ECE 375 LAB 5

Large Number Arithmetic

Lab Time: Wednesday 12-2

Bradley Martin

INTRODUCTION

The purpose of this lab is to better our understanding of arithmetic using registers. Using our knowledge of the AVR instructions and registers we are to use the skeleton code to implement adding 8-bit registers, subtracting 8-bit registers, and multiplying 24-bit registers.

PROGRAM OVERVIEW

The large number arithmetic program provides 4 different functions to compute different operands. There is ADD16 for 16-bit x 16-bit addition, SUB16 for 16-bit x 16-bit, MUL24 for 24-bit x 24-bit multiplication, and COMPOUND which uses all the functions.

INITIALIZATION ROUTINE

The initialization routine first initializes for this program is very short. First the Stack pointer is Initialized. Then the Zero flag resister to set to zero.

MAIN ROUTINE

The main routine calls all the functions with 8 breakpoints to check the results of each function. Each function will load the values from program memory to data memory before calling their respective function. Add16 comes first, then SUB16, MUL24, and COMPOUND.

ADD16

The ADD16 function will add two 16-bit registers and store the result in a 24-bit register. First the high and low of the first values address' are loaded, then the second values address' high and low bytes are load. Next, we load the high and low bytes of the result address'. For the arithmetic we simply add the low bytes first, then the high bytes and add the carry if there is one.

SUB16

The SUB16 function will subtract two 16-bit registers and store the result in a 16-bit register. First the high and low of the first values address' are loaded, then the second values address' high and low bytes are load. Next, we load the high and low bytes of the result address'. For the arithmetic we simply subtract the low bytes first, then the high bytes and subtract the carry.

MUL24

The MUL24 function will multiply two 24-bit registers and store the result in a 48-bit register. Each value is split into three registers with their high and low bytes loaded. Next, we load the address' for the registers that will hold the 48- bit result. For the arithmetic we multiply each register for an operand with each register of the other operand. We add the results of these steps to get our final answer.

COMPOUND

The COMPOUND function will run a pre-determined equation using all the functions. The equation is ((D-E) + F)^2. First, we load the values and address' for the subtraction function. Then we use that result and load the values and address' for the addition function. Finally, we us that result to load the values and address' for the multiplication to complete the square operation.

Additional Questions

1) Although we dealt with unsigned numbers in this lab, the ATmega128 microcontroller 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 V flag is known as the overflow flag and is used to signifies is there is a carry for two's complement arithmetic. An example could be 0b0100 + 0b0100 = 0b1000 will trigger the overflow flag.

2) In the skeleton file 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 .byte directive lets us be more precise especially how we used it in conjunction with .org. We can give the memory address a name rather just a number making it easier to read follow along with the arithmetic.

Conclusion

In this lab we are to use our knowledge of memory management and AVR instructions to implement large number arithmetic. This lab was particularly hard for me in trying to keep track of where all the different pointers were pointing. Towards the end I started to see a clearer picture and furthered my understanding of accessing memory in AVR.

