# ECE 375 LAB 8

Introduction to AVR Development Tools

**Lab Time: Friday 2-3:50**

*Bryce Albertazzi*

# INTRODUCTION

The purpose of this lab was to learn how to read in byte packed data and perform operations on it using assembly language. We made a square root function in this lab which turned out to me way harder than I thought, but really tested and developed my assembly skills. This lab involved a lot of register pointer management and moving data around for calculations. A wide variety of assembly instructions were used in this lab such as: st, ldi, ld, mov, inc, dec, branch instructions and more. This was a very comprehensive lab.

# PROGRAM OVERVIEW

This program has a wide variety and quantity of functions and routines. Some of these routines are, sorting out the packed data info from TreasureInfo to memory, calculating X squared plus Y squared, finding the square root, finding the average distance and determining the closest treasure chest. For this program I used more functions and lines of code that all of the other labs by far, there was so much data management which needed to be done.

## ORGANIZE PACKED DATA FROM TREASURE INFO TO MEMORY

In this routine, I used the rol and lsl instructions to shift the bytes the correct number of times to get the 2 most significant bits in a register r0 and the last 8 least significant bits in a register r21. The chest coordinates were packed into consecutive bytes, where each 10 bits represented a coordinate. The program did the same thing for each group of 2 bytes until all of the coordinates were gathered and stored somewhere in data memory.

## CALCULATE X SQUARED PLUS Y SQUARED

For this routine, we took the coordinates stored in memory and performed arithmetic as necessary. I reused the MUL16 and ADD 24 functions from lab 5 to get the perfect squares and add them. I stored X into both MUL16 operands and ran the MUL16 function to get X^2, and then did the same thing for Y to get Y^2. Once X^2 and Y^2 were stored in memory, this routine used the ADD24 operation to add them and get the result for X^2 + Y^2. Then this value was stored into the first 3 bytes of Result1, 2, or 3.

## SQUARE ROOT

This routine took the X^2 plus Y^2 values stored in the Resultx(2..0) and calculated the square root. The square root function kept track of a counter starting at 1 incrementing each loop. In each loop the program would calculate the square root of the counter and compare it to XSqPlusYSq. If the counter squared was bigger, then we've found our rounded square root, so we return the counter. Otherwise, we loop again and increment the counter.

## CALCULATE AVERAGE DISTANCE

This routing takes the calculated square roots stored in Resultx(4..3), adds them up and divides them by 3. First this routing adds Distance1 and Distance2 using ADD16, then it adds the sum of Result1 & Result2 to Result3 with another ADD16 call. Once we have this sum of all the distances, we divide by 3. In the division process, the program keeps track of a counter, in each loop sum of the distances is subtracted by 3 with SUB16. If the result of this operation is less than 3, then we exit the function and return the counter. If not we loop again and increment the counter.

### FIND THE CLOSEST TREASURE CHEST

This is the last routine to get called. At this point we have all of the distances stored in Resultx(4..3). First we compare Distance1 with Distance2, then compare the lesser of Distances 1 and 2 with Distance3. The lowest distance represents the closest treasure chest and is our answer.

## CONCLUSION

This was by far the hardest and most time-consuming lab of the term. However, I feel like my assmembly language skills have developed significantly while completing this lab. Many parts of it were tricky, like finding the square root and the closest chest. I had to get help from the instructor a couple of times. I thought this lab involved a lot of data moving and management to make the program work properly.

## SOURCE CODE

```
;***********************************************************
;*
;*  Bryce_Albertazzi_Lab8_sourcecode.asm
;*
;*  This program find the treasure in a treasure hunt algorithm using assembly code
;*
;*  This is the skeleton file for Lab 8 of ECE 375
;*
;***********************************************************
;*
;*   Author: Bryce Albertazzi
;*     Date: 11/25/20
;*
;***********************************************************
.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

;***********************************************************
;*  Data segment variables
```

```
;*  (feel free to edit these or add others)
;***********************************************************
.dseg
.org    $0100                       ; data memory allocation for operands
Treasure1X: .byte 2
Treasure1Y: .byte 2
Treasure2X: .byte 2
Treasure2Y: .byte 2
Treasure3X: .byte 2
Treasure3Y: .byte 2

.org    $0110
addrA:  .byte 2
addrB:  .byte 2
LAddrP: .byte 4

.org    $0120
XSquared: .byte 4
YSquared: .byte 4
XSqPlusYSq: .byte 3
SquareRoot: .byte 2

.org    $0130
ADD16_OP1: .byte 2
ADD16_OP2: .byte 2
ADD16_Result: .byte 3
SUB16_OP1: .byte 2 ; Will be continuously subtracted by 3 in the division process
until it's value is less than 3
SUB16_OP2: .byte 2 ; Will have a constant value of 3
SUB16_Result: .byte 3 ; This value will be stored, then transferred to SUB16_OP1 for
the next subtraction operation

// Loop goes around 3 times, for each treasure chest, pythagoragen theorem

// Use lab 4 process to store into .byte

// Calculate which treasure is closest in separate function

// Then




;***********************************************************
;*  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

; To do
; your code goes here
    ldi mpr, LOW(RAMEND)
    out spl, mpr
    ldi mpr, HIGH(RAMEND)
    out sph, mpr

;***********************************************************
;* Main Program
;***********************************************************
MAIN:
    ; Store the TreasureInfo values in data memory
    rcall storeChest1XCoordinate
    rcall storeChest1YCoordinate
    rcall storeChest2XCoordinate
    rcall storeChest2YCoordinate
    rcall storeChest3XCoordinate
    rcall storeChest3YCoordinate

    ; Calculate the distance to each treasure chest and store in the required memory
locations
    rcall calculateChest1
    rcall calculateChest2
    rcall calculateChest3

    ; Calculate the average distance to the treasure chests
    rcall FIND_AVERAGE

    ; Determing the closest chest (1, 2, 3) or -1
    rcall BEST_CHOICE

    jmp Grading
DONE: rjmp DONE

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

