, low(addrB)                          ; pointer to second operand
                ldi             ZH, high(addrB)

                ldi             lpcounter, 2                            ; set loop counter to 2
sqX2lp:

                ld              temp1, X+                                       ; load contents of X2
                st              Y+,     temp1                                   ; and copy into Y and Z
                st              Z+, temp1
                dec             lpcounter
                brne    sqX2lp

                rcall   MUL16                                           ; calculate Square

                ldi             XL, low(LaddrP)                         ; pointer to LaddrP
                ldi             XH, high(LaddrP)

                ldi             YL, low(ADD32_op1)                      ; pointer to add32 operand 1
                ldi             YH, high(ADD32_op1)

                ldi             lpcounter, 4                            ; set loop counter to 4

sqX2_ADD32lp:
                ld              temp1, X+                                       ; copy contents of
square to
                st              Y+, temp1                                       ; the first operand in
add32
                dec             lpcounter
                brne    sqX2_Add32lp

                ldi             YL, low(LaddrP)                         ; pointer to LaddrP
                ldi             YH, high(LaddrP)

                ldi             lpcounter, 4                            ; set loop counter to 4
                ldi             mpr, $00

X2clrLaddrP:
                st              Y+, mpr                                         ; set LaddrP to 0
                dec             lpcounter
                brne    X2clrLaddrP

                ret                                             ; End a function with RET

;-----------------------------------------------------------
; Func: Template function header
; Desc: Cut and paste this and fill in the info at the
;               beginning of your functions
;-----------------------------------------------------------
sqY2:
                ldi             XL, low(Y2)                                     ; pointer to Y2
                ldi             XH, high(Y2)

                ldi             YL, low(addrA)                          ; pointer to first operand
                ldi             YH, high(addrA)

                ldi             ZL, low(addrB)                          ; pointer to second operand
                ldi             ZH, high(addrB)
```

```
                ldi             lpcounter, 2                            ; set loop counter to 2
sqY2lp:

                ld              temp1, X+                                       ; load contents of Y2
                st              Y+,     temp1                                   ; and copy into Y and Z
                st              Z+, temp1
                dec             lpcounter
                brne    sqY2lp

                rcall   MUL16                                           ; Calculate square

                ldi             XL, low(LaddrP)                         ; pointer to LaddrP
                ldi             XH, high(LaddrP)

                ldi             YL, low(ADD32_op2)                      ; pointer to add32 operand 2
                ldi             YH, high(ADD32_op2)

                ldi             lpcounter, 4                            ; set loop counter to 4

sqY2_ADD32lp:
                ld              temp1, X+                                       ; copy contents of
square to
                st              Y+, temp1                                       ; the second operand in
add32
                dec             lpcounter
                brne    sqY2_Add32lp

                ldi             YL, low(LaddrP)                         ; pointer to LaddrP
                ldi             YH, high(LaddrP)

                ldi             lpcounter, 4                            ; set loop counter to 4
                ldi             mpr, $00

Y2clrLaddrP:
                st              Y+, mpr                                         ; set LaddrP to 0
                dec             lpcounter
                brne    Y2clrLaddrP

                ret                                             ; End a function with RET

;-----------------------------------------------------------
; Func: Template function header
; Desc: Cut and paste this and fill in the info at the
;               beginning of your functions
;-----------------------------------------------------------
sqX3:
                ldi             XL, low(X3)                                     ; pointer to X3
                ldi             XH, high(X3)

                ldi             YL, low(addrA)                          ; pointer to first operand
                ldi             YH, high(addrA)

                ldi             ZL, low(addrB)                          ; pointer to second operand
                ldi             ZH, high(addrB)

                ldi             lpcounter, 2                            ; set loop counter to 2
sqX3lp:

                ld              temp1, X+                                       ; load contents of X3
                st              Y+,     temp1                                   ; and copy into Y and Z
                st              Z+, temp1
                dec             lpcounter
                brne    sqX3lp

                rcall   MUL16                                           ; calculate square

                ldi             XL, low(LaddrP)                         ; pointer to LaddrP
                ldi             XH, high(LaddrP)
```

