ECE 375 Lab 4

Large Number Arithmetic

Lab Time: Friday 2 PM - 3:50 PM

Student 1: Winnie Woo
Student 2: Joseph Borisch

TA Signature

1 Introduction

The purpose of the lab is to be able to use the simulator and set breakpoints to observe what happens to certain values. As well as understanding how to use ALU instructions (ADD, SUB, MUL etc.) and knowing how to handle numbers that are larger than 8 bits.

2 Program Overview

The program provides the behavior to do 16-bit addition, 16-bit subtraction, 24-bit multiplication and a compound function to provide the correct inputs for ADD, SUB and MUL. Besides the standard INIT and MAIN routines within the program, four additional routines were created to load the functions for the ADD, SUB, MUL and COMPOUND (G-H) + I)$\hat{2}$ operands. As well as data memory allocation for the functions.

3 Initialization Routine

The initialization routine initializes the stack pointer and sets the zero register to zero.

4 Main Routine

Within the MAIN routine there are 8 subroutine calls: LoadADD, ADD16, LoadSUB, SUB16, LoadMUL, MUL24, LoadCCOMPOUND, COMPOUND to either load the operands or call the function to display the results.

5 LoadADD

Loads 2 bytes of data into the addresses specified by ADD16_OP1 and ADD16_OP2

6 ADD16

Takes in the data from LoadADD and calculates the addition results by setting the pointers in the X (memory), Y and Z (result) register. Then loaded into registers A and B and added with X, Y registers post incremented. The addition results are then stored in the first address. The add with carry bit is used if it is necessary. Then if the carry bit is set, a 1 is written to the address in Z.

7 LoadSUB

Loads 2 bytes of data into SUB16_OP1 and SUB16_OP2

8 SUB16

Similar to ADD16 function except instead of ADC, SBC is used for subtraction.

9 LoadMUL

Loads the values from program memory for multiplication.

10 MUL24

Saved the current state of the registers (A, B, rhi, rlo) by pushing them to the stack and cleared the zero register. Then loaded the operand C address into Z and the first byte of the operand 2 address into Y. Then set the outer loop count to 3, within the outer loop we are loading the first byte of MUL24_OP1 into X. The inner loop count is also set to 3, and within the inner loop we are grabbing the first bytes of the A and B operand and multiplying then getting the result byte from memory and storing the results in rlo, rhi, then grabbing a third byte from the result. Finally the Z address is shifted to the second byte and the inner loop is decremented.

11 LoadCOMPOUND

Unable to complete.

12 COMPOUND

Unable to complete.

13 Additional Questions

1. Although we dealt with unsigned numbers in this lab, the ATmega32 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 the two's complement overflow flag which is triggered when 2 two's complement numbers are added together and the result overflows and has the opposite sign. The example would overflow because the two's complement range only supports -128 t0 127.

   ```
       01100100 (100)
   +   01111100 (124)
   ----------------
       11100000 (224)
   ```

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?

Benefits of using the .BYTE directive is that memory is not being used so you can use the directive to set an address that can be used to store data.

14 Difficulties

Having an example to look at for the MUL24 function was helpful but that was the part we had difficulty with, it was hard to figure out what the proper steps to take were.

15 Conclusion

In conclusion, having that original planned extra week would have been nice but the previous lab took up too much time with midterms coming up but it was good practice to use the debugger.

