ECE 375 Lab 4

Large Number Arithmetic

Lab session: 015
Time: 12:00-13:50

Author: Astrid Delestine
Programming partner: Lucas Plastid

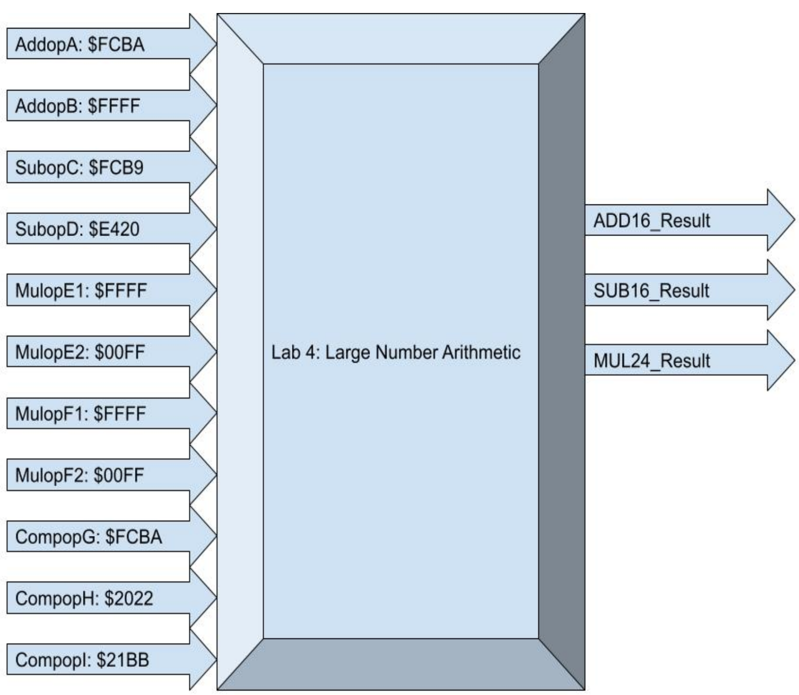
TA Signature

1 Introduction

This is the fourth lab in the ECE 375 series and it covers adding and subtracting words as well as multiplying words together. In this sense words designate 16 bit numbers. It is important to note that this assembly was written for the m128 chipset and not the regular atmetga32u4 chipset. This is due to the fact that we can operate a simulation tool when using the m128 chipset that is not available when we use our regular chipset. The student will write their assembly to do these expected operations with large numbers and is given the expected inputs and can calculate the outputs.

2 Design

This lab Lucas and I collaborated and built out ideas for multi-byte addition, subtraction, and multiplication. It was quite interesting to see our ideas collide and how different approaches can be taken to the same problem. Addition can be considered very trivial, however subtraction becomes a little more difficult as the programmer has to consider negative values and if they are possible with this code. In the handout it clearly states that in this lab we will only be dealing with a more simple unsigned subtraction so the result can be held in two 8 bit registers. Next the multiplication of two 24 bit numbers. This will be more difficult than the previous two problems. In this case we decided to solve the problem linearly and not worry about looping or recursion. Finally we move on to the compound function, which uses each of the previously implemented methods to solve the problem of $((G - H) + I)^2$.



3 Assembly Overview

As for the Assembly program an overview can be seen below. It is also important to note that for each of the subroutines all the registers used are pushed and popped to and from the stack unless otherwise stated.

3.1 Internal Register Definitions and Constants

The multipurpose register was setup as r16. At r0 and r1 any multiplication output will be set, such that the outputs of any multiplication operation are automatically assigned to them. A default zero register is set to r2. Two other generic variable registers are defined as r3 and r4. Finally two registers named oloop and iloop are used for counting within the assembly itself.

3.2 Initialization Routine

Firstly the stack pointer is initialized, then the register defined above as the zero register is cleared.