```
                ldi             YL, low(ADD32_op1)                      ; pointer to add32 operand 2
                ldi             YH, high(ADD32_op1)

                ldi             lpcounter, 4                            ; set loop counter to 4

sqX3_ADD32lp:
                ld              temp1, X+                                       ; copy contents of
square to
                st              Y+, temp1                                       ; the first operand in
add32
                dec             lpcounter
                brne    sqX3_Add32lp

                ldi             YL, low(LaddrP)                         ; pointer to LaddrP
                ldi             YH, high(LaddrP)

                ldi             lpcounter, 4                            ; set loop counter to 4
                ldi             mpr, $00

X3clrLaddrP:
                st              Y+, mpr                                         ; clear LaddrP
                dec             lpcounter
                brne    X3clrLaddrP
                ret                                             ; End a function with RET

;-----------------------------------------------------------
; Func: Template function header
; Desc: Cut and paste this and fill in the info at the
;               beginning of your functions
;-----------------------------------------------------------
sqY3:
                ldi             XL, low(Y3)                                     ; pointer to Y3
                ldi             XH, high(Y3)

                ldi             YL, low(addrA)                          ; pointer to first operand
                ldi             YH, high(addrA)

                ldi             ZL, low(addrB)                          ; pointer to second operand
                ldi             ZH, high(addrB)

                ldi             lpcounter, 2                            ; set loop counter to 2
sqY3lp:

                ld              temp1, X+                                       ; load contents of Y3
                st              Y+,     temp1                                   ; and copy into Y and Z
                st              Z+, temp1
                dec             lpcounter
                brne    sqY3lp

                rcall   MUL16                                           ; calculate square

                ldi             XL, low(LaddrP)                         ; pointer to LaddrP
                ldi             XH, high(LaddrP)

                ldi             YL, low(ADD32_op2)                      ; pointer to add32 operand 2
                ldi             YH, high(ADD32_op2)

                ldi             lpcounter, 4                            ; set loop counter to 4

sqY3_ADD32lp:
                ld              temp1, X+                                       ; copy contents of
square to
                st              Y+, temp1                                       ; the second operand in
add32
                dec             lpcounter
                brne    sqY3_Add32lp

                ldi             YL, low(LaddrP)                         ; pointer to LaddrP
                ldi             YH, high(LaddrP)
```

```
                ldi             lpcounter, 4                    ; set loop counter to 4
                ldi             mpr, $00
                ldi             lpcounter, 4                    ; set loop counter to 4
                ldi             mpr, $00

Y3clrLaddrP:
                st              Y+, mpr                                 ; clear LaddrP
                dec             lpcounter
                brne    Y3clrLaddrP
                ret                                             ; End a function with RET

;-----------------------------------------------------------
; Func: ADD32
; Desc: Adds two 32-bit registers.
;-----------------------------------------------------------
ADD32:
                clr             zero                                    ; clear zero register

                ldi             XL, low(ADD32_op1)              ; pointer to first operand
                ldi             XH, high(ADD32_op2)

                ldi             YL, low(ADD32_op2)              ; pointer to second operand
                ldi             Yh, high(ADD32_op2)

                ldi             ZL, low(ADD32_Res)              ; pointer to store the result
                ldi             ZH, high(ADD32_Res)

                ld              temp1, X+                               ; load the first bytes
                ld              temp2, Y+
                add             temp1, temp2                    ; add the two registers
                st              Z+, temp1                               ; store in lowest byte
of result

                ld              temp1, X+                               ; load the next bytes
                ld              temp2, Y+
                adc             temp2, temp1                    ; add the two registers with
carry
                st              Z+, temp2                               ; store in result

                ld              temp1, X+                               ; load the next bytes
                ld              temp2, Y+
                adc             temp2, temp1                    ; add the two registers with
carry
                st              Z+, temp2                               ; store in result

                ld              temp1, X                        ; load last bytes
                ld              temp2, Y
                adc             temp2, temp1                    ; add the two registers with
carry
                st              Z+, temp2                               ; store in the result

                brcc    EXIT                                    ; exit if ther is no carry
                st              Z, XH                                   ; store the carry and
clear
                clc
EXIT:
                ret                                             ; End a function with RET

;-----------------------------------------------------------
; Func: IncCounter
; Desc: Increments a 16-bit counter
;-----------------------------------------------------------
IncCounter:
                ldi             XL, low(SqrtCount)              ; pointer towards current value
                ldi             XH, high(SqrtCount)

                ldi             temp3, $01                              ; load $01 into 16-bit
register
```

