# ECE 375 LAB 5

*Large Number Arithmetic*

James Stallkamp

## INTRODUCTION

The purpose of this lab is to perform arithmetic operation on numbers that are larger than 8-bits. In this lab we were provided with a procedure that illustrated how could we perform an arithmetic operations that could multiply two 16-bit numbers. And then debug the program in the Atmel studio.

The objective of this lab was to learn how to use arithmetic operation and manipulates data with 16 or more bits using x,y,z-pointers. The projects we have created makes addition function of two 16-bit numbers and produce a 24-bit result and function that multiplies two 24-bit numbers.

## PROGRAM OVERVIEW

This lab required us to create program using AVR assembly code. This process work as follows.

- The program begins with initializing the two stack pointer is load with SPL with low byte of RAMEND, and SPH with high byte of RAMEND.
- Build a function that ADDs and SUBC two 16-bit numbers and generate 24-bit number.
- A function that multiplies two 24-bit numbers and generates a 48-bit number.

### DEFINITIONS

The initial section involves using assembly directives to name constants and variables. Important to note here is register 0 and 1, which have been named rhi and rlo, are the register through 8-bit multiplication is stored. In addition, r2 is designated as the zero register(to maintain zero semantics), and r3 , r4 are designated register A and B to assist in addition, subtraction and multiplication operation. In addition, oloop,r17, will contain a counter for an outer loop, and iloop for an inner loop.

addrA, addrB and LAddrP are the starting addresses of memory locations used to store the results of addition, subtraction and multiplication. These have been kept to preserve the structure of the program; if this program was rewritten, these would be better suited as .byte labels.

## INITIALIZATION ROUTINE

The initialization routine provides a one-time initialization of the two-stack pointer is load with SPL with low byte of RAMEND, and SPH with high byte of RAMEND.

## MAIN ROUTINE

The main routine sets up the Add and subtract function, where it is adds the two 16-bit numbers corresponding to A and B in (A+B). also, subtract two 16-bit numbers corresponding to(A-B).Finally, the multiplication function,MUL24,takes these 24-bit numbers and multiplies them, storing them in another memory location.

## ADD16

The ADD16 routine was a simple exercise in adding two 16-bit numbers and storing it into a 24-bit number .to do this, a specific set of bytes in memory was designated.

## SUB16

The SUB16 routine was a simple exercise in subtracting two 16-bit numbers and storing it into a 24-bit number .to do this, a specific set of bytes in memory was designated.

## MUL16

MUL16 was defined before as an example of how to multiply two 16-bit numbers and store the result into a 24-bit number, this function basically works by performing a sum of multiplicands and storing it in a given memory location.

## MUL24

This is supposed to multiply two 24 bit numbers to create a 48 bit result. Was not completed

## DATA MEMORY

The .DSEG command defines a specific area of memory in which the data memory can be accessed. From here, .ORG designates a specific point beyond which consecutive bytes can be stored.

## ADDITIONAL QUESTIONS:

*1) Although we dealt with unsigned numbers in this lab, the ATmega128 micro-controller also has some features, which are important for performing signed arithmetic. What does the V flag in the status register indicate? Give an example (in binary) of two 8-bit values that will cause the V flag to be set when they are added together.*

The overflow flag (V) is set when the result of adding or subtracting two numbers does not fit within 8 bits. Like 11111111 + 11111111 because the result is larger than can be represented by 8 bits.

*2) In the skeleton le for this lab, the .BYTE directive was used to allocate some data memory locations for MUL16's input operands and result. What are some benefits of using this directive to organize your data memory, rather than just declaring some address constants using the .EQU directive?*

The .EQU instruction produces a named constant that allows y you to write the new constant instead of a constant expression. Whereas .byte actually allocates bytes in memory containing the information.

# CONCLUSION

This lab was very difficult. I was able to implement the more simple add and subtract functions. However I was unable to finish the multiplication function.

