ECE 375 LAB 5

Large Number Arithmetic

Lab Time: Tuesday 4-6

Alexander Uong

INTRODUCTION

This purpose of this lab is to learn how to implement large number arithmetic, such as adding 16 bit, subtracting 16 bit, and multiplying 24 bit numbers using our AVR board. The lab also requires we store our operands in program memory, but move the operands to data memory prior to using arithmetic on the operands. This builds off on lab 4, where data manipulation is implemented. In general, this lab allowed me to get more comfortable with programming in assembly and accessing memory. This lab also allowed me to get more comfortable with the debugger, as that's the only way we can check the results of our functions.

PROGRAM OVERVIEW

In this program, we were asked to implement four functions, add16, sub16, mul24, and a compound function. Values of the operands that will be used are stored in program memory initially, however prior to executing the arithmetic functions on the operands, the operands are moved from program memory into data memory. Once this has been completed, the arithmetic functions can be called, storing the result in data memory as well. Prior the program executing operands are stored in program data with particular values already assigned for each operand. Addresses to store the operands are pre defined in the data memory allocation section.

In the program , moving the values of the operands from program memory to data memory is done. The addresses of the operands are stored into the X and Y registers. The value of the operands are stored into the Z register and are loaded into registers r16 and r17, r16 for the lower 8 bytes and r17 for the higher 8 bytes. From then, the registers are stored into the X and Y registers, which are pointing at data memory. This process is essentially done for all of the functions.

INITIALIZATION ROUTINE

Only initialization involved in INIT is initializing the stack pointer.

MAIN ROUTINE

Moving value of operands from program memory to data memory and calling respective arithmetic functions to compute result.

SUBROUTINES

1. ADD16 Routine

   The ADD16 Routine is called once the operands are stored into data memory. Within the function, the addresses of where the operands are stored are defined and loaded into the X,Y, and Z register. X and Y point to operand 1 and 2 respectively, as Z register points to where the result will be stored. I then add the lower bytes of operand 1 and 2, store into r17, and store that into the lower Z register. The higher bytes of operand 1 and 2 are then added with carry, and stored into the higher byte of the z register.

2. SUB16 Routine

The SUB16 Routine is very similar to the ADD16 routine. Like the ADD16 routine, the addresses of where the operands are stored are defined and loaded into the X,Y, and Z register. X and Y point to operand 1 and 2 respectively, as Z register points to where the result will be stored. The lower byte of operand 2 is subtracted from the lower byte of operand 1 and stored into the lower Z register. The higher byte of operand 2 is subtracted with carry from the higher byte of operand 1 and stored into the higher Z register.

3. COMPOUND Routine

   This routine is essentially a combination of the above routines, except you're instead passing in the compound operands in place of the usual add, subtraction, and multiplication operands. For example, to subtract D-E, the compound operands are passed into the subtraction function. To add (D-E)+F, the subtraction result is passed in to the first add operand, and F is passed in as the second add operand.

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 in the status register is two's complement overflow indicator. The overflow indicator will never be set for same signed numbers. An example of an example of two 8-bit binary values that will cause the V flag to be set are adding -7 and 20 together.

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 benefits of using this directive is that it the user is able to the data memory locations of the operands and result using a user specified name, making it more readable and easier to understand. It allows the user to conveniently place the operands and result next to each other in memory, making it easier to debug and check if the operands are successfully getting stored into data memory, and if the correct result is being produced.

CONCLUSION

In conclusion, this lab taught me a lot regarding using the built in arithmetic calls to create larger arithmetic subroutines. This lab also allowed me to get a lot more comfortable regarding data manipulation and using the debugger, which are definitely areas I'm not completely comfortable yet.