```
; your code can go here as well

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; TreasureInfo getter functions
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
storeChest1XCoordinate:
    push mpr
    push YL
    push YH
    push ZL
    push ZH
    push r20
    push r21
    push r0
    push r17

    ldi ZL, low(2 * TreasureInfo)
    ldi ZH, high(2 * TreasureInfo)
    ldi YL, low(Treasure1X)
    ldi YH, high(Treasure1X)

    clr r17
    // Load first and second bytes from TreasureInfo into data memory
    lpm mpr, Z+
    mov r21, mpr
    st Y+, mpr
    lpm mpr, Z+
    mov r20, mpr
    st Y+, mpr

    clr r0

    lsl r20
    rol r21
    rol r0
    lsl r20
    rol r21
    rol r0

    // Move Y register back to where it started
    ldi YL, low(Treasure1X)
    ldi YH, high(Treasure1X)
    // Load the shifted value into data memory
    st Y+, r21
    st Y+, r0

    pop r17
    pop r0
```

```
    pop r21
    pop r20
    pop ZH
    pop ZL
    pop YH
    pop YL
    pop mpr
    ret



storeChest1YCoordinate:
    push mpr
    push YL
    push YH
    push ZL
    push ZH
    push r20
    push r21
    push r0
    push r17

    ldi ZL, low(2 * TreasureInfo)
    ldi ZH, high(2 * TreasureInfo)
    ldi YL, low(Treasure1Y)
    ldi YH, high(Treasure1Y)

    // Load first and second bytes from TreasureInfo into data memory
    lpm mpr, Z+
    lpm mpr, Z+
    mov r21, mpr
    st Y+, mpr
    lpm mpr, Z+
    mov r20, mpr
    st Y+, mpr

    clr r0
    clr r17
    // Shift by 2 bits to get desired 10-bit value
    LOOP1Y:
        lsl r20
        rol r21
        inc r17
        cpi r17, 2
        brne LOOP1Y

    lsl r20
    rol r21
```

```
    rol r0
    lsl r20
    rol r21
    rol r0


    // Move Y register back to where it started
    ldi YL, low(Treasure1Y)
    ldi YH, high(Treasure1Y)
    // Load the shifted value into data memory
    st Y+, r21
    st Y+, r0

    pop r17
    pop r0
    pop r21
    pop r20
    pop ZH
    pop ZL
    pop YH
    pop YL
    pop mpr

    ret


storeChest2XCoordinate:
    push mpr
    push YL
    push YH
    push ZL
    push ZH
    push r20
    push r21
    push r0
    push r17

    ldi ZL, low(2 * TreasureInfo)
    ldi ZH, high(2 * TreasureInfo)
    ldi YL, low(Treasure2X)
    ldi YH, high(Treasure2X)

    // Load first and second bytes from TreasureInfo into data memory
    lpm mpr, Z+
    lpm mpr, Z+
    lpm mpr, Z+
    mov r21, mpr
    st Y+, mpr
```

```
    lpm mpr, Z+
    mov r20, mpr
    st Y+, mpr

    clr r0
    clr r17
    // Shift by 2 bits to get desired 10-bit value
    LOOP2X:
        lsl r20
        rol r21
        inc r17
        cpi r17, 4
        brne LOOP2X

    lsl r20
    rol r21
    rol r0
    lsl r20
    rol r21
    rol r0


    // Move Y register back to where it started
    ldi YL, low(Treasure2X)
    ldi YH, high(Treasure2X)
    // Load the shifted value into data memory
    st Y+, r21
    st Y+, r0

    pop r17
    pop r0
    pop r21
    pop r20
    pop ZH
    pop ZL
    pop YH
    pop YL
    pop mpr

    ret

storeChest2YCoordinate:
    push mpr
    push YL
    push YH
    push ZL
    push ZH
    push r20
```