Source Code

```
;***********************************************************
;*
;*      Bradley_Martin_Lab5_sourcecode.asm
;*
;*      Program to handle Large number arithmetic for Lab 5 of ECE 375
;*
;*
;***********************************************************
;*
;*       Author: Bradley Martin
;*         Date: 11/3/2020
;*
;***********************************************************

.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

.def    temp1 = r19                             ; temporary registers for loading
.def    temp2 = r20
.def    temp3 = r21
.def    temp4 = r22


;***********************************************************
;*      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
                ldi             mpr, low(RAMEND)
                out             SPL, mpr
                ldi             mpr, high(RAMEND)
                out             SPH, mpr
                ; TODO                          ; Init the 2 stack pointer registers

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

                                ldi             ZL, low(ADDOP1 << 1)            ; Load low byte of first
operand
                                ldi             ZH, high(ADDOP1 << 1)           ; Load high byte of
first operand
                                lpm             r16, Z+                                         ; Load
low byte to r16
                                lpm             temp1, Z                                ; Load high byte
to temp1

                                ldi             ZL, low(ADDOP2 << 1)            ; Load low byte of
second operand
                                ldi             ZH, high(ADDOP2 << 1)           ; Load high byte of
second operand
                                lpm             temp2, Z+                                       ; Load
low byte to temp2
                                lpm             temp3, Z                                ; Load high byte
to temp3

                                ldi             XL, low(ADD16_OP1)                      ; Load low byte
of op1 memory address
```

```
                        ldi     XH, high(ADD16_OP1)             ; Load high byte
of op1 memory address

                        ldi     YL, low(ADD16_OP2)              ; Load low byte
of op2 memory address
                        ldi     YH, high(ADD16_OP2)             ; Load high byte
of op2 memory address

                        ldi     ZL, low(ADD16_Result)       ; Load low byte of
result memory address
                        ldi     ZH, high(ADD16_Result)      ; Load high byte of
result memory address

                        st      X+, r16                             ; Store
r16 into low byte of op1
                        st      X, temp1                    ; Store temp1
into high byte of op1
                        st      Y+, temp2                           ; Store
temp2 into low byte of op2
                        st      Y, temp3                    ; Store temp3
into high byte of op2

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

                        rcall ADD16

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

        ; Setup the SUB16 function direct test

                        ; Move values 0xFCB9 and 0xE420 in program memory to data memory
                        ; memory locations where SUB16 will get its inputs from

                        ldi     ZL, low(SUBOP1 << 1)        ; Load low byte of first
operand
                        ldi     ZH, high(SUBOP1 << 1)       ; Load high byte of
first operand
                        lpm     r16, Z+                             ; Load
low byte to r16
                        lpm     temp1, Z                    ; Load high byte
to temp1

                        ldi     ZL, low(SUBOP2 << 1)        ; Load low byte of
second operand
                        ldi     ZH, high(SUBOP2 << 1)       ; Load high byte of
second operand
                        lpm     temp2, Z+                           ; Load
low byte to temp2
                        lpm     temp3, Z                    ; Load high byte
to temp3

                        ldi     XL, low(SUB16_OP1)              ; Load low byte
of op1 memory address
                        ldi     XH, high(SUB16_OP1)             ; Load high byte
of op1 memory address

                        ldi     YL, low(SUB16_OP2)              ; Load low byte
of op2 memory address
                        ldi     YH, high(SUB16_OP2)             ; Load high byte
of op2 memory address

                        ldi     ZL, low(SUB16_Result)       ; Load low byte of
result memory address
                        ldi     ZH, high(SUB16_Result)      ; Load high byte of
result memory address
```

```
                                st              X+, r16                                         ; Store
r16 into low byte of op1
                                st              X, temp1                                ; Store temp1
into high byte of op1
                                st              Y+, temp2                                       ; Store
temp2 into low byte of op2
                                st              Y, temp3                                ; Store temp3
into high byte of op2

                nop ; Check load SUB16 operands (Set Break point here #3)
                                ; Call SUB16 function to test its correctness
                                ; (calculate FCB9 - E420)

                                rcall SUB16

                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

                                ;Use three registers to load from memory split into 0xFF,0xFF,0xFF

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

                                ;rcall MUL24

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

                                ldi             ZL, low(OperandD << 1)          ; Load low byte of first
operand
                                ldi             ZH, high(OperandD << 1)         ; Load high byte of
first operand
                                lpm             r16, Z+                                         ; Load
low byte to r16
                                lpm             temp1, Z                                ; Load high byte
to temp1

                                ldi             ZL, low(OperandE << 1)          ; Load low byte of
second operand
                                ldi             ZH, high(OperandE << 1)         ; Load high byte of
second operand
                                lpm             temp2, Z+                                       ; Load
low byte to temp2
                                lpm             temp3, Z                                ; Load high byte
to temp3

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

                                rcall COMPOUND

                nop ; Check COMPUND result (Set Break point here #8)
                                ; Observe final result in Memory window

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

                ldi             YL, low(ADD16_OP2)      ; Load low byte of second value address
                ldi             YH, high(ADD16_OP2)     ; Load high byte of second value address

                ldi             ZL, low(ADD16_Result)   ; Load low byte of result address
                ldi             ZH, high(ADD16_Result)  ; Load high byte of result address