## SOURCE CODE

```
;***********************************************************
;*
;*     lab 5
;*
;*     large data arithmetic
;*
;*     This is the skeleton file for Lab 5 of ECE 375
;*
;***********************************************************
;*
;*      Author: James Stallkamp
;*        Date: 10/31/18
;*
;***********************************************************

.include "m128def.inc"                  ; Include definition file

;***********************************************************
;*     Internal Register Definitions and Constants
;***********************************************************
.def   mpr = r16                        ; Multipurpose register
.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   oloop = r17                      ; Outer Loop Counter
.def   iloop = r18                      ; Inner Loop Counter


;***********************************************************
;*     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
             ; Initialize Stack Pointer
             ; TODO                             ; Init the 2 stack pointer registers
             ldi mpr, low(RAMEND)
             out spl, mpr
             ldi mpr, high(RAMEND)
             out sph, mpr
             clr          zero                  ; Set the zero register to zero, maintain
                                                ;       these semantics, meaning, don't
                                                ;       load anything else into it.

;-----------------------------------------------------------
; Main Program
;-----------------------------------------------------------
MAIN:                                           ; The Main program
             ; Setup the ADD16 function direct test

                          ; Move values 0xA2FF and 0xF477 in program memory to data
memory
                          ; memory locations where ADD16 will get its inputs from
                          ; (see "Data Memory Allocation" section below)

               nop ; Check load ADD16 operands (Set Break point here #1)
                          ; Call ADD16 function to test its correctness
                          ; (calculate A2FF + F477)

               nop ; Check ADD16 result (Set Break point here #2)
                          ; Observe result in Memory window

             ; Setup the SUB16 function direct test

                          ; Move values 0xF08A and 0x4BCD in program memory to data
memory
                          ; memory locations where SUB16 will get its inputs from

               nop ; Check load SUB16 operands (Set Break point here #3)
                          ; Call SUB16 function to test its correctness
                          ; (calculate F08A - 4BCD)

               nop ; Check SUB16 result (Set Break point here #4)
                          ; Observe result in Memory window

             ; Setup the MUL24 function direct test
```

```
                        ; Move values 0xFFFFFF and 0xFFFFFF in program memory to data
memory
                        ; memory locations where MUL24 will get its inputs from

              nop ; Check load MUL24 operands (Set Break point here #5)
                        ; Call MUL24 function to test its correctness
                        ; (calculate FFFFFF * FFFFFF)

              nop ; Check MUL24 result (Set Break point here #6)
                        ; Observe result in Memory window

              nop ; Check load COMPOUND operands (Set Break point here #7)
            ; Call the COMPOUND function

              nop ; Check COMPUND result (Set Break point here #8)
                        ; Observe final result in Memory window
                        rcall COMPOUND
DONE:  rjmp   DONE                ; Create an infinite while loop to signify the
                                                        ; end of the program.

;***********************************************************
;*     Functions and Subroutines
;***********************************************************

;-----------------------------------------------------------
; 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:
             clc
             ; 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
             ldi           yl, low(ADD16_OP2)   ; Load low byte of address
             ldi           yh, high(ADD16_OP2)  ; Load high byte of address
             ldi           zl, low(ADD16_Result)       ; Load low byte of address
             ldi           zh, high(ADD16_Result)      ; Load high byte of address


             ; Load beginning address of second operand into Y

             ; Load beginning address of result into Z

             ; Execute the function
             ld a,X+
```

```
                ld b,y+

                add b,a

                st z+,b

                ld a,x
                ld b,y

                adc b,a

                st z+,b

                brcc skip
                ldi mpr,$01
                st z,mpr

skip:  nop
                ret                                     ; End a function with RET

;-----------------------------------------------------------
; Func: SUB16
; Desc: Subtracts two 16-bit numbers and generates a 16-bit
;               result.
;-----------------------------------------------------------
SUB16:
                ; Load beginning 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
                ldi             yl, low(SUB16_OP2)   ; Load low byte of address
                ldi             yh, high(SUB16_OP2)  ; Load high byte of address
                ldi             zl, low(SUB16_Result)       ; Load low byte of address
                ldi             zh, high(SUB16_Result)      ; Load high byte of address


                ; Load beginning address of second operand into Y

                ; Load beginning address of result into Z

                ; Execute the function
                ld a,X+
                ld b,y+

                sub a,b

                st z+,a
```

```
                ld a,x
                ld b,y

                sbc a,b

                st z+,a
                ret                                     ; End a function with RET

;-----------------------------------------------------------
; Func: MUL24
; Desc: Multiplies two 24-bit numbers and generates a 48-bit
;               result.
;-----------------------------------------------------------
MUL24:
                ldi             xl, low(MUL24_OP1)
                ldi             xh, high(MUL24_OP1)
                ldi             yl, low(MUL24_OP2)
                ldi             yh, high(MUL24_OP2)
                ldi             zl, low(MUL24_RESULT)
                ldi             zl, high(MUL24_RESULT)
                ret                                     ; End a function with RET

;-----------------------------------------------------------
; Func: COMPOUND
; Desc: Computes the compound expression ((D - E) + F)^2
;               by making use of SUB16, ADD16, and MUL24.
;
;               D, E, and F are declared in program memory, and must
;               be moved into data memory for use as input operands.
;
;               All result bytes should be cleared before beginning.
;-----------------------------------------------------------
COMPOUND:

                clr mpr
                ldi xl, low(SUB16_RESULT)
                ldi xh, high(SUB16_RESULT)
                st x+, mpr
                st x+, mpr

                ldi xl, low(ADD16_RESULT)
                ldi xh, high(ADD16_RESULT)
                st x+, mpr
                st x+, mpr
                st x+, mpr
                ; Setup SUB16 with operands D and E
                ; Perform subtraction to calculate D - E
```

```
		ldi zl, low(OperandD<<1)
		ldi zh, high(OperandD<<1)
		ldi xl, low(SUB16_OP1)
		ldi xh, high(SUB16_OP1)
		lpm mpr, z+
		st x+,mpr
		lpm mpr,z+
		st x+,mpr

		ldi zl, low(OperandE<<1)
		ldi zh, high(OperandE<<1)
		ldi xl, low(SUB16_OP2)
		ldi xh, high(SUB16_OP2)
		lpm mpr, z+
		st x+,mpr
		lpm mpr,z+
		st x+,mpr

		rcall SUB16
		nop
		; Setup the ADD16 function with SUB16 result and operand F
		; Perform addition next to calculate (D - E) + F
		ldi xl, low(SUB16_Result)
		ldi xh, high(SUB16_Result)
		ldi yl, low(ADD16_OP1)
		ldi yh, high(ADD16_OP1)
		ld mpr, x+
		st y+,mpr
		ld mpr,x+
		st y+,mpr

		ldi zl, low(OperandF<<1)
		ldi zh, high(OperandF<<1)
		ldi xl, low(ADD16_OP2)
		ldi xh, high(ADD16_OP2)
		lpm mpr, z+
		st x+,mpr
		lpm mpr,z+
		st x+,mpr

		rcall ADD16
		; Setup the MUL24 function with ADD16 result as both operands
		; Perform multiplication to calculate ((D - E) + F)^2
		nop
		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:
              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

              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

                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

;-----------------------------------------------------------
; Func: Template function header
; Desc: Cut and paste this and fill in the info at the
;               beginning of your functions
;-----------------------------------------------------------
FUNC:                                           ; Begin a function with a label
                ; Save variable by pushing them to the stack

                ; Execute the function here

                ; Restore variable by popping them from the stack in reverse order
                ret                                     ; End a function with RET
```