```
    push r21
    push r0
    push r17

    ldi ZL, low(2 * TreasureInfo)
    ldi ZH, high(2 * TreasureInfo)
    ldi YL, low(Treasure2Y)
    ldi YH, high(Treasure2Y)

    // Load first and second bytes from TreasureInfo into data memory
    lpm mpr, Z+
    lpm mpr, Z+
    lpm mpr, Z+
    lpm mpr, Z+
    mov r21, mpr
    st Y+, mpr
    lpm mpr, Z+
    mov r20, mpr
    st Y+, mpr

    clr r0
    clr r17
    // Shift by 2 bits to get desired 10-bit value
    LOOP2Y:
        lsl r20
        rol r21
        inc r17
        cpi r17, 6
        brne LOOP2Y

    lsl r20
    rol r21
    rol r0
    lsl r20
    rol r21
    rol r0


    // Move Y register back to where it started
    ldi YL, low(Treasure2Y)
    ldi YH, high(Treasure2Y)
    // Load the shifted value into data memory
    st Y+, r21
    st Y+, r0

    pop r17
    pop r0
    pop r21
```

```
pop r20
pop ZH
pop ZL
pop YH
pop YL
pop mpr

ret


storeChest3XCoordinate:
push mpr
push YL
push YH
push ZL
push ZH
push r20
push r21
push r0
push r17

ldi ZL, low(2 * TreasureInfo)
ldi ZH, high(2 * TreasureInfo)
ldi YL, low(Treasure3X)
ldi YH, high(Treasure3X)

// Load first and second bytes from TreasureInfo into data memory
lpm mpr, Z+
lpm mpr, Z+
lpm mpr, Z+
lpm mpr, Z+
lpm mpr, Z+
lpm mpr, Z+
mov r21, mpr
st Y+, mpr
lpm mpr, Z+
mov r20, mpr
st Y+, mpr

clr r0
clr r17

lsl r20
rol r21
rol r0
lsl r20
rol r21
rol r0
```

```
    // Move Y register back to where it started
    ldi YL, low(Treasure3X)
    ldi YH, high(Treasure3X)
    // Load the shifted value into data memory
    st Y+, r21
    st Y+, r0

    pop r17
    pop r0
    pop r21
    pop r20
    pop ZH
    pop ZL
    pop YH
    pop YL
    pop mpr

    ret

storeChest3YCoordinate:
    push mpr
    push YL
    push YH
    push ZL
    push ZH
    push r20
    push r21
    push r0
    push r17

    ldi ZL, low(2 * TreasureInfo)
    ldi ZH, high(2 * TreasureInfo)
    ldi YL, low(Treasure3Y)
    ldi YH, high(Treasure3Y)

    // Load first and second bytes from TreasureInfo into data memory
    lpm mpr, Z+
    lpm mpr, Z+
    lpm mpr, Z+
    lpm mpr, Z+
    lpm mpr, Z+
    lpm mpr, Z+
    lpm mpr, Z+
    mov r21, mpr
    st Y+, mpr
    lpm mpr, Z+
```

```
    mov r20, mpr
    st Y+, mpr

    clr r0
    clr r17
    // Shift by 2 bits to get desired 10-bit value
    LOOP3Y:
        lsl r20
        rol r21
        inc r17
        cpi r17, 2
        brne LOOP3Y

    lsl r20
    rol r21
    rol r0
    lsl r20
    rol r21
    rol r0


    // Move Y register back to where it started
    ldi YL, low(Treasure3Y)
    ldi YH, high(Treasure3Y)
    // Load the shifted value into data memory
    st Y+, r21
    st Y+, r0

    pop r17
    pop r0
    pop r21
    pop r20
    pop ZH
    pop ZL
    pop YH
    pop YL
    pop mpr

    ret
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Distance calculation Functions
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
calculateChest1:
    push mpr
    push ZH
    push ZL
```

```
    push YH
    push YL

    ; Load Treasure X coordinate into AddrA & AddrB
    ldi ZL, low(Treasure1X)
    ldi ZH, high(Treasure1X)

    ldi YL, low(AddrA)
    ldi YH, high(AddrA)
    ld  mpr, Z+
    st Y+, mpr
    ld mpr, Z+
    st Y+, mpr

    ldi ZL, low(Treasure1X)
    ldi ZH, high(Treasure1X)

    ldi YL, low(AddrB)
    ldi YH, high(AddrB)
    ld  mpr, Z+
    st Y+, mpr
    ld mpr, Z+
    st Y+, mpr

    ; Call MUL16 function to store product into LAddrP
    rcall MUL16

    ; Store LAddrP into XSquared
    ldi ZL, low(LAddrP)
    ldi ZH, high(LAddrP)

    ldi YL, low(XSquared)
    ldi YH, high(XSquared)
    ld mpr, Z+
    st Y+, mpr
    ld mpr, Z+
    st Y+, mpr
    ld mpr, Z+
    st Y+, mpr
    ld mpr, Z+
    st Y+, mpr