3.3 Main Routine

The main operations that happen in the main routine are those that are initialized below and are called in this order. Firstly the adding operations take place, those being LOADADD16 and ADD16. Next the functions associated with the subtraction operation are called, those being LOADSUB16 and SUB16. Next the functions referencing the MUL24 operation are called. In order they are called LOADMUL24 and MUL24. Finally the compound function set is called, being LOADCOMPOUND and COMPOUND. Once all of these functions have been called the main method loops at the done flag, determining the program to be complete.

3.4 Subroutines

3.5 ADD16

This subroutine adds two 16 bit numbers together. It does this by taking each X Y and Z registers and setting them to operator 1 operator 2 and the location to save the result respectively. Next, using the A variable register as an intermediary the two inputs were added together and saved into the Z register location. Finally if the carry flag is set at the end of the whole operation then we can assume the most significant bit needs to be set to 1, so it is done.

3.6 SUB16

This subroutine takes two different inputs and subtracts the first from the second. The X Y and Z registers are initialized at the beginning of this subroutine first, the same way that they are in the add subroutine. In the actual operation part of this subroutine A and B registers are loaded with the lower values of X and Y and subtracted from each other. This is then preformed again to account for the second word. Due to the fact that we do not need to worry about signage these are each saved directly to the location pointed to by Z

3.6.1 MUL24

This subroutine takes 4 different data locations and multiplies 24 bits by 24 bits, resulting in a 48 bit number. It must be built differently to the MUL16 operation. Every time addition occurs we need to check for the carry bit and pass it forward if necessary. This will continue until there is no carry bit to pass upward. In reality this can only ever happen up to 4 times. In this subroutine the first operand is loaded into the Z pointer. Then the second operand is loaded into the Y pointer, finally the result is loaded into the X register. For each of these, they load the start of each because they will increment throughout the method. The data in Y an Z are multiplied and the result is stored in r0 and r1. ADDMUL2x is then called. This fixes the carry bit problem of multiplying by 24 bits and as long as we call ADDMUL2x after our multiplication then everything will work out.

3.6.2 ADDMUL2x

This subroutine adds a partial multiplication result to the location x is pointing to. This presumes that x is already pointing to the location where the low result of the current multiplication needs to go. Essentially it takes the multiplication outputs and cycles the carry bit up until it cannot anymore. It utilized a loop moving the carry byte in and out of X when necessary.

3.6.3 COMPOUND

Preforms the operation $((G - H) + I)^2$ Using multiplication, addition and subtraction.

3.6.4 CLRRES

Clears the result memory locations for each ADD16 SUB16 and MUL24. Makes heavy use of the zero register

3.6.5 LOADMUL24, LOADADD16, LOADSUB16, & LOADCOMPOUND

These subroutines have all been combined as they all do essentially the same operation, they just differ in the data they point to. They load all of the numbers used in each subroutine called above from data memory into program memory so that they are more easily accessible.

3.6.6 MUL16

A pre-supplied basis subroutine that multiplies 2 16 bit numbers together.

3.6.7 FUNC

A pre-supplied boiler plate function template that each subroutine is based off of.

4 Testing

Tested Each input value and compared to external calculations.

Case	Expected	Actual meet expected
$\$FCBA + \$FFFF$	\$01FCB9	✓
$\$FCB9 - \$E420$	\$1899	✓
$\$00FFFFFF * \$00FFFFFF$	\$FFFFFE000001	✓
$((\$FCBA - \$2022) + \$21BB)^2$	\$FCA8CEE9	✓

Table 1: Assembly Testing Cases

5 Study Questions

1. Although we dealt with unsigned numbers in this lab, the ATmega32 micro-controller also has some features which are important for performing signed arithmetic. What does the V flag in the status register indicate? Give an example (in binary) of two 8-bit values that will cause the V flag to be set when they are added together.

   The V flag is the 2's complement overflow indicator, this would be useful when two negative values are being added or subtracted for example 0b1000_0000 - 0b1000_0000. This result would no longer fit in the 2's complement space, and so the V flag would be thrown

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?