```
                ldi             temp4, $00

                ldi             ZL, low(IncResult)                      ; pointer to IncResult
                ldi             ZH, high(IncResult)
                clc

                ld              temp1, X+                                       ; add 1 to counter
                add             temp1, temp3
                st              Z+, temp1                                       ; store in result

                ld              temp1, X                                ; add carry if there is one
                adc             temp4, temp1
                st              Z+,     temp4                                   ; store in result

                brcc    EXIT1                                           ; exit if no carry
                st              Z, XH                                           ; store carry and exit
                clc
EXIT1:
                ldi             XL, low(SqrtCount)                      ; pointer to sqrtcount
                ldi             XH, high(SqrtCount)

                ldi             YL, low(IncResult)                      ; pointer to incresult
                ldi             YH, high(IncResult)

                ld              temp1, Y+                                       ; copy new value into
current value
                st              X+, temp1
                ld              temp1, Y
                st              X, temp1

                ldi             XL, low(IncResult)                      ; pointer towards our result
                ldi             YH, high(IncResult)

                ldi             lpcounter, 2                            ; load loop counter with 2
                ldi             mpr, $00

clrIncResult:
                st              X+, mpr                                         ; clear our result
                dec             lpcounter
                brne    clrIncResult

                ret                                             ; End a function with RET

;-----------------------------------------------------------
; Func: clrLaddP
; Desc: Clears the contents stored in LaddrP
;-----------------------------------------------------------
clrLaddrP:
                ldi             YL, low(LaddrP)                         ; pointer to LaddrP
                ldi             YH, high(LaddrP)

                ldi             lpcounter, 4                            ; set loop counter to 4
                ldi             mpr, $00

clrLaddrPlp:
                st              Y+, mpr                                         ; clear the contents of
LaddrP
                dec             lpcounter
                brne    clrLaddrPlp

                ret                                             ; End a function with RET

;-----------------------------------------------------------
; Func: sqrt
; Desc: calculates the square root of the squared number passed in.
;-----------------------------------------------------------
sqrt:
                rcall   IncCounter                              ; increment counter
                rcall   clrLaddrP                               ; clear LaddrP
```

```
                ldi             XL, low(SqrtCount)              ; pointer to SqrtCount
                ldi             XH, high(SqrtCount)

                ldi             YL, low(addrA)                  ; pointer to addrA
                ldi             YH, high(addrA)

                ldi             ZL, low(addrB)                  ; pointer to addrB
                ldi             ZH, high(addrB)

                ldi             lpcounter, 2                    ; set loop counter to 2
sqIncCounterlp:

                ld              temp1, X+                               ; load the current count
                st              Y+,     temp1                           ; copy it into Y and Z
                st              Z+, temp1
                dec             lpcounter
                brne    sqIncCounterlp

                rcall   MUL16                                   ; sq count
                rcall   Checksqrt                               ; check if we have hit base case

                ret                                             ; End a function with RET

;-----------------------------------------------------------
; Func: Checksqrt
; Desc: Compares target number with incremented test value.
;-----------------------------------------------------------
Checksqrt:
                ldi             XL, low(ADD32_Res)              ; pointer to add32_res
                ldi             XH, high(ADD32_Res)

                ldi             YL, low(LaddrP)                 ; pointer to LaddrP
                ldi             YH, high(LaddrP)

                ld              temp1, x+                               ; load the first bytes
                ld              temp2, Y+
                cp              temp1, temp2                    ; compare the two bytes

                ld              temp1, X+                               ; load the next two bytes
                ld              temp2, Y+
                cpc             temp1, temp2                    ; compare them with carry

                ld              temp1, X+                               ; load the next two bytes
                ld              temp2, Y+
                cpc             temp1,temp2                             ; compare them with carry

                ld              temp1, X+                               ; load the last bytes
                ld              temp2, Y+
                cpc             temp1,temp2                             ; compare them with carry
                BREQ    EXIT2                                   ; if the values are equal then branch
                BRSH    sqrt                                    ; if Add32_res is smaller than LaddrP
continue incrementing
                clc                                                             ; clear flag
EXIT2:
                rcall   clrLaddrP                               ; clear LaddrP
                ret                                             ; End a function with RET
;-----------------------------------------------------------
; Func: clrSqrtCount
; Desc: clears the contents of SqrtCount.
;-----------------------------------------------------------
clrSqrtCount:
                ldi             XL, low(SqrtCount)              ; pointer to SqrtCount
                ldi             XH, high(SqrtCount)

                ldi             lpcounter, 2                    ; set loop counter to 2
                ldi             mpr, $00

clrSqrtCountlp:
```

```
                st              x+, mpr                                 ; clear Sqrtcount
                dec             lpcounter
                brne    clrSqrtCountlp
                ret                                             ; End a function with RET