    ; Load Treasure Y coordinate into AddrA & AddrB
    ldi ZL, low(Treasure1Y)
    ldi ZH, high(Treasure1Y)

    ldi YL, low(AddrA)
    ldi YH, high(AddrA)
```

```
    ld  mpr, Z+
    st Y+, mpr
    ld mpr, Z+
    st Y+, mpr

    ldi ZL, low(Treasure1Y)
    ldi ZH, high(Treasure1Y)

    ldi YL, low(AddrB)
    ldi YH, high(AddrB)
    ld  mpr, Z+
    st Y+, mpr
    ld mpr, Z+
    st Y+, mpr

    ; Call MUL16 function to store product into LAddrP
    rcall MUL16

    ; Store LAddrP int YSquared
    ldi ZL, low(LAddrP)
    ldi ZH, high(LAddrP)

    ldi YL, low(YSquared)
    ldi YH, high(YSquared)
    ld mpr, Z+
    st Y+, mpr
    ld mpr, Z+
    st Y+, mpr
    ld mpr, Z+
    st Y+, mpr
    ld mpr, Z+
    st Y+, mpr

    ; Add XSquared + YSquared (24-bit addition) and store sum into
XSquaredPlusYSquared
    rcall ADD24

    ; Calculate square root of XSquaredPlusYSquared and store result into SquareRoot
    rcall SQUARE_ROOT

    ; Store XSqPlusYSq into Result1(2..0)
    ldi ZL, low(XSqPlusYSq)
    ldi ZH, high(XSqPlusYSq)
    ldi YL, low(Result1)
    ldi YH, high(Result1)

    ld mpr, Z+
    st Y+, mpr
```

```
    ld mpr, Z+
    st Y+, mpr
    ld mpr, Z+
    st Y+, mpr

    ; Store SquareRoot into Result1(4..3)
    ld mpr, Z+
    st Y+, mpr
    ld mpr, Z+
    st Y+, mpr

    pop YL
    pop YH
    pop ZL
    pop ZL
    pop mpr

    ret

calculateChest2:
    push mpr
    push ZH
    push ZL
    push YH
    push YL

    ; Load Treasure X coordinate into AddrA & AddrB
    ldi ZL, low(Treasure2X)
    ldi ZH, high(Treasure2X)

    ldi YL, low(AddrA)
    ldi YH, high(AddrA)
    ld  mpr, Z+
    st Y+, mpr
    ld mpr, Z+
    st Y+, mpr

    ldi ZL, low(Treasure2X)
    ldi ZH, high(Treasure2X)

    ldi YL, low(AddrB)
    ldi YH, high(AddrB)
    ld  mpr, Z+
    st Y+, mpr
    ld mpr, Z+
    st Y+, mpr

    ; Call MUL16 function to store product into LAddrP
```

```
rcall MUL16

; Store LAddrP into XSquared
ldi ZL, low(LAddrP)
ldi ZH, high(LAddrP)

ldi YL, low(XSquared)
ldi YH, high(XSquared)
ld mpr, Z+
st Y+, mpr
ld mpr, Z+
st Y+, mpr
ld mpr, Z+
st Y+, mpr
ld mpr, Z+
st Y+, mpr

; Load Treasure Y coordinate into AddrA & AddrB
ldi ZL, low(Treasure2Y)
ldi ZH, high(Treasure2Y)

ldi YL, low(AddrA)
ldi YH, high(AddrA)
ld  mpr, Z+
st Y+, mpr
ld mpr, Z+
st Y+, mpr

ldi ZL, low(Treasure2Y)
ldi ZH, high(Treasure2Y)

ldi YL, low(AddrB)
ldi YH, high(AddrB)
ld  mpr, Z+
st Y+, mpr
ld mpr, Z+
st Y+, mpr

; Call MUL16 function to store product into LAddrP
rcall MUL16

; Store LAddrP int YSquared
ldi ZL, low(LAddrP)
ldi ZH, high(LAddrP)

ldi YL, low(YSquared)
ldi YH, high(YSquared)
ld mpr, Z+
```

```
    st Y+, mpr
    ld mpr, Z+
    st Y+, mpr
    ld mpr, Z+
    st Y+, mpr
    ld mpr, Z+
    st Y+, mpr

    ; Add XSquared + YSquared (24-bit addition) and store sum into
XSquaredPlusYSquared
    rcall ADD24

    ; Calculate square root of XSquaredPlusYSquared and store result into SquareRoot
    rcall SQUARE_ROOT

    ; Store XSqPlusYSq into Result2(2..0)
    ldi ZL, low(XSqPlusYSq)
    ldi ZH, high(XSqPlusYSq)
    ldi YL, low(Result2)
    ldi YH, high(Result2)

    ld mpr, Z+
    st Y+, mpr
    ld mpr, Z+
    st Y+, mpr
    ld mpr, Z+
    st Y+, mpr