SOURCE CODE

```
;***********************************************************
;*
;*     Alexander_Uong_Lab5_sourcecode.asm
;*
;*     Enter the description of the program here
;*
;*     This is the skeleton file for Lab 5 of ECE 375
;*
;***********************************************************
;*
;*      Author: Alexander Uong
;*        Date: 2/5/21
```

```
;*
;***********************************************************

.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

                LDI mpr, LOW(RAMEND)    ; Low Byte of End SRAM Address
                OUT SPL, mpr                    ; Write Byte to SPL
                LDI mpr, HIGH(RAMEND)   ; High Byte of End SRAM Address
                OUT SPH, mpr                    ; Write Byte to SPH

                ; 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             XL, low(ADD16_OP1)      ; X register pointer points
to low byte of address of operand1 in data memory
                                ldi             XH, high(ADD16_OP1)     ; X register pointer points
to high byte of address of operand1 in data memory
                                ldi             ZL, LOW(OperandAdd1 << 1)   ; Z register pointer
points to low byte of Add16 operand 1 in program memory
                                ldi             ZH, HIGH(OperandAdd1 << 1) ; Z register pointer
points to high byte of Add16 operand 1 in program memory

                                lpm             mpr, Z+         ; Load data from Z register to R16.
Post increment for Z to point at high byte
                                lpm             R17, Z          ; Load data from Z register to R17
```

```
                              st             X+, mpr        ; Store R16, the low byte of the
operand, into lower x register, which points at operand1 address location
                              st             X, R17         ; Store R17, the high byte of the
operand, into the higher x register

                              ldi            YL, low(ADD16_OP2)     ; Y register pointer points
to low byte of address of operand2 in data memory
                              ldi            YH, high(ADD16_OP2)    ; Y register pointer points
to high byte of address of oeprand2 in data memory
                              ldi            ZL, LOW(OperandAdd2 << 1)  ; Z register pointer
points to low byte of Add16 operand 2 in program memory
                              ldi            ZH, HIGH(OperandAdd2 << 1) ; Z register pointer
points to high byte of Add16 operand 2 in program memory

                              lpm            mpr, Z+        ; Load data from Z register to R16.
Post increment for Z to point at high byte
                              lpm            R17, Z         ; Load data from Z register to R17
                              st             Y+, mpr        ; Store R16, the low byte of the
operand, into lower Y register, which points at operand2 address location
                              st             Y, R17         ; Store R17, the high byte of the
operand, into the higher Y register


                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            XL, low(SUB16_OP1)     ; X register pointer points
to low byte of address of operand1 in data memory
                              ldi            XH, high(SUB16_OP1)    ; X register pointer points
to high byte of address of operand1 in data memory
                              ldi            ZL, LOW(OperandSub1 << 1)  ; Z register pointer
points to low byte of SUB16 operand 1 in program memory
                              ldi            ZH, HIGH(OperandSub1<< 1) ; Z register pointer
points to low byte of SUB16 operand 1 in program memory

                              lpm            mpr, Z+        ; Load data from Z register to R16.
Post increment for Z to point at high byte
                              lpm            R17, Z         ; Load from Z register to R17
                              st             X+, mpr        ; Store R16, the low byte of the
operand, into lower X register, which points at operand1 address location
                              st             X, R17         ; Store R17, the high byte of the
operand, into the higher x register

                              ldi            YL, low(SUB16_OP2)     ; Y register pointer points
to low byte of address of operand2 in data memory
                              ldi            YH, high(SUB16_OP2)    ; Y register pointer points
to high byte of address of operand2 in data memory
                              ldi            ZL, LOW(OperandSub2 << 1)  ;  Z register pointer
points to low byte of SUB16 operand 2 in program memory
                              ldi            ZH, HIGH(OperandSub2 << 1) ;  Z register pointer
points to high byte of SUB16 operand 2 in program memory

                              lpm            mpr, Z+        ; Load data from Z register to R16.
Post increment for Z to point at high byte
                              lpm            R17, Z         ; Load data from Z register to R17
                              st             Y+, mpr        ; Store R16, the low byte of the
operand, into lower Y register, which points at operand2 address location
                              st             Y, R17         ; Store R17, the high byte of the
operand, into the higher Y register
```

```
                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
                      /*
                              ldi            XL, low(MUL24_OP1)     ; Load low byte of address
                              ldi            XH, high(MUL24_OP1)    ; Load high byte of address