;-----------------------------------------------------------
; Func: Template function header
; Desc: Cut and paste this and fill in the info at the
;               beginning of your functions
;-----------------------------------------------------------
SortBestChoice:


                ldi             XL, low(distance1)              ; pointer to distance1
                ldi             XH, high(distance1)

                ldi             YL, low(distance2)              ; pointer to distance2
                ldi             YH,     high(distance2)

                ldi             ZL, low(distance3)              ; pointer to distance3
                ldi             ZH, high(distance3)

                ld              temp1, x+                               ; load distance 1
                ld              temp2, X

                ld              temp3, Y+                               ; load distance 2
                ld              temp4, Y

                ld              temp5, Z+                               ; load distance 3
                ld              temp6, Z

                cp              temp1, temp3                    ; compare lower byte
                cpc             temp2, temp4                    ; compare upper byte
                BREQ    AllSameTest                             ; if they are equal test if 3 is equal
as well
                BRLO    Next                                    ; if 1<2 branch

                cp              temp3, temp5                    ; compare lower byte
                cpc             temp4, temp6                    ; compare upper byte
                BRLO    Best2                                   ; 2<3
Next:
                cp              temp1, temp5                    ; compare lower byte
                cpc             temp2, temp6                    ; compare upper byte
                BRLO    Best1                                   ; if 2<3

                ldi             XL, low(BestChoice)             ; pointer to BestChoice
                ldi             XH, high(BestChoice)

                ldi             mpr, $03                        ; load 3 into mpr
                st              X, mpr                                  ; store 3 into best choice
                ret

Best1:
                ldi             XL, low(BestChoice)             ; pointer to BestChoice
                ldi             XH, high(BestChoice)

                ldi             mpr, $01                        ; Load 1 into bestchoice
                st              X, mpr
                ret
Best2:
                ldi             XL, low(BestChoice)             ; pointer to bestchoice
                ldi             XH, high(BestChoice)

                ldi             mpr, 2                                  ; load 2 into bestchoice
                st              X, mpr
                ret
AllSameTest:
                cp              temp1, temp5                    ; compare 1 with 3
                cpc             temp2, temp6
```

```
                BREQ    AllSame                                 ; if 1 = 3 branch

AllSame:
                ldi             XL, low(BestChoice)             ; pointer to bestchoice
                ldi             XH,     high(BestChoice)

                ldi             mpr, -1                                 ; load -1 into best choice
                st      X, mpr

                ret                                             ; End a function with RET
;-----------------------------------------------------------
; Func: CalAvg
; Desc: Caculates the average distance between the 3 treasures.
;-----------------------------------------------------------
CalAvg:
                ldi             XL, low(distance1)                              ; pointer to distance 1
                ldi             XH, high(distance1)

                ldi             YL, low(distance2)                              ; pointer to distance 2
                ldi             YH,     high(distance2)

                ldi             ZL, low(ADD_D1D2)                               ; pointer to ADD_D1D2
                ldi             ZH, high(ADD_D1D2)

                ld              temp1, X+                                               ; Load the first
bytes of the first two distances
                ld              temp2, Y+
                add             temp1, temp2                                    ; add them together
                st              Z+, temp1                                               ; store in
result

                ld              temp1, X                                        ; Load the next bytes
                ld              temp2, Y
                adc             temp2, temp1                                    ; add them together with
carry
                st              Z+, temp2

                brcc    EXIT3                                           ; if no carry branch
                st              Z, XH                                           ; store to XH
                clc                                                                     ; clear
flag
EXIT3:
                ldi             XL, low(distance3)                              ; pointer to distance 3
                ldi             XH, high(distance3)

                ldi             YL, low(padded_D3)                              ; pointer to padded_D3
                ldi             YH, high(padded_D3)

                ldi             mpr, $00                                        ; load 0 into mpr

                ld              temp1, X+                                               ; copy distance
3 over
                st              Y+, temp1
                ld              temp1, X
                st              Y+,     temp1
                st              Y, mpr                                          ; add padded
zeros