    ; Store SquareRoot into Result2(4..3)
    ld mpr, Z+
    st Y+, mpr
    ld mpr, Z+
    st Y+, mpr

    pop YL
    pop YH
    pop ZL
    pop ZL
    pop mpr

    ret

calculateChest3:
    push mpr
    push ZH
    push ZL
    push YH
    push YL
```

```
; Load Treasure X coordinate into AddrA & AddrB
ldi ZL, low(Treasure3X)
ldi ZH, high(Treasure3X)

ldi YL, low(AddrA)
ldi YH, high(AddrA)
ld  mpr, Z+
st Y+, mpr
ld mpr, Z+
st Y+, mpr

ldi ZL, low(Treasure3X)
ldi ZH, high(Treasure3X)

ldi YL, low(AddrB)
ldi YH, high(AddrB)
ld  mpr, Z+
st Y+, mpr
ld mpr, Z+
st Y+, mpr

; Call MUL16 function to store product into LAddrP
rcall MUL16

; Store LAddrP into XSquared
ldi ZL, low(LAddrP)
ldi ZH, high(LAddrP)

ldi YL, low(XSquared)
ldi YH, high(XSquared)
ld mpr, Z+
st Y+, mpr
ld mpr, Z+
st Y+, mpr
ld mpr, Z+
st Y+, mpr
ld mpr, Z+
st Y+, mpr

; Load Treasure Y coordinate into AddrA & AddrB
ldi ZL, low(Treasure3Y)
ldi ZH, high(Treasure3Y)

ldi YL, low(AddrA)
ldi YH, high(AddrA)
ld  mpr, Z+
st Y+, mpr
```

```
    ld mpr, Z+
    st Y+, mpr

    ldi ZL, low(Treasure3Y)
    ldi ZH, high(Treasure3Y)

    ldi YL, low(AddrB)
    ldi YH, high(AddrB)
    ld  mpr, Z+
    st Y+, mpr
    ld mpr, Z+
    st Y+, mpr

    ; Call MUL16 function to store product into LAddrP
    rcall MUL16

    ; Store LAddrP int YSquared
    ldi ZL, low(LAddrP)
    ldi ZH, high(LAddrP)

    ldi YL, low(YSquared)
    ldi YH, high(YSquared)
    ld mpr, Z+
    st Y+, mpr
    ld mpr, Z+
    st Y+, mpr
    ld mpr, Z+
    st Y+, mpr
    ld mpr, Z+
    st Y+, mpr

    ; Add XSquared + YSquared (24-bit addition) and store sum into
XSquaredPlusYSquared
    rcall ADD24

    ; Calculate square root of XSquaredPlusYSquared and store result into SquareRoot
    rcall SQUARE_ROOT

    ; Store XSqPlusYSq into Result3(2..0)
    ldi ZL, low(XSqPlusYSq)
    ldi ZH, high(XSqPlusYSq)
    ldi YL, low(Result3)
    ldi YH, high(Result3)

    ld mpr, Z+
    st Y+, mpr
    ld mpr, Z+
    st Y+, mpr
```

```
    ld mpr, Z+
    st Y+, mpr

    ; Store SquareRoot into Result3(4..3)
    ld mpr, Z+
    st Y+, mpr
    ld mpr, Z+
    st Y+, mpr

    pop YL
    pop YH
    pop ZL
    pop ZL
    pop mpr

    ret
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; Calculate square root, and store result in SquareRoot
SQUARE_ROOT:
    push XH
    push XL
    push YH
    push YL
    push ZH
    push ZL
    push oloop
    push iloop
    push r20
    push r21
    push r22
    push r23


    ldi YL, low(SquareRoot)
    ldi YH, high(SquareRoot)

    ;r20 & r21 will represent the 16-bit square root counter
    clr r20
    clr r21
    clr r23

    SQ_ROOT_ILOOP:
        inc r20
        ; Load r20:r21 into addrA & addrB
        ldi XL, low(addrA)
        ldi XH, high(addrA)
        st X+, r20
```

```
        st X+, r21
        ldi XL, low(addrB)
        ldi XH, high(addrB)
        st X+, r20
        st X+, r21
        ; Perform MUL16 operation and compare result with XSqPlusYSq
        rcall MUL16
        rcall COMPARE_TO_PERFECT_SQUARE
        ; If we've found our square root (if LAddrP >= XSqPlusYSq), exit the function
        cpi r23, 0
        brne SQ_ROOT_DONE
        ; Increment the square root counter
        cpi r20, $ff
        breq INC_HIGH_BYTE
        rjmp SQ_ROOT_ILOOP

    ; When r20 maxex out at ff, clear r20 and increment r21
    INC_HIGH_BYTE:
        clr r20
        inc r21
        rjmp SQ_ROOT_ILOOP