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

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

                brcc    EXIT                                    ; Check if there is a carry
                st              Z, XH                                   ; Store carry
                clc                                                             ; Clear the carry flag
EXIT:
                ret                                             ; End a function with RET

;-------------------------------------------------------
; Func: SUB16
; Desc: Subtracts two 16-bit numbers and generates a 16-bit
;               result.
;-------------------------------------------------------
SUB16:
                ldi             XL, low(SUB16_OP1)      ; Load low byte of first value address
                ldi             XH, high(SUB16_OP1)     ; Load high byte of first value address

                ldi             YL, low(SUB16_OP2)      ; Load low byte of second value address
                ldi             YH, high(SUB16_OP2)     ; Load high byte of second value address

                ldi             ZL, low(SUB16_Result)   ; Load low byte of result address
                ldi             ZH, high(SUB16_Result)  ; Load high byte of result address

                ld              A, X+                                   ; Put the low byte of op1 in A
                ld              B, Y+                                   ; Put the low byte of op2 in B
                sub             A, B                                    ; Subtract B from A
                st              Z+, A                                   ; Store the result in Z

                ld              A, X                                    ; Put the high byte of op1 in A
                ld              B, Y                                    ; Put the high byte of op2 in B
                sbc             A, B                                    ; Subtract B from A with carry
                st              Z+, A                                   ; Store the result in Z

                clc                                                             ; Clear carry flag

                ret                                             ; End a function with RET

;-------------------------------------------------------
; Func: MUL24
; Desc: Multiplies two 24-bit numbers and generates a 48-bit
;               result.
;-------------------------------------------------------
MUL24:
                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
MUL24_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, 3        ; Load counter
MUL24_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

                ;add a carry for Z

                ;store A, rlo, rhi each at -Z

                adiw    ZH:ZL, 1        ; Z <= Z + 1
                dec             iloop                   ; Decrement counter
                brne    MUL24_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    MUL24_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


                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:

                ; Setup SUB16 with operands D and E
                ; Perform subtraction to calculate D - E
                ldi             XL, low(COMP_OP1)       ; Load low byte of first value address
                ldi             XH, high(COMP_OP1)      ; Load high byte of first value address

                ldi             YL, low(COMP_OP2)       ; Load low byte of second value address
                ldi             YH, high(COMP_OP2)      ; Load high byte of second value address

                ldi             ZL, low(COMP_Result)    ; Load low byte of result address
                ldi             ZH, high(COMP_Result)   ; Load high byte of result address

                ld              A, X+                                   ; Put the low byte of op1 in A
                ld              B, Y+                                   ; Put the low byte of op2 in B
                sub             A, B                                    ; Subtract B from A
                st              Z+, A                                   ; Store the result in Z

                ld              A, X                                    ; Put the high byte of op1 in A
                ld              B, Y                                    ; Put the high byte of op2 in B
                sbc             A, B                                    ; Subtract B from A with carry
                st              Z+, A                                   ; Store the result in Z



                ; Setup the ADD16 function with SUB16 result and operand F
                ; Perform addition next to calculate (D - E) + F

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

                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
ADDOP1:
                .DW 0xFCBA                      ; test value for ADD operand 1
ADDOP2:
                .DW 0xFFFF                      ; test value for ADD operand 2

; SUB16 operands
SUBOP1:
                .DW 0xFCB9                      ; test value for SUB operand 1
SUBOP2:
                .DW 0xE420                      ; test value for SUB operand 2

; MUL24 operands
MULOP1:
                .DW 0xFFFFFF            ; test value for MUL operand 1
MULOP2:
                .DW 0xFFFFFF            ; test value for MUL operand 2
; Compoud operands
OperandD:
        .DW     0xFCBA                          ; test value for operand D
OperandE:
        .DW     0x2019                          ; test value for operand E
OperandF:
        .DW     0x21BB                          ; test value for operand F

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

.dseg
.org    $0100                           ; data memory allocation for MUL16/MUL24 example
addrA:  .byte 2
addrB:  .byte 2
LAddrP: .byte 6

; 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 result
SUB16_Result:
                .byte 3                         ; allocate three bytes for SUB16 result

.org    $0150                           ; data memory allocation for operands
MUL24_OP1:
                .byte 3                         ; allocate three bytes for first operand of MUL24
MUL24_OP2:
                .byte 3                         ; allocate three bytes for second operands of MUL24
.org    $0160
MUL24_Result:                           ; data memory allocation for result
                .byte 6                         ; allocate six bytes for result of MUL24

.org    $0170                           ; data memory allocation for operands
COMP_OP1:
                .byte 2                         ; allocate two bytes for first operand of COMPOUND
COMP_OP2:
                .byte 2                         ; allocate two bytes for second operand of COMPOUND
COMP_OP3:
                .byte 2                         ; allocate two bytes for thrid operand of COMPOUND
COMP_OP4:
                .byte 2                         ; allocate two bytes for fourth operand of COMPOUND
.org    $0180                           ; data memory allocation for result
COMP_Result:
                .byte 6                         ; allocate six bytes for result of COMPOUND
;***********************************************************
;*      Additional Program Includes
;***********************************************************
; There are no additional file includes for this program
```