ADD24:
                ldi             XL, low(ADD_D1D2)                               ; pointer to add_D1D2
                ldi             XH, high(ADD_D1D2)

                ldi             YL, low(padded_D3)                              ; pointer to padded
distance
                ldi             YH, high(padded_D3)

                ldi             ZL, low(ADD24_Res)                              ; pointer to result
                ldi             ZH, high(ADD24_Res)
```

```
                ld              temp1, X+                                       ; load the first
bytes
                ld              temp2, Y+
                add             temp1, temp2                            ; add the two registers
                st              Z+, temp1                                       ; store in
result

                ld              temp1, X+                                       ; load next
bytes
                ld              temp2, Y+
                adc             temp2, temp1                            ; add the two registers
with carry
                st              Z+, temp2                                       ; store in
result

                ld              temp1, X                                ; load next bytes
                ld              temp2, Y
                adc             temp2, temp1                            ; add the two registers
with carry
                st              Z+, temp2                                       ; store in
result

                brcc    EXIT4                                           ; branch if there is a
carry
                st              Z, XH                                           ; store in Z
                clc                                                             ; clear
flags
EXIT4:

                ldi             XL, low(const_3)                ; pointer to const_3
                ldi             XH, high(const_3)

                ldi             mpr, $03                                ; load 3 into a 32-bit
number
                st              X+, mpr
                ldi             mpr, $00
                st              X+, mpr
                st              X+, mpr
                st              X, mpr

                ldi             XL, low(const_0)                ; pointer to const_0
                ldi             XH, high(const_0)

                ldi             mpr, $00                                ; load a 32-bit zero
number
                st              X+, mpr
                st              X+, mpr
                st              X+, mpr
                st              X, mpr

GetAvg:
                rcall   IncCounter                                      ; increment counter
                rcall   SUB32                                           ; subtract total
distance by 3
CheckAvg:
                ldi             XL, low(ADD24_Res)                      ; pointer to total
distance
                ldi             XH, high(ADD24_Res)

                ldi             YL, low(Const_0)                ; pointer to const_0
                ldi             YH, high(Const_0)

                ld              temp1, x+                                       ; compare the
lower bytes
                ld              temp2, Y+
                cp              temp1, temp2

                ld              temp1, X+                                       ; compare next
bytes with carry
```

```
                ld              temp2, Y+
                cpc             temp1, temp2

                ld              temp1, X+                                       ; compare with
carry
                ld              temp2, Y+
                cpc             temp1,temp2

                ld              temp1, X+                                       ; compare last
byte
                ld              temp2, Y+
                cpc             temp1,temp2
                BREQ    EXIT5                                           ; if we have reahed 0
break
                BRGE    GetAvg                                          ; if we have not divided
by 3 all the way keep incrementing
                clc                                                             ; clear
flags
EXIT5:

                ldi             XL, low(SqrtCount)                      ; pointer to counter
                ldi             XH, high(SqrtCount)

                ldi             YL, low(AvgDistance)                    ; pointer to average distance
                ldi             YH, high(AvgDistance)

                ld              temp1, X+                                       ; copy the count
into the avgdistance memory
                st              Y+, temp1
                ld              temp1, X
                st              Y, temp1

                ret                                             ; End a function with RET
;-----------------------------------------------------------
; Func: SUB32
; Desc: Subtracts two 32-bit numbers and copys the result to another
; place in memory.
;-----------------------------------------------------------
SUB32:
                ldi             XL, low(ADD24_Res)              ; pointer to first operand
                ldi             XH,     high(ADD24_Res)

                ldi             YL, low(const_3)        ; pointer to second operand
                ldi             YH, high(const_3)

                ldi             ZL, low(SUB32_Res)              ; pointer to store result
                ldi             ZH,     high(SUB32_Res)

                ld              temp1, X+                               ; load lowest bytes
                ld              temp2, Y+
                sub             temp1, temp2                    ; subtract the two registers
                st              Z+, temp1                               ; store in result

                ld              temp1, X+                               ; load next bytes
                ld              temp2, Y+
                sbc             temp1, temp2                    ; subtract the two registers with carry
                st              Z+, temp1                               ; store in result

                ld              temp1, X+                               ; load next bytes
                ld              temp2, Y+
                sbc             temp1, temp2                    ; subtract the two registers with carry
                st              Z+, temp1                               ; store in result

                ld              temp1, X                        ; load last bytes
                ld              temp2, Y
                sbc             temp1, temp2                    ; subtract the two registers with carry
                st              Z, temp1
                clc
```