    COMPARE_TO_PERFECT_SQUARE:
        ; Point Z to XSqPlusYSq
        ldi ZL, low(XSqPlusYSq)
        ldi ZH, high(XSqPlusYSq)
        ; Load most significant byte of XSqPlusYSq into mpr
        ld mpr, Z+
        ld mpr, Z+
        ld mpr, Z
        ; Point X to LAddrP, the product of r20:r21 x r20:r21
        ldi XL, low(LAddrP)
        ldi XH, high(LAddrP)
        ; Load most significant byte of LAddrP
        ld r19, X+
        ld r19, X+
        ld r19, X

        ldi r22, 2 ; Counter to determine if we are comparing the lowest bytes in
LAddrP & XSqPlusYSq
        COMPARE_INDIVIDUAL_BYTES:
            cp mpr, r19
            brlo LOAD_SQUARE_ROOT_TO_MEMORY ; If r19 is bigger, we have found our
square root
            breq SHIFT_TO_NEXT_LOWER_BYTES ; If the bytes are the same compare their
next lower respective bytes

            ret ; If mpr is bigger, we still have to increment r20:r21, so return
```

```
            SHIFT_TO_NEXT_LOWER_BYTES:
                ld mpr, -Z
                ld r19, -X
                dec r22
                cpi r22, 0
                breq COMPARE_LOWEST_BYTES
                rjmp COMPARE_INDIVIDUAL_BYTES

            ; If this function is called, we are comparing the lowest bytes in LAddrP
& XSqPlusYSq
            ; if they are equal or LAddrP is gt XSqPlusYSq, we have found our square
root
            COMPARE_LOWEST_BYTES:
                cp r19, mpr
                brsh LOAD_SQUARE_ROOT_TO_MEMORY
                ; If not, run the square root loop
                ret


    LOAD_SQUARE_ROOT_TO_MEMORY:
        st Y+, r20
        st Y+, r21
        inc r23
        ret

    SQ_ROOT_DONE:
        ; Finished Square Root Function

    pop r23
    pop r22
    pop r21
    pop r20
    pop iloop
    pop oloop
    pop ZL
    pop ZH
    pop YL
    pop YH
    pop XL
    pop XH

    ret

; Returns the average of the 3 treasure distances by dividing SumDistances by 3
FIND_AVERAGE:
    push mpr
    push YL
```

```
    push YH
    push ZL
    push ZH
    push XL
    push XH
    push oloop
    push iloop
    push r20
    push r21
    push r22

    ; Load 3 into SUB_16_OP2, this will remain constant throughout the program
    ldi ZL, low(SUB16_OP2)
    ldi ZH, high(SUB16_OP2)
    ldi mpr, $03
    st Z+, mpr
    ldi mpr, $00
    st Z+, mpr

    ; Find the sum of the calculated square roots i.e the sum of the chest distances
    ldi YL, low(Result1)
    ldi YH, high(Result1)
    ldi ZL, low(Result2)
    ldi ZH, high(Result2)
    ldi XL, low(ADD16_OP1)
    ldi XH, high(ADD16_OP1)
        ; Point Y and Z to the fourth bytes of Result1 & Result2 respectively, that's
where the distane is stored
        ld mpr, Y+
        ld mpr, Y+
        ld mpr, Y+
        ld mpr, Z+
        ld mpr, Z+
        ld mpr, Z+
        ; Load distances (fourth & fifth bytes) of Result 1 & 2 into ADD16_OP1 and
ADD16_OP2 respectively
        ld mpr, Y+
        st X+, mpr
        ld mpr, Y+
        st X+, mpr
        ld mpr, Z+
        st X+, mpr
        ld mpr, Z+
        st X+, mpr

        ; Sum chests 1 & 2 distance, then move the result from ADD16_Result to
ADD16_OP1
        rcall ADD16
```

```
        ldi ZL, low(ADD16_Result)
        ldi ZH, high(ADD16_Result)
        ldi YL, low(ADD16_OP1)
        ldi YH, high(ADD16_OP1)
        ld mpr, Z+
        st Y+, mpr
        ld mpr, Z+
        st Y+, mpr

        ; Store chest 3 distance into ADD16_OP2 and run the function again, the sum of
the three will be in ADD16_Result
        ldi ZL, low(Result3)
        ldi ZH, high(Result3)
        ld mpr, Z+
        ld mpr, Z+
        ld mpr, Z+
        ldi YL, low(ADD16_OP2)
        ldi YH, high(ADD16_OP2)
        ld mpr, Z+
        st Y+, mpr
        ld mpr, Z+
        st Y+, mpr

        rcall ADD16

        ; Move the sum of the chests from ADD16_Result into SUB16_OP1 to set up the
division process
        ldi ZL, low(ADD16_Result)
        ldi ZH, high(ADD16_Result)
        ldi YL, low(SUB16_OP1)
        ldi YH, high(SUB16_OP1)
        ld mpr, Z+
        st Y+, mpr
        ld mpr, Z+
        st Y+, mpr