                              ldi            ZH, HIGH(OperandMul1<< 1) ; Load high byte of
operand into high byte of register
                              ldi            ZL, LOW(OperandMul1 << 1)  ; Load low byte of
operand into high byte of register

                              lpm            mpr, Z+                         ; Load from Z
register to R16
                              lpm            R17, Z+                         ; Load from Z
register to R17
                              lpm            R18, Z                          ; Load from Z
register to R18
                              st             X+, R16        ; Store R16 into where X points with
post increment
                              st             X+, R17        ; Store R17 into where X points with
post increment
                              st             X+, R18        ; Store R18 into where X points with
post increment

                              ldi            YL, low(MUL24_OP2)     ; Load low byte of address
                              ldi            YH, high(MUL24_OP2)    ; Load high byte of address
                              ldi            ZH, HIGH(OperandMul2 << 1) ; Load high byte of
operand into high byte of register
                              ldi            ZL, LOW(OperandMul2 << 1)  ; Load low byte of
operand into high byte of register


                              lpm            mpr, Z+                         ; Load from Z
register to R19
                              lpm            R17, Z+                         ; Load from Z
register to R20
                              lpm            R18, Z                          ; Load from Z
register to R21
                              st             Y+, mpr        ; Store R19 into where Y points with
post increment
                              st             Y+, r17        ; Store R20 into where Y points with
post increment
                              st             Y+, R8         ; Store R21 into where Y points with
post increment
               */


                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            XL, low(COMP_OP1)      ; X register pointer points
to low byte of address of COMP operand1 in data memory
                              ldi            XH, high(COMP_OP1)     ; X register pointer points
to high byte of address of COMP operand1 in data memory
```

```
                              ldi            ZL, LOW(OperandD << 1)  ; Z register pointer points
to low byte of  COMP operandD in program memory
                              ldi            ZH, HIGH(OperandD << 1) ; Z register pointer points
to high byte of COMP operandD  in program memory

                              lpm            mpr, Z+        ; Load data from Z register to R16.
Post increment for Z to point at high byte
                              lpm            R17, Z         ; Load from Z register to R17
                              st             X+, mpr        ; Store R16, the low byte of the
operand, into lower X register, which points at operand1 address location
                              st             X, R17         ; Store R17, the high byte of the
operand, into the higher X register

                              ldi            XL, low(COMP_OP2)      ; X register pointer points
to low byte of address of COMP operand2 in data memory
                              ldi            XH, high(COMP_OP2)     ; X register pointer points
to high byte of address of COMP operand2 in data memory
                              ldi            ZL, LOW(OperandE << 1)  ; Z register pointer points
to low byte of COMP operandE in program memory
                              ldi            ZH, HIGH(OperandE << 1) ; Z register pointer points
to low byte of COMP operandE in program memory

                              lpm            mpr, Z+         ; Load data from Z register to R16.
Post increment for Z to point at high byte
                              lpm            R17, Z          ; Load from Z register to R17
                              st             X+, mpr        ; Store R16, the low byte of the
operand, into lower X register, which points at operand2 address location
                              st             X, R17         ; Store R17, the high byte of the
operand, into the higher X register

                              ldi            XL, low(COMP_OP3)      ; X register pointer points
to low byte of address of COMP operand3 in data memory
                              ldi            XH, high(COMP_OP3)     ; X register pointer points
to high byte of address of COMP operand3 in data memory
                              ldi            ZL, LOW(OperandF << 1)  ; Z register pointer points
to low byte of COMP operandF  in program memory
                              ldi            ZH, HIGH(OperandF << 1) ; Z register pointer points
to low byte of COMP operandF in program memory

                              lpm            mpr, Z+                         ; Load data from Z
register to R16. Post increment for Z to point at high byte
                              lpm            R17, Z                          ; Load from Z
register to R17
                              st            X+, mpr                          ; Store R16, the low
byte of the operand, into lower X register, which points at operand3 address location
                              st            X, R17                           ; Store R17, the
high byte of the operand, into the higher x register


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