```
                ldi             XL, low(ADD24_Res)              ; Set to point towards ADD24_Res
                ldi             XH,     high(ADD24_Res)

                ldi             ZL, low(SUB32_Res)              ; Set to point towards SUB32_Res
                ldi             ZH,     high(SUB32_Res)

                ld              temp1, Z+                               ;Copy over the subtracted value
to the new starting point
                st              X+,     temp1
                ld              temp1, Z+
                st              X+, temp1
                ld              temp1, Z+
                st              X+, temp1
                ld              temp1, Z
                st              X, temp1


                ret                                             ; End a function with RET
;-----------------------------------------------------------
; Func: Check_Neg
; Desc: Checks to see if the signed number is a negative number.
; Converts to positive unsigned if so.
;-----------------------------------------------------------
Check_Neg:
                ldi             XL, low(Check_Neg_addr)         ; pointer to check_neg_addr
                ldi             XH, high(Check_Neg_addr)

                ld              temp1, X+                                       ; point to second byte
                ld              temp1, X

                cpi             temp1, $02                                      ; check if negative bit
is set
                BREQ    Con_Unsigned                            ; if it is then convert
                ret
Con_Unsigned:
                ldi             XL, low(Check_Neg_addr)         ; pointer to check_neg_addr
                ldi             XH, high(Check_Neg_addr)

                ld              temp1, X+                                       ; point towards second
byte
                ld              temp1, X
                ldi             mpr, 0b11111110                         ; turn upper half into 1
                OR              temp1, mpr
                st              X, temp1                                ; store new upper byte

                ldi             XL, low(Check_Neg_addr)         ; pointer to check_neg_addr
                ldi             XH, high(Check_Neg_addr)

                ld              temp1, X+                                       ; load upper and lower
bytes
                ld              temp2, X

                com             temp1                                           ; perform ones
complement on both
                com             temp2

                ldi             XL, low(Check_Neg_addr)         ; pointer to check_neg_addr
                ldi             XH, high(Check_Neg_addr)

                st              X+, temp1                                       ; store the values of
check_neg_addr
                st              X, temp2

                ldi             YL, low(const_1)                ; pointer to const_1
                ldi             YH, high(const_1)

                ldi             mpr, $01                                ; load 1 into const_1
                st              Y+, mpr
                ldi             mpr, $00
```

```
                st              Y, mpr

                ldi             XL, low(Check_Neg_addr)         ; pointer to check_neg_addr
                ldi             XH, high(Check_Neg_addr)

                ldi             YL, low(const_1)                ; pointer to const_1
                ldi             YH, high(const_1)

                rcall ADD16                                                     ; add 1 to
check_neg_addr

                ret                                             ; End a function with RET
;-----------------------------------------------------------
; Func: ADD16
; Desc: Adds two 16-bit registers as well as copy its contents
; to a seperate memory location before clearing itself.
;-----------------------------------------------------------
ADD16:
                ldi             ZL, low(ADD16_Result)   ; Load low byte of result address
                ldi             ZH, high(ADD16_Result)  ; Load high byte of result address

                ld              temp1, X+                               ; Put the low byte of
op1 in temp1
                ld              temp2, Y+                               ; Put the low byte of
op2 in temp2
                add             temp1, temp2                            ; Add the two values
                st              Z+, temp1                               ; Store the result in Z

                ld              temp1, X                        ; Put the high byte of op1 in
temp1
                ld              temp2, Y                        ; Put the high byte of op2 in
temp2
                adc             temp2, temp1                            ; Add with a carry the
two values
                st              Z+, temp2                               ; Store the result in Z

                brcc    EXIT6                                   ; Check if there is a carry
                st              Z, XH                                   ; Store carry
                clc                                                             ; Clear the
carry flag
EXIT6:
                ldi             XL, low(ADD16_Result)           ; pointer to add16 result
                ldi             XH, high(ADD16_Result)

                ldi             YL, low(Check_Neg_addr)         ; pointer to check_neg_addr
                ldi             YH, high(Check_Neg_addr)

                ldi             mpr, $00                                ; set mpr to 0

                ld              temp1, X+                               ; copy the add16 result
to check_neg_addr
                st              Y+, temp1
                ld              temp1, X
                st              Y, temp1

                ldi             XL, low(ADD16_Result)           ; pointer to add16 result
                ldi             XH, high(ADD16_Result)

                st              X+, mpr                                 ; set add16_result to 0
                st              X, mpr

                ret                                             ; End a function with RET

;-----------------------------------------------------------
; Func: Store_Result
; Desc: Stores the found values into the target locations
; specified in the Lab Handout.
;-----------------------------------------------------------
Store_Result:
```