16 Source Code

```
;***********************************************************
;* This is the skeleton file for Lab 4 of ECE 375
;*
;*  Author: Joseph Borisch and Winnie Woo
;*  Date: 10/28/2022
;*
;***********************************************************

.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 $0056           ; End of Interrupt Vectors

;-----------------------------------------------------------
; Program Initialization
;-----------------------------------------------------------
INIT:                ; The initialization routine

     ; Initialize Stack Pointer
     ldi mpr, low(RAMEND)
     out SPL, mpr ; Load SPL with low byte of RAMEND
     ldi mpr, high(RAMEND)
     out SPH, mpr ; Load SPH with high byte of RAMEND

     ; TODO

     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

     ; Call function to load ADD16 operands
     nop ; Check load ADD16 operands (Set Break point here #1)
     rcall LoadADD

     ; Call ADD16 function to display its results (calculate FCBA + FFFF)
     nop ; Check ADD16 result (Set Break point here #2)
     rcall ADD16

     ; Call function to load SUB16 operands
     nop ; Check load SUB16 operands (Set Break point here #3)
     rcall LoadSUB

     ; Call SUB16 function to display its results (calculate FCB9 - E420)
     nop ; Check SUB16 result (Set Break point here #4)
     rcall SUB16

     ; Call function to load MUL24 operands
     nop ; Check load MUL24 operands (Set Break point here #5)
     rcall LoadMUL

     ; Call MUL24 function to display its results (calculate FFFFFF * FFFFFF)
```

```
    nop ; Check MUL24 result (Set Break point here #6)
    rcall MUL24

    ; Setup the COMPOUND function direct test
    nop ; Check load COMPOUND operands (Set Break point here #7)
    rcall LoadCOMPOUND

    ; Call the COMPOUND function
    nop ; Check COMPOUND result (Set Break point here #8)
    rcall COMPOUND

DONE: rjmp DONE      ; Create an infinite while loop to signify the
                     ; end of the program.

;***********************************************************
;* Functions and Subroutines
;***********************************************************
;-----------------------------------------------------------
; Func: LoadADD
; Desc: Load ADD16 values from program memory
;-----------------------------------------------------------
LoadADD:
    ; Load OperandA into Z
    ldi ZL, low(OperandA << 1)
    ldi ZH, high(OperandA << 1)

    ; 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)

    ; Load bytes from program memory into data memory at X address
    lpm A, Z+
    st X+, A
    lpm A, Z
    st X, A

    ; Load OperandB into Z
    ldi ZL, low(OperandB << 1)
    ldi ZH, high(OperandB << 1)

    ; Load bytes from program memory into data memory at Y address
    lpm A, Z+
    st Y+, A
    lpm A, Z
    st Y, A
```

```
     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:
     ; 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)

     ; Load beginning address of result into Z
     ldi ZL, low(ADD16_Result)
     ldi ZH, high(ADD16_Result)

     ; Execute the function
     ; Add low byte of A & B
     ld A, X+ ; load low byte into A
     ld B, Y+ ; load high byte into B
     add A, B ; add bytes
     st Z+, A ; store byte addition
     ; Add high byte of A & B
     ld A, X    ; load low byte into A
     ld B, Y    ; load high byte into B
     ADC A, B
     st Z+, A
     brcc ADD_EXIT
     st Z, XH

     ADD_EXIT:
       ret                ; End a function with RET
;-----------------------------------------------------------
; Func: LoadSUB
; Desc: Load SUB16 values from program memory
;-----------------------------------------------------------
LoadSUB:
     ; Load Operand C address into Z
     ldi ZL, low(OperandC << 1)
     ldi ZH, high(OperandC << 1)

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

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

    ; Load bytes from program memory into data memory at X address
    lpm A, Z+
    st X+, A
    lpm A, Z
    st X, A

    ; Load Operand D into Z
    ldi ZL, low(OperandD << 1)
    ldi ZH, high(OperandD << 1)

    ; Load bytes from program memory into data memory at Y address
    lpm A, Z+
    ST Y+, A
    lpm A, Z
    ST Y, A

    ret     ; End a function with RET

;-----------------------------------------------------------
; Func: SUB16
; Desc: Subtracts two 16-bit numbers and generates a 16-bit
;      result. Always subtracts from the bigger values.
;-----------------------------------------------------------
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

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