```
               ; Load beginning address of first operand into X
               ldi            XL, low(ADD16_OP1)     ;X register pointer points to low byte of
address of operand1 in data memory
               ldi            XH, high(ADD16_OP1)    ;X register pointer points to high byte of
address of operand1 in data memory

               ; Load beginning address of second operand into Y
               ldi            YL, low(ADD16_OP2)     ;Y register pointer points to low byte of
address of operand1 in data memory
               ldi            YH, high(ADD16_OP2)    ;Y register pointer points to high byte of
address of operand1 in data memory

               ; Load beginning address of result into Z
               ldi            ZL, low(ADD16_Result) ;Z register pointer points to low byte of
address of result in data memory
               ldi            ZH, high(ADD16_Result);Z register pointer points to high byte of
address of result in data memory

               ; Execute the function
               ld             R16, X+       ;load high byte of operand1
               ld             R17, Y+       ;load low byte of operand2

               add            R17, R16      ;add contents inside r17 and r16 and store in r17
               st             Z+, R17       ;store r17 into lower z register

               ld             R16, X        ;load high byte of operand1
               ld             R17,Y         ;load high byte of operand2

               adc            R17,R16       ;add with carry
               st             Z+, R17       ;store in higher z register

               brcc    EXIT

               st             Z,XH
               EXIT:
                       ret                                          ; End a function with RET

;-----------------------------------------------------------
; Func: SUB16
; Desc: Subtracts two 16-bit numbers and generates a 16-bit
;              result.
;-----------------------------------------------------------
SUB16:
               ; Execute the function here

               ; Load beginning address of first operand into X
               ldi            XL, low(SUB16_OP1)           ;X register pointer points to low
byte of address of operand1 in data memory
               ldi            XH, high(SUB16_OP1)          ;X register pointer points to high
byte of address of operand1 in data memory

               ; Load beginning address of second operand into Y
               ldi            YL, low(SUB16_OP2)     `      ;Y register pointer points to low
byte of address of operand2 in data memory
               ldi            YH, high(SUB16_OP2)          ;Y register pointer points to high
byte of address of operand2 in data memory

               ; Load beginning address of result into Z
               ldi            ZL, low(SUB16_Result) ;Z register pointer points to low byte of
address of result in data memory
               ldi            ZH, high(SUB16_Result);Z register pointer points to high byte of
address of result in data memory

               ; Execute the function
               ld             mpr, X+ ;store low byte of first number and post increment to point
at high byte of first number
               ld             R17, Y+ ;store low byte of second number and post increment to
point at high byte of second number

               sub            mpr,R17 ;subtract low byte of second number from low  byte of first
number
```

```
               st             Z+, mpr ;store result in Z and post increment to point at high byte
of result

               ld             mpr, X  ;store high byte of first number into r16
               ld             R17, Y  ;store high byte of second number into r17

               sbc            mpr,R17 ;subtract with carry high byte of second number from high
byte of first number
               st             Z+,mpr  ;store in high byte of result



               ret                                         ; End a function with RET

;------------------------------------------------------------
; Func: MUL24
; Desc: Multiplies two 24-bit numbers and generates a 48-bit
;              result.
;------------------------------------------------------------
MUL24:
               ; 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

               ; Set Y to beginning address of B
               ldi            XL, low(MUL24_OP1);
               ldi            XH, high(MUL24_OP1)

               ; Set Z to begginning address of resulting Product
               ldi            ZL, low(MUL24_Result)  ; Load low byte
               ldi            ZH, high(MUL24_Result); Load high byte