    ; In a loop, subract SUB16_OP2 (value of 3) from SUB16_OP1 until SUB16_OP1 < 3
    ; and keep track of how many times we subtracted, the counter is the quotient (i.e
the average distance).
    clr r20
    clr r21
    clr r22 ; Will exit function when it's value is NOT 0
    DIV_MAIN_LOOP:
        rcall SUB16
        ; Check if SUB16_Result is < 3
        ldi ZL, low(SUB16_Result)
        ldi ZH, high(SUB16_Result)
        ld mpr, Z+
```

```
    ld mpr, Z
    rcall COMPARE_TO_3
    cpi r22, 0
    brne DIV_DONE
    ; Check if low byte is overflowed
    inc r20
    cpi r20, $ff
    breq DIV_HIGH_BYTE_INC
    ; Load SUB16_Result into SUB16_OP1
    ldi ZL, low(SUB16_Result)
    ldi ZH, high(SUB16_Result)
    ldi YL, low(SUB16_OP1)
    ldi YH, high(SUB16_OP2)
    ld mpr, Z+
    st Y+, mpr
    ld mpr, Z+
    st Y+, mpr

    rjmp DIV_MAIN_LOOP

DIV_HIGH_BYTE_INC:
    clr r20
    inc r21
    rjmp DIV_MAIN_LOOP

COMPARE_TO_3:
    cpi mpr, 0
    breq CMP_LOW_BYTE
    ret

CMP_LOW_BYTE:
    ld mpr, -Z
    cpi mpr, 3
    brlo FOUND_QUOTIENT
    ret

; Load the average distance (stored in the counter) into AvgDistance
FOUND_QUOTIENT:
    ldi YL, low(AvgDistance)
    ldi YH, high(AvgDistance)
    st Y+, r20
    st Y+, r21
    inc r22
    ret

DIV_DONE:

pop r22
```

```
    pop r21
    pop r20
    pop iloop
    pop oloop
    pop XH
    pop XL
    pop ZH
    pop ZL
    pop YH
    pop YL
    pop mpr

    ret

BEST_CHOICE:
    push mpr
    push r17
    push YL
    push YH
    push ZL
    push ZH
    push r20
    push r21
    push r22

    ldi r22, -1

    ldi ZL, low(Result1)
    ldi ZH, high(Result1)
    ldi YL, low(Result2)
    ldi YH, high(Result2)

    ldi r20, 1
    ldi r21, 2
    ; If Result1 < Result2 set r22 to 1, otherwise set r22 to 2, if they are the same
leave r22 at 0
    rcall COMPARE_RESULTS
    ; Compare the smaller of Results 1 and 2 (if the same use Result1) to Result3, if
Res3 is smaller set r22 to 3, otherwise leave as is
    rcall SET_UP_SECOND_COMPARISON

    rcall COMPARE_RESULTS

    rjmp CMP_DONE

    ; Result compare logic
    COMPARE_RESULTS:
        ; Point registers at the high bytes of each distance
```

```
    ld mpr, Z+
    ld mpr, Z+
    ld mpr, Z+
    ld mpr, Z+
    ld mpr, Z
    ld r17, Y+
    ld r17, Y+
    ld r17, Y+
    ld r17, Y+
    ld r17, Y
    rcall BC_CP_HIGH
    ret

BC_CP_HIGH:
    cp mpr, r17
    breq BC_CP_LOW
    brlo CHOOSE_FIRST_RESULT
    brsh CHOOSE_SECOND_RESULT

BC_CP_LOW:
    ld mpr, -Z
    ld r17, -Y
    cp mpr, r17
    brlo CHOOSE_FIRST_RESULT
    brsh CHOOSE_SECOND_RESULT

CHOOSE_FIRST_RESULT:
    mov r22, r20
    ret

CHOOSE_SECOND_RESULT:
    cp mpr, r17
    breq EQUAL_RESULTS
    mov r22, r21
    ret

EQUAL_RESULTS:
    ldi r22, -1
    ret

SET_UP_SECOND_COMPARISON:
    cpi r22, 2
    brlo CP_1AND3
    brsh CP_2AND3

CP_1AND3:
    ldi r20, 1
    ldi r21, 3
```

```
        ldi ZL, low(Result1)
        ldi ZH, high(Result1)
        ldi YL, low(Result3)
        ldi YH, high(Result3)
        ret

    CP_2AND3:
        ldi r20, 2
        ldi r21, 3
        ldi ZL, low(Result2)
        ldi ZH, high(Result2)
        ldi YL, low(Result3)
        ldi YH, high(Result3)
        ret

    ; Move r22 into BestChoice and exit function
    CMP_DONE:
        ldi ZL, low(BestChoice)
        ldi ZH, high(BestChoice)
        st Z, r22


    pop r22
    pop r21
    pop r20
    pop ZH
    pop ZL
    pop YH
    pop YL
    pop r17
    pop mpr

    ret

;-----------------------------------------------------------
; Func: ADD16
; Desc: Adds two 16-bit numbers and generates a 24-bit number
;       where the high byte of the result contains the carry
;       out bit.
;-----------------------------------------------------------
ADD16:
        push A
        push B
        push XH
        push XL
        push YH
        push YL
        push ZH
```