   Using the .BYTE directive allows you to not only assume the space is empty but also it allows you to pre-allocate a size of the space that .equ does not.

3. In computing, there are traditionally two ways for a microprocessor to listen to other devices and communicate: polling and interrupts. Give a concise overview/description of each method, and give a few examples of situations where you would want to choose one method over the other.

   Polling is essentially when the computer is constantly asking a device for its status. This might be good when plotting a data stream that is constantly flowing into the computer. Interrupts allow the computer to work on other tasks and if a button or certain signal is received, it stops whatever it is doing, goes and runs the triggered task, and returns to whatever it was doing prior. This would be very good for a mouse and keyboard.

4. Describe the function of each bit in the following ATmega32U4 I/O registers: EICRA, EICRB, and EIMSK. Do not just give a brief summary of these registers; give specific details for each bit of each register, such as its possible values and what function or setting results from each of those values. Also, do not just directly paste your answer from the datasheet, but instead try to describe these details in your own words.

   in EICRA bits 0 and 1 determine if the signal is detected on a rising edge, falling edge, any edge, or a low level. These control the first 4 interrupts in a fashion such that interrupt 0 is the 0th and 1st bits, interrupt 1 is the 2nd and 3rd bits, and so on. EICRB is very similar except it only handles external interrupt 6, on its 4th and 5th bits. all of its other bits are reserved. EIMSK is an interrupt mask, to enable an interrupt it must be enabled here, the interrupt is associated with its bit, ie bit 6 masks for interrupt 6. This is active high.

5. The ATmega32U4 microcontroller uses interrupt vectors to execute particular instructions when an interrupt occurs. What is an interrupt vector? List the interrupt vector (address) for each of the following ATmega32U4 interrupts: Timer/Counter0 Overflow, External Interrupt 6, and Analog Comparator.

   The atmega32u4 has 43 different reset and interrupt vectors an interrupt vector is a trigger that allows a certain function to be run. Timer/Counter0:$002E External Interrupt 6:$000E and Analog Comparator:$0038

6. Microcontrollers often provide several different ways of configuring interrupt triggering, such as level detection and edge detection. Suppose the signal shown in Figure 1 was connected to a microcontroller pin that was configured as an input and had the ability to trigger an interrupt based on certain signal conditions. List the cycles (or range of cycles) for which an external interrupt would be triggered if that pin's sense control was configured for: (a) rising edge detection, (b) falling edge detection, (c) low level detection, and (d) high level detection. Note: There should be no overlap in your answers, i.e., only one type of interrupt condition can be detected during a given cycle.

   (a) rising edge detection: 7,14
   (b) falling edge detection: 2,11
   (c) low level detection: $3 \rightarrow 6$, $12 \rightarrow 13$
   (d) high level detection: 1, $8 \rightarrow 10$, 15

6 Difficulties

This lab was more challenging than the last, however it did not require us to learn anything outside of lecture. This is a good thing, due to the fact that we re only expected to know exactly what we are taught.

7 Conclusion

This lab cemented the ideas of logical operands and allowed the student to understand how computers operate with large numbers, especially larger numbers than they might be able to handle naively. Additionally the pencil and paper method described in the handout was not how I was taught how to do multiplication, so the solution may be more or less difficult depending on the students type of education.

8 Source Code

Listing 1: Assembely Bump Bot Script

```
;***********************************************************
;*  This is the skeleton file for Lab 4 of ECE 375
;*
;*   Author: Astrid Delestine and Lucas Plastied
;*      Date: 2/16/2023
;*
;***********************************************************

.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
        ldi     mpr, high(RAMEND)
        out     SPH, mpr
        ; 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
        rcall LOADADD16
        ; Operands stored in $0110 and $0112
        nop ; Check load ADD16 operands (Set Break point here #1)
        rcall ADD16
        ; Call ADD16 function to display its results
        ;(calculate FCBA + FFFF)
        ; Result stored in $0120, should be $1FCB9
        nop ; Check ADD16 result (Set Break point here #2)


        ; Call function to load SUB16 operands
        rcall LOADSUB16
        ; Operands stored in $0114 and $0116
        nop ; Check load SUB16 operands (Set Break point here #3)

        ; Call SUB16 function to display its results
        ;(calculate FCB9 - E420)
        rcall SUB16
        ; Result stored in $0130, should be $1899
        nop ; Check SUB16 result (Set Break point here #4)


        ; Call function to load MUL24 operands
        rcall LOADMUL24
        ; Operands stored in $0118 and $011B
        nop ; Check load MUL24 operands (Set Break point here #5)

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