               ; Begin outer for loop
               ldi            oloop, 3               ; Load counter
MUL24_OLOOP:
               ; Set X to beginning address of A
               ldi            XL, low(MUL24_OP2);
               ldi            XH, high(MUL24_OP2)

               ; Begin inner for loop
               ldi            iloop, 3               ; Load counter, changed from 2 to 3
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
               st             Z, A                   ; Store third byte to memory
               st             -Z, rhi                ; Store second byte to memory
               st             -Z, rlo                ; Store third 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 - 1 changed from 1 to 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: 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)     ;X register pointer points to low byte of
address of comp operand1 in data memory
               ldi           XH, high(COMP_OP1)    ;X register pointer points to high byte of
address of comp operand1 in data memory
               ldi           ZL, low(SUB16_OP1)    ;Z register pointer points to low byte of
address of sub16 operand in data memory
               ldi           ZH, high(SUB16_OP1)   ;Z register pointer points to high byte of
address of sub16 operand in data memory

               ld            mpr, X+                      ; Load data from X register to R16.
Post increment for X to point at high byte
               ld            R17, X
               st            Z+, mpr                      ;store result in Z and post increment
to point at high byte of result
               st            Z, R17

               ldi           XL, low(COMP_OP2)     ;same above except for second operand
               ldi           XH, high(COMP_OP2)
               ldi           ZL, low(SUB16_OP2)
               ldi           ZH, high(SUB16_OP2)

               ld            mpr, X+                      ;same above except for second operand
               ld            R17, X
               st            Z+, mpr
               st            Z, R17

               rcall SUB16                                ;call sub16


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

               ldi            XL, low(SUB16_Result)  ;X register pointer points to low byte of
address of sub16_result in data memory
               ldi            XH, high(SUB16_Result) ;X register pointer points to high byte of
address of sub16_result in data memory
               ldi            ZL, low(ADD16_OP1)            ;Z register pointer points to low
byte of address of add16 operand1 in data memory
               ldi            ZH, high(ADD16_OP1)           ;Z register pointer points to low
byte of address of add16 operand1 in data memory

               ld             mpr, X                                ; Load data from X register
to R16. Post increment for X to point at high byte
               ld             R17, X
               st             Z+, mpr                               ;store result in Z and post
increment to point at high byte of result
               st             Z, R17

               ldi            XL, low(COMP_OP3)             ;same as above except for comp
operand 3
               ldi            XH, high(COMP_OP3)
               ldi            ZL, low(ADD16_OP2)
               ldi            ZH, high(ADD16_OP2)

               ld             mpr, X+                               ;same as above except for
comp operand 3
               ld             R17, X
               st             Z+, mpr
               st             Z, R17

               rcall ADD16                                          ;call add16



               ; 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

OperandAdd1:
        .DW 0xFCBA
OperandAdd2:
        .DW 0xFFFF

; SUB16 operands
OperandSub1:
        .DW 0xFCB9
OperandSub2:
        .DW 0xE420

; MUL24 operands
OperandMul1:
        .DW 0xFFFFFF
OperandMul2:
        .DW 0xFFFFFF

; 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 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 $0140
SUB16_OP1:
               .byte 2
SUB16_OP2:
               .byte 2

.org    $150
SUB16_Result:
        .byte 2



.org $0158
MUL24_OP1:
               .byte 3
MUL24_OP2:
               .byte 3

.org    $0170
```

```
MUL24_Result:
        .byte 6

.org    $0180                         ; data memory allocation for operands
COMP_OP1:
                .byte 2                       ; allocate three bytes for first operand of COMP
COMP_OP2:
                .byte 2                       ; allocate three bytes for second operand of COMP
COMP_OP3:
                .byte 2                       ; allocate three bytes for third operand of COMP

.org    $0190                         ; data memory allocation for results
COMP_Result:
                .byte 6

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