```
push ZL
push rhi
push rlo
push oloop
push iloop
push zero

clr zero

; Load beginning address of first operand into X
ldi     XL, low(ADD16_OP1)  ; Load low byte of address
ldi     XH, high(ADD16_OP1) ; Load high byte of address

; Load beginning address of second operand into Y
ldi     YL, low(ADD16_OP2)
ldi     YH, high(ADD16_OP2)

ldi ZL, low(ADD16_Result);
ldi ZH, high(ADD16_Result);

; Execute the function

ld A, X+
ld B, Y+
add A, B
st Z+, A
ld A, X+
ld B, Y+
adc A, B
st Z+, A
clr A
adc zero, A
st Z, A

; Restore variable by popping them from the stack in reverse order
pop zero
pop iloop
pop oloop
pop rlo
pop rhi
pop Zl
pop ZH
pop YL
pop YH
pop XL
pop XH
pop B
pop A
```

```
        ret                     ; End a function with RET

;-----------------------------------------------------------
; Func: SUB16
; Desc: Subtracts two 16-bit numbers and generates a 16-bit
;       result.
;-----------------------------------------------------------
SUB16:
        ; Execute the function here
        push A
        push B
        push XH
        push XL
        push YH
        push YL
        push ZH
        push ZL
        push rhi
        push rlo
        push oloop
        push iloop
        push zero

        clr zero

        ; Load address of first operand into X
        ldi     XL, low(SUB16_OP1)  ; Load low byte of address
        ldi     XH, high(SUB16_OP1) ; Load high byte of address

        ; Load address of second operand into Y
        ldi     YL, low(SUB16_OP2)
        ldi     YH, high(SUB16_OP2)

        ldi ZL, low(SUB16_Result);
        ldi ZH, high(SUB16_Result);

        ; Execute the function
        ld A, Y+
        ld B, X+
        sub B, A
        st Z+, B
        ld A, Y
        ld B, X
        sbc B, A
        st Z+, B
        clr A
        adc zero, A
```

```
        st Z, A


        ; Restore variable by popping them from the stack in reverse order
        pop zero
        pop iloop
        pop oloop
        pop rlo
        pop rhi
        pop Zl
        pop ZH
        pop YL
        pop YH
        pop XL
        pop XH
        pop B
        pop A

        ret                     ; End a function with RET

; Adds XSquared + YSquared, writes the sum into XSqPlusYSq
ADD24:
        push A
        push B
        push XH
        push XL
        push YH
        push YL
        push ZH
        push ZL
        push rhi
        push rlo
        push oloop
        push iloop
        push zero

        clr zero

        ; Load beginning address of first operand into X
        ldi     XL, low(XSquared)   ; Load low byte of address
        ldi     XH, high(XSquared)  ; Load high byte of address

        ; Load beginning address of second operand into Y
        ldi     YL, low(YSquared)
        ldi     YH, high(YSquared)

        ldi ZL, low(XSqPlusYSq);
        ldi ZH, high(XSqPlusYSq);
```

```
        ; Execute the function

        ld A, X+
        ld B, Y+
        add A, B
        st Z+, A

        ld A, X+
        ld B, Y+
        adc A, B
        st Z+, A

        ld A, X+
        ld B, Y+
        adc A, B
        st Z+, A

        clr A
        adc zero, A
        st Z, A

        ; Restore variable by popping them from the stack in reverse order
        pop zero
        pop iloop
        pop oloop
        pop rlo
        pop rhi
        pop Zl
        pop ZH
        pop YL
        pop YH
        pop XL
        pop XH
        pop B
        pop A


        ret                     ; End a function with RET

; Function to perform 16-bit multiplication
MUL16:
        push    A               ; Save A register
        push    B               ; Save B register
        push    rhi             ; Save rhi register
        push    rlo             ; Save rlo register
        push    zero            ; Save zero register
        push    XH              ; Save X-ptr
```

```
        push    XL
        push    YH              ; Save Y-ptr
        push    YL
        push    ZH              ; Save Z-ptr
        push    ZL
        push    oloop           ; Save counters
        push    iloop
        push mpr

        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

        ; clear LAddrP
        clr r17
        CLEAR_LOOP:
            ldi mpr, 0
            st Z+, mpr
            inc r17
            cpi r17, 4
            brne CLEAR_LOOP

        ; 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

        pop mpr
        pop     iloop           ; Restore all registers in reverves order
        pop     oloop
        pop     ZL
        pop     ZH
        pop     YL
        pop     YH
        pop     XL
        pop     XH
        pop     zero
        pop     rlo
        pop     rhi
        pop     B
        pop     A

        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
```