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

; Enter any stored data you might need here

; ADD16 operands

; SUB16 operands

; MUL24 operands

; Compoud operands
OperandD:
       .DW    0xFD51                    ; test value for operand D
OperandE:
       .DW    0x1EFF                    ; test value for operand E
OperandF:
       .DW    0xFFFF                    ; test value for operand F

;***********************************************************
;*     Data Memory Allocation
;***********************************************************

.dseg
.org   $0100                        ; data memory allocation for MUL16 example
addrA: .byte 2
addrB: .byte 2
LAddrP:       .byte 4

; Below is an example of data memory allocation for ADD16.
; Consider using something similar for SUB16 and MUL24.

.org   $0110                        ; data memory allocation for operands
ADD16_OP1:
              .byte 2                           ; allocate two bytes for first operand of
ADD16
ADD16_OP2:
              .byte 2                           ; allocate two bytes for second operand
of ADD16

.org   $0120                        ; data memory allocation for results
ADD16_Result:
              .byte 3                           ; allocate three bytes for ADD16 result
```

```
.org   $0130                      ; data memory allocation for operands
SUB16_OP1:
              .byte 2                           ; allocate two bytes for first operand of
SUB16
SUB16_OP2:
              .byte 2                           ; allocate two bytes for second operand
of SUB16

.org   $0140                      ; data memory allocation for results
SUB16_Result:
              .byte 3                           ; allocate three bytes for SUB16 result

; MUL24 data Memory Allocation
.org   $0150
MUL24_OP1:
              .byte 3
MUL24_OP2:
              .byte 3
MUL24_RESULT:
              .byte 6
MUL24_ADDER:
              .byte 6
;***********************************************************
;*     Additional Program Includes
;***********************************************************
; There are no additional file includes for this program
```