ECE 375 LAB 8

Treasure Hunt

Lab Time: Wednesday 10-12

Tu Lam

INTRODUCTION

The purpose of the lab is to introduce the user to a treasure hunt game. The game is given the user three different coordinates, which the user uses it to determine it distance from the user location. Then it will determine for which one is the shortest one to get to first. And last it will be stored in the location that was given to be store. In this lab the user will also learn about bit shifting and learn to align it.

PROGRAM OVERVIEW

The program overall will run the treasure hunt and it will call the subroutine that was created to run the program. The program will run to determine the right coordinate, find the x^2 + y^2 from the coordinate and calculate the square root. After that it will run the subroutine to find the closest distance and calculate the average distance between the three points.

INITIALIZATION ROUTINE

The initialization routine provides a one-time initialization of the stack pointer to set up where to point in the program and it use the RCALL opcode to call the remaining subroutine.

MAIN ROUTINE

The Main routine will not be executed in this program.

COORDINATE ROUTINE

The Coordinate subroutine will load in the program memory of the coordinates and will determine how it being shifted to get the right align of coordinate from the program memory. From there, store it into data memory so it is easier to access the data later.

COMPUTE_SQUARE ROUTINE

The Compute_Square routine will perform the square of the x and y coordinates. This will go through the data memory of where it being store. This will multiply the x coordinate with itself and multiply the y coordinate by itself to get the value of both and store it in another location in the data memory.

ADD16 ROUTINE

The ADD16 routine is a reuse function from Lab #5 when handling with arithmetic and add the two square numbers together. This function handling adding two 16-bit numbers together and store it in the Result destination that it will need to be store in the spot that they wanted.

ADDITIONAL QUESTIONS

No additional question is needed for this section.

CONCLUSION

For Lab #8, the challenge was to learn and come up with a way that will determine the exact alignment of the coordinate. Also, the lab challenges the way we have been coding in AVR throughout the term and apply the knowledge and see we can do it. Some of the hard stuff such as calculating the square root and dividing is hard as there is no function to do that and we would have to come up with it ourselves. During this lab, I was not able to be finished it in time and got at least one coordinate to work but the time it taken to finish this lab is a lot. But overall, the challenge was fair and moderate level in building your own function.

SUMMARY

For Lab #8, I did not get a chance to finish the entire project, but I will describe what my program is so far. I created a way to rotate and align the first coordinate point in the lab, so it fit perfectly in 2 bytes. Then, I was able to manage to multiply the two together and add it so it can get the square of it. And I was success in doing the addition of the two and store it in the right location. My program stop there as I did not have enough time to finished. Some problems I encounter was dividing store it right in how many bytes as the number get bigger and aligning it to get negative number and converting. But overall, I just did not have enough time to do it and end up with at least 25% of it done.