```
        rcall MUL24
        ; Result stored in $0140, should be $FFFFFE000001
        nop ; Check MUL24 result (Set Break point here #6)

        ; Setup the COMPOUND function direct test
        rcall LOADCOMPOUND
        ; Operands stored in
        ;$0114 (G = $FCBA),
        ;$0116 (H = $2022), and
        ;$0112 (I = $21BB)
        nop ; Check load COMPOUND operands (Set Break point here #7)

        ; Call the COMPOUND function, ((G-H)+I)^2
        rcall COMPOUND
        ; Result stored in $0140, should be $0000FCA8CEE9
        nop ; Check COMPOUND result (Set Break point here #8)

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:
        push mpr
        push A
        push XH
        push YH
        push ZH
        push XL
        push YL
        push ZL

        clr mpr
        bclr 0 ; CLEAR THE CARRY FLAG!!! (just in case lamp)
        ; 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)  ; Load low byte of address
        ldi     YH, high(ADD16_OP2) ; Load high byte of address
        ; Load beginning address of result into Z
        ldi     ZL, low(ADD16_Result) ; points the end of Z
        ldi     ZH, high(ADD16_Result)
        ; Execute the function
        ;2 16 bit numbers being added generates a max of 24 bit number
        ; ie 1111_1111_1111_1111
        ;   +1111_1111_1111_1111
        ;   --------------------
        ;   1_1111_1111_1111_1110
        ; can do 8 bits at a time



        ;add 2 lowest bytes
        ld A, X+
        add mpr, A
        ld A, Y+
        add mpr, A
        ;mpr now equals x + y & carry flag is included
        ;store this lower result in the lowest memory of the result
        st Z+, mpr
        clr mpr

        ld A, X+
        adc mpr, A
        ld A, Y+
        add mpr, A
        ;mpr now equals x + y & carry flag is included
        ;store this lower result in the lowest memory of the result
        st Z+, mpr
        clr mpr


        ;if c flag is set
        brcc noCarry;
        ldi mpr, $01;
        st Z, mpr
noCarry:
        pop ZL
        pop YL
        pop XL
        pop ZH
        pop YH
```

```
        pop XH
        pop A
        pop mpr
        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:
        ; Execute the function here
        push mpr
        push A
        push B
        push XH
        push YH
        push ZH
        push XL
        push YL
        push ZL

        ; 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)   ; Load low byte of address
        ldi     YH, high(SUB16_OP2)  ; Load high byte of address
        ; Load beginning address of result into Z
        ldi     ZL, low(SUB16_Result) ; points the end of Z
        ldi     ZH, high(SUB16_Result)

        ld A, X+        ; Load low byte of OP1 into A,
                        ; X now points at high byte
        ld B, Y+        ; Load low byte of OP2 into B,
                        ; Y now points at high byte
        sub A, B        ; Subtract low byte of OP2 from low byte of OP1
        st Z+, A        ; Store result into low byte of SUB16_Result,
                        ; Z now points go high byte
        ld A, X         ; Load high byte of OP1 into A
        ld B, Y         ; Load high byte of OP2 into B
        sbc A, B        ; Subtract high byte of OP2 from
                        ; low byte of OP1 with carry

        st Z, A         ; Store result to high byte of SUB16_Result

```

```
        pop ZL
        pop YL
        pop XL
        pop ZH
        pop YH
        pop XH
        pop B
        pop A
        pop mpr
        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.
/*
Imagine we are multiplying two 24-bit numbers, ABC and DEF.
A is the highest byte, C is the lowest byte. So A is referring
to only the highest byte of ABC. Imagine that when multiplying
C and F, the result is CFH:CFL where CFH and CFL are the high
and low bytes respectively of the result of multiplying C and F.

If you were to write out
how to multiply these together with individual 8-bit multiplication,
it might look something like this:

                A   B   C   *
                D   E   F
=====================================
                    CFH CFL
                CEH CEL
            CDH CDL
                BFH BFL
            BEH BEL
        BDH BDL
            AFH AFL
        AEH AEL
+   ADH ADL
    5   4   3   2   1   0
=====================================

This result does in fact take up 6 bytes, or 48-bits.
```

```
Something to keep in mind here is that when adding each
individual result to the total (such as CDL) there will
be previous results already added to that byte. That means
that we need to keep in mind carries. There will be no
carry coming out of the last byte, as two 24-bit numbers
multiplied together (FFFFFF x FFFFFF) have a maximum
possible value of FFFFFE000001, which is only 6 bytes!

The way I will handle carries is every time I do addition,
I will check the carry bit to see if it is set. If it is,
I will move up where I am looking at by one byte, then add
the carry. Since this is also addition, I will check the
carry AGAIN! Repeat until no more carries. This means
I could carry up to 4 times, (the lowest byte will never
carry since it is only added to once) so for reliabilities
sake I will use a loop to check/add caries instead of just
adding the carry to every possibly byte even if there is
no carry.

*/
		; Execute the function here
		push XH		; Push literally everything because
		push XL		; I have no idea what I will need :)
		push YH
		push YL
		push ZH
		push ZL
		push mpr
		push rlo
		push rhi
		push A
		push B
		push iloop
		push oloop

		; Load beginning address of MUL24 result to X
		ldi	XL, low(MUL24_Result)
		ldi	XH, high(MUL24_Result)
		st X+, zero
		st X+, zero
		st X+, zero
		st X+, zero
		st X+, zero
		st X, zero	; Clear result just in case!

		; Load beginning address of first operand into Z
```

```
        ldi     ZL, low(MUL24_OP1)   ; Load low byte of address
        ldi     ZH, high(MUL24_OP1)  ; Load high byte of address
        ; Load beginning address of second operand into Y
        ldi     YL, low(MUL24_OP2)   ; Load low byte of address
        ldi     YH, high(MUL24_OP2)  ; Load high byte of address
        ; Load beginning address of result into X
        ldi     XL, low(MUL24_Result) ; points the end of X
        ldi     XH, high(MUL24_Result)
/*
                A   B   C   *       (Z pointer)
                D   E   F           (Y pointer)
                2   1   0           offset
==========================================
                        CF
                    CE
                CD
                    BF
                BE
            BD
                AF
            AE
        AD
    5   4   3   2   1   0   (X offset)
==========================================
*/
        ld A, Z         ; Load C byte of OP1 to A
                        ; A will hold the first op
        ld B, Y         ; Load F byte of OP1 to B
                        ; B will hold the second op
        mul A, B        ; C*F
        rcall ADDMUL2X  ; Add to result
;               A   B   C   *       (Z pointer)
;               D   E   F           (Y pointer)
;               2   1   0           offset
        adiw XH:XL, 1   ; X offset = 1, changed so that ADDMUL2X
                        ; adds to the correct place
                        ; Need to do CE and BF
;               A   B   C   *       (Z pointer)
;               D   E   F           (Y pointer)
;               2   1   0           offset
        ld A, Z         ; A <- C
        ldd B, Y+1      ; B <- E
        mul A, B        ; C*E
        rcall ADDMUL2X  ; Add to result
```

```
        ldd A, Z+1    ; A <- B
        ld B, Y       ; B <- F
        mul A, B      ; B*F
        rcall ADDMUL2X
        adiw XH:XL, 1     ; X offset = 2
                          ; Need to do CD, BE, and AF
;               A   B   C   *       (Z pointer)
;               D   E   F           (Y pointer)
;               2   1   0           offset
        ld A, Z       ; A <- C
        ldd B, Y+2    ; B <- D
        mul A, B      ; C*D
        rcall ADDMUL2X
        ldd A, Z+1    ; A <- B
        ldd B, Y+1    ; B <- E
        mul A, B      ; B*E
        rcall ADDMUL2X
        ldd A, Z+2    ; A <-A
        ld B, Y       ; B <-F
        mul A, B      ; A*F
        rcall ADDMUL2X
        adiw XH:XL, 1     ; X offset = 3
                          ; Need to do BD and AE
;               A   B   C   *       (Z pointer)
;               D   E   F           (Y pointer)
;               2   1   0           offset
        ldd A, Z+1    ; A <- B
        ldd B, Y+2    ; B <- D
        mul A, B      ; B*D
        rcall ADDMUL2X
        ldd A, Z+2    ; A <- A (nice)
        ldd B, Y+1    ; B <- E
        mul A, B      ; A*E
        rcall ADDMUL2X
        adiw XH:XL, 1     ; X offset = 4
                          ; Need to do AD
;               A   B   C   *       (Z pointer)
;               D   E   F           (Y pointer)
;               2   1   0           offset
        ldd A, Z+2    ; A <- A (nice)
        ldd B, Y+2    ; B <- D
        mul A, B      ; A*D
        rcall ADDMUL2X
        ; done!

        pop oloop
```

```
        pop iloop
        pop B
        pop A
        pop rhi
        pop rlo
        pop mpr
        pop ZL
        pop ZH
        pop YL
        pop YH
        pop XL
        pop XH

        ret                         ; End a function with RET

;-----------------------------------------------------------
; Func: ADDMUL2X
; Desc: Adds a partial multiplication result word to X, assuming
;       that X is already pointing to where the low result
;       byte should be placed
;       AKA if result of the multiplication needs to be placed
;       into the X starting at the second byte, then X had
;       better already be pointing to the second byte.
;       Multiplication should already be done and sitting in
;       rlo and rhi!
;-----------------------------------------------------------
ADDMUL2X:
        push XL
        push XH
        push A

        ld A, X         ; Pull out low byte from X to A
        add A, rlo      ; Add rlo to A
        st X+, A        ; Place A back into X, inc X
        ld A, X         ; Pull out high byte from X to A
        adc A, rhi      ; Add rhi to high byte plus carry
        st X+, A        ; Place A back into X, inc X
        ; Carry until no longer carry
addmulloop:
        brcc addmulfinish ; Finish if carry no longer set
        ld A, X         ; Pull out current byte from X
        adc A, zero     ; Add carry to current byte
        st X+, A        ; Place back into X, inc X
        rjmp addmulloop ; Return to begining of loop
addmulfinish:

```

```
        pop A
        pop XH
        pop XL
        ret

;-----------------------------------------------------------
; Func: LOADMUL24
; Desc: Loads the numbers needed for the example MUL24
;-----------------------------------------------------------
LOADMUL24:
        ; Execute the function here
        push YH ; push regs to stack
        push YL
        push ZH
        push ZL
        push mpr
        push iloop
        push oloop

        ; Uses OperandE1, OperandE2, OperandF1, and OperandF2
        ; Placing these into MUL24_OP1 and MUL24_OP2 respectively
        ldi ZH, high(OperandE1) ; load OperandE1 location to Z
        ldi ZL, low(OperandE1)
        ; Shift Z to prepare for program memory access:
        lsl ZH
        lsl ZL
        adc ZH, zero ; shift carry from lower byte to upper byte
        ldi YH, high(MUL24_OP1) ; Load OP1 location into Y
        ldi YL, low(MUL24_OP1)  ; ($0118)

        ldi oloop, 3    ; load oloop with 3 to loop 3 times.
mulloadloop1:
        lpm mpr, Z+ ;load mpr from Z, inc Z
        st Y+, mpr  ; store mpr to Y, inc Y
        dec oloop       ; decrement oloop to run loop 3 times
        brne mulloadloop1
        ; since operand E2 is immediately after E1 in the program data
        ; we should be able to just increment to it :)
        ; Operand E is now loaded to MUL24_OP1

        ldi ZH, high(OperandF1) ; load OperandF1 location to Z
        ldi ZL, low(OperandF1)
        ; Shift Z to prepare for program memory access:
        lsl ZH
        lsl ZL
        adc ZH, zero ; shift carry from lower byte to upper byte
```

```
        ldi YH, high(MUL24_OP2) ; Load OP1 location into Y
        ldi YL, low(MUL24_OP2)  ; ($011B)

        ldi oloop, 3    ; load oloop with 3 to loop 3 times.
mulloadloop2:
        lpm mpr, Z+ ; load mpr from Z, inc Z
        st Y+, mpr  ; store mpr to Y, inc Y
        dec oloop       ; decrement oloop to run loop 3 times
        brne mulloadloop2
        ; Both operands should be loaded into program mem now!

        pop oloop   ; pop regs from stack
        pop iloop
        pop mpr
        pop ZL
        pop ZH
        pop YL
        pop YH

        ret                     ; End a function with RET

;-----------------------------------------------------------
; Func: LOADADD16
; Desc: Loads the numbers needed for the example ADD16
;-----------------------------------------------------------
LOADADD16:
        ; Execute the function here
        push YH ; push regs to stack
        push YL
        push ZH
        push ZL
        push mpr
        push iloop
        push oloop

        ; Uses OperandA and OperandB
        ; Placing these into ADD16_OP1 and ADD16_OP2 respectively
        ldi ZH, high(OperandA)  ; load OperandA location to Z
        ldi ZL, low(OperandA)
        ; Shift Z to prepare for program memory access:
        lsl ZH
        lsl ZL
        adc ZH, zero ; shift carry from lower byte to upper byte
        ldi YH, high(ADD16_OP1) ; Load OP1 location into Y
        ldi YL, low(ADD16_OP1)  ; ($0110)

```

```
        ldi oloop, 2    ; load oloop with 2 to loop 2 times.
addloadloop1:
        lpm mpr, Z+ ; load mpr from Z, inc Z
        st Y+, mpr  ; store mpr to Y, inc Y
        dec oloop       ; decrement oloop to run loop 2 times
        brne addloadloop1
        ; Operand A is now loaded to ADD16_OP1

        ldi ZH, high(OperandB)  ; load OperandB location to Z
        ldi ZL, low(OperandB)
        ; Shift Z to prepare for program memory access:
        lsl ZH
        lsl ZL
        adc ZH, zero ; shift carry from lower byte to upper byte
        ldi YH, high(ADD16_OP2) ; Load OP2 location into Y
        ldi YL, low(ADD16_OP2)  ; ($0112)

        ldi oloop, 2    ; load oloop with 2 to loop 2 times.
addloadloop2:
        lpm mpr, Z+ ; load mpr from Z, inc Z
        st Y+, mpr  ; store mpr to Y, inc Y
        dec oloop       ; decrement oloop to run loop 2 times
        brne addloadloop2
        ; Both operands should be loaded into program mem now!

        pop oloop   ; pop regs from stack
        pop iloop
        pop mpr
        pop ZL
        pop ZH
        pop YL
        pop YH

        ret                     ; End a function with RET

;-----------------------------------------------------------
; Func: LOADSUB16
; Desc: Loads the numbers needed for the example SUB16
;-----------------------------------------------------------
LOADSUB16:
        ; Execute the function here
        push YH ; push regs to stack
        push YL
        push ZH
        push ZL
        push mpr
```

```
        push iloop
        push oloop

        ; Uses OperandC and OperandD
        ; Placing these into SUB16_OP1 and SUB16_OP2 respectively