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

    ; Execute the function
    ld A, X+ ; load low byte into A
    ld B, Y+ ; load high byte into B
    sub A, B ; subtract bytes
    st Z+, A ; store byte addition

    ; Add high byte of A & B
    ld A, X   ; load low byte into A
```

```
	ld B, Y    ; load high byte into B
	sub A, B ;
	st Z+, A
	;brcc SUB_EXIT
	;st Z, XH


	;SUB_EXIT:
	  ret                ; End a function with RET

;-----------------------------------------------------------
; Func: LoadMUL
; Desc: Adds two 16-bit numbers and generates a 24-bit number
;      where the high byte of the result contains the carry
;      out bit.
;-----------------------------------------------------------
LoadMUL:
	; Load Operand E1 address into Z
	ldi ZL, low(OperandE1 << 1)
	ldi ZH, high(OperandE2 << 1)

	; Load beginning address of first operand into X
	ldi XL, low(MUL24_OP1)
	ldi XH, high(MUL24_OP1)

	; Load beginning address of first operand into Y
	ldi YL, low(MUL24_OP2)
	ldi YH, high(MUL24_OP2)

	; Load bytes from program memory into data memory at X address
	lpm A, Z+
	st X+, A
	lpm A, Z+
	st X+, A
	lpm A, Z
	st X+, A

	; Load Operand E2 into Z
	ldi ZL, low(OperandF1 << 1)
	ldi ZH, high(OperandF2 << 1)

	; Load bytes from program memory into data memory at Y address
	lpm A, Z+
	ST Y+, A
	lpm A, Z+
	ST Y+, A
	lpm A, Z
	st Y+, A
```

```
    ret     ; End a function with RET

;-----------------------------------------------------------
; Func: MUL24
; Desc: Multiplies two 24-bit numbers and generates a 48-bit
;       result.
;-----------------------------------------------------------
MUL24:
;* - Simply adopting MUL16 ideas to MUL24 will not give you steady results. You should
   come up with different ideas.
     ; Execute the function here

    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

    ; Load Operand C address into Z
    ldi ZL, low(MUL24_RESULT)
    ldi ZH, high(MUL24_RESULT)

    ; Load beginning address of first operand into Y
    ldi YL, low(MUL24_OP2)  ;loading the first byte of operand2 into Y
    ldi YH, high(MUL24_OP2)

    ; Begin outer for loop
    ldi     oloop, 3   ; Load counter

MUL24_OLOOP:
    ; Load beginning address of first operand into X
    ldi XL, low(MUL24_OP1)  ;loading first byte of operand1 into x
    ldi XH, high(MUL24_OP1)

    ; 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
    ld   B, Z
    adc     B, zero       ;need carry for the carry / overflow with multiplication

    st   Z, B       ; Store third byte to memory
    st   -Z, A
    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 MUL24_ILOOP   ; Loop if iLoop != 0
    ; End inner for loop

    sbiw ZH:ZL, 2   ; Z <= Z - 2
    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

;-----------------------------------------------------------
; 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.
;-----------------------------------------------------------
LoadCOMPOUND:
  ret
```

```
;-----------------------------------------------------------
; Func: COMPOUND
; Desc: Computes the compound expression ((G - H) + I)^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 G and H
      ; Perform subtraction to calculate G - H

      ; Setup the ADD16 function with SUB16 result and operand I
      ; Perform addition next to calculate (G - H) + I

      ; Setup the MUL24 function with ADD16 result as both operands
      ; Perform multiplication to calculate ((G - H) + I)^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
;* Do not section.
;***********************************************************
; ADD16 operands
OperandA:
   .DW 0xFCBA
OperandB:
   .DW 0xFFFF

; SUB16 operands
OperandC:
   .DW 0XFCB9
OperandD:
   .DW 0XE420

; MUL24 operands
OperandE1:
   .DW   0XFFFF
OperandE2:
   .DW   0X00FF
OperandF1:
   .DW   0XFFFF
OperandF2:
   .DW   0X00FF

; Compoud operands
```

```
OperandG:
   .DW  0xFCBA          ; test value for operand G
OperandH:
   .DW  0x2022          ; test value for operand H
OperandI:
   .DW  0x21BB          ; test value for operand I

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

; SUB16 memory allocation
.org  $0130
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 2         ; allocate two bytes for SUB16 result

; MUL24 memory allocation
.org  $0150
MUL24_OP1:
      .byte 3         ; allocate 3 bytes for first operand of MUL24
MUL24_OP2:
      .byte 3         ; allocate 3 bytes for second operand of MUL24

.org  $0160        ; data memory allocation for results
MUL24_Result:
      .byte 6         ; allocate 6 bytes for MUL24 result
```

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