SOURCE CODE

Provide a copy of the source code. Here you should use a mono-spaced font and can go down to 8-pt in order to make it fit. Sometimes the conversion from standard ASCII to a word document may mess up the formatting. Make sure to reformate the code so it looks nice and is readable.

```
;***********************************************************
;*
;*      Tu_Lam_Lab8_SourceCode.asm
;*
;*      Description: This is a code to find the shortest distance
;*                               of the treasure hunt. And learn to handle the
;*                               10 bits data.
;*
;*
;***********************************************************
;*
;*       Author: Tu Lam
;*         Date: November 29th, 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    dataptr = r19                   ; data ptr
.def    counter = r20                   ; A counter
.def    temp = r21                              ; A temporary register
.def    C = r22
```

```
.def    D = r23

;***********************************************************
;*      Data segment variables
;*      (feel free to edit these or add others)
;***********************************************************
.dseg
.org    $0100                                                   ; data memory allocation for
operands
coordinate1x:           .byte 2                                 ; allocate 2 bytes for a variable named
coordinate1x

.org    $0103
coordinate1y:           .byte 2                                 ; allocate 2 bytes for a variable named
coordinate1y

.org    $0106
coordinate2x:           .byte 2                                 ; allocate 2 bytes for a variable named
coordinate2x

.org    $0109
coordinate2y:           .byte 2                                 ; allocate 2 bytes for a variable named
coordinate2y

.org    $0126
sq1x:                           .byte 2                         ; allocate 2 bytes for a
variable named sq1x

.org    $0129
sq1y:                           .byte 2                         ; allocate 2 bytes for a
variable named sq1y

.org    $012C
resultxy1:                      .byte 2                         ; allocate 2 bytes for a
variable named resultxy1

.org    $012F
tempxy1:                .byte 2                                 ; allocate 2 bytes for a variable named
tempxy1

;***********************************************************
;*      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
                clr             counter

                ldi             mpr, low(RAMEND)                ; Initialize Stack Pointer
                out             SPL, mpr
                ldi             mpr, high(RAMEND)
                out             SPH, mpr

                ; To do
                rcall   Coordinate
                rcall   Compute_Square

                jmp     Grading
```

```
;***********************************************************
;*      Procedures and Subroutines
;***********************************************************
; your code can go here as well

;***********************************************************
;*  Function: Coordinate
;*  Description: This function help to get the right x,y
;*                              for the set of coordinates
;***********************************************************
Coordinate:
                ldi             ZL, low(TreasureInfo << 1)      ; Initialize Z-pointer in .DB for
coordinate
                ldi             ZH, high(TreasureInfo << 1)     ; (multiply by 2) to get to the first
index of the .DB
                ldi             YL, low(coordinate1x)           ; Initialize the character destination
(Data Memory)
                ldi             YH, high(coordinate1x)

                ;Finding X1 coordinate
                clr             zero                                            ; Clear the zero counter
                ldi             counter, 2                                      ; Load 2 into the
counter
                lpm             A, Z+                                           ; Load the program data
into A
                lpm             B, Z                                            ; Load the next data
into B
X1:
                lsl             B                                                       ; Shift the
register B to the left
                rol             A                                                       ; Rotate the A
register to the left
                rol             zero                                            ; Rotate zero with the
carry from A
                dec             counter                                         ; Decrement counter
                brne    X1                                                      ; If counter hasn't
reach 0, keep looping
                st              Y+, zero                                ; Store the 1st byte into Y
                st              Y, A                                            ; Store the 2nd byte

                ; Finding Y1 coordinate
                ldi             YL, low(coordinate1y)           ; Initialize the character destination
(Data Memory)
                ldi             YH, high(coordinate1y)
                clr             counter                                         ; Clear counter

                clr             zero                                            ; Clear the zero counter
                ldi             counter, 4                                      ; Load 4 into the
counter
                lpm             A, Z+                                           ; Load the program data
into A
                lpm             B, Z                                            ; Load the next data
into B
Y1:
                lsl             B                                                       ; Shift the
register B to the left
                rol             A                                                       ; Rotate the A
register to the left
                rol             zero                                            ; Rotate zero with the
carry from A
                dec             counter                                         ; Decrement counter
                brne    Y1                                                      ; If counter hasn't
reach 0, keep looping
                st              Y+, zero                                ; Store the 1st byte into Y
                st              Y, A                                            ; Store the 2nd byte


                ret                                                                     ; Return the
subroutine
```

```
;*********************************************************
;*  Function: Compute_Square
;*  Description: This function help to get the right x,y
;*                              square value of both x,y coordinate
;*********************************************************
Compute_Square:
                ldi             XL, low(coordinate1x)           ; Get the first x coordinate
                ldi             XH, high(coordinate1x)

                ld              A, X+                                           ; Load the value into A
& B
                ld              B, X

                ldi             XL, low(coordinate1y)           ; Get the first y coordinate
                ldi             XH, high(coordinate1y)

                ld              C, X+                                           ; Load the value into C
& D
                ld              D, X+

                ldi             ZL, low(sq1x)                           ; Setup the result of square in
x
                ldi             ZH, high(sq1x)
                ldi             YL, low(sq1y)                           ; Setup the result of square in
y
                ldi             YH, high(sq1y)

                mul             B, B                                            ; Multiply the x
coordinate
                st              Z+, r0                                          ; Store it into the sq1x
                st              Z, r1

                mul             D, D                                            ; Multiply the y
coordinate
                st              Y+, r0                                          ; Store it into the sq1y
                st              Y, r1

                rcall   ADD16                                           ; Call the addition to add the
two together

                ret                                                             ; Return the subroutine

;-----------------------------------------------------------
; 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:
                ; Load beginning address of first operand into X
                ldi             XL, low(sq1x)                   ; Load low byte of address
                ldi             XH, high(sq1x)                  ; Load high byte of address

                ; Load beginning address of second operand into Y
                ldi             YL, low(sq1y)                   ; Load low byte of address into Y-
register
                ldi             YH, high(sq1y)                  ; Load high byte of address into Y-
register

                ; Load beginning address of result into Z
                ldi             ZL, low(Result1)        ; Load low byte of address into Z-register
                ldi             ZH, high(Result1)               ; Load high byte of address into Z-
register

                ; Execute the function
                ld              A, X+                                   ; Load the low byte X into A and
move to the high byte
```

```
                ld              B, Y+                                   ; Load the low byte Y into B and
move to the high byte
                add             A, B                                    ; Add value A and B (low byte)
together and store it in B
                st              Z+, A                                   ; Store value of low byte into Z
and increment to the high byte

                ld              A, X                                    ; Load high byte X into A
                ld              B, Y                                    ; Load high byte Y into B
                adc             A, B                                    ; Add the value of A & B (high
byte) together and store in B w/ carry
                st              Z+, A                                   ; Store value into the high byte
of Z and increment to the next register

                brcc    DONE_JOB                        ; Check if the carry flag is clear, if so, jump
to exit
                ldi             mpr, 1                                  ; If not load 1 into the mpr
                st              Z, mpr                                  ; Store that carry into the Z-
register

DONE_JOB:
                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
```