```
                ldi             XL, low(X1_Y1)                                  ; pointer to squared
distance
                ldi             XH, high(X1_Y1)

                ldi             YL,     low(distance1)                          ; pointer to treasure 1
distance
                ldi             YH, high(distance1)

                ldi             ZL, low(Result1)                        ; pointer to target memory
                ldi             ZH, high(Result1)

                ld              temp1, X+                                               ; copy the
squared distance
                st              Z+, temp1
                ld              temp1, X+
                st              Z+, temp1
                ld              temp1, X
                st              Z+, temp1

                ld              temp1, Y+                                               ; copy the
distance
                st              Z+, temp1
                ld              temp1, Y
                st              Z, temp1

Sec_Result:
                ldi             XL, low(X2_Y2)                                  ; pointer to squared
distance
                ldi             XH, high(X2_Y2)

                ldi             YL,     low(distance2)                          ; pointer to treasure 2
distance
                ldi             YH, high(distance2)

                ldi             ZL, low(Result2)                        ; pointer to target memory
                ldi             ZH, high(Result2)

                ld              temp1, X+                                               ; copy the
squared distance
                st              Z+, temp1
                ld              temp1, X+
                st              Z+, temp1
                ld              temp1, X
                st              Z+, temp1

                ld              temp1, Y+                                               ; copy the
distance
                st              Z+, temp1
                ld              temp1, Y
                st              Z, temp1

Third_Result:
                ldi             XL, low(X3_Y3)                                  ; pointer to squared
distance
                ldi             XH, high(X3_Y3)

                ldi             YL,     low(distance3)                          ; pointer to treasure 3
distance
                ldi             YH, high(distance3)

                ldi             ZL, low(Result3)                        ; pointer to target memory
                ldi             ZH, high(Result3)

                ld              temp1, X+                                               ; copy the
squared distance
                st              Z+, temp1
                ld              temp1, X+
                st              Z+, temp1
                ld              temp1, X
```

```
                st              Z+, temp1

                ld              temp1, Y+                                       ; copy the
distance
                st              Z+, temp1
                ld              temp1, Y
                st              Z, temp1
                ret                                             ; End a function with RET


;***end of your code***end of your code***end of your code***end of your code***end of your code***
;****************************** Do not change below this point***********************************
;****************************** Do not change below this point***********************************
;****************************** Do not change below this point***********************************

Grading:
                nop                                     ; Check the results and number of cycles (The TA
will set a breakpoint here)
rjmp Grading


;***********************************************************
;*      Stored Program Data
;***********************************************************

; Contents of program memory will be changed during testing
; The label names (Treasures, UserLocation) are not changed
; See the lab instructions for an explanation of TreasureInfo. The 10 bit values are packed together.
; In this example, the three treasures are located at (5, 25), (35, -512), and (0, 511)
TreasureInfo:   .DB     0x01, 0x41, 0x90, 0x8E, 0x00, 0x00, 0x1F, 0xF0
UserLocation:   .DB 0x00, 0x00, 0x00    ; this is only used for the challenge code

;***********************************************************
;*      Data Memory Allocation for Results
;***********************************************************
.dseg
.org    $0E00                                   ; data memory allocation for results - Your
grader only checks $0E00 - $0E11
Result1:        .byte 5                 ; x2_plus_y2, square_root (for treasure 1)
Result2:        .byte 5                 ; x2_plus_y2, square_root (for treasure 2)
Result3:        .byte 5                 ; x2_plus_y2, square_root (for treasure 3)
BestChoice:             .byte 1                 ; which treasure is closest? (indicate this with
a value of 1, 2, or 3)
                                                                ; this should have a value of -1
in the special case when the 3 treasures
                                                                ; have an equal (rounded)
distance
AvgDistance:    .byte 2                 ; the average distance to a treasure chest (rounded
upward if the value was not already an integer)

;***********************************************************
;*      Additional Program Includes
;***********************************************************
; There are no additional file includes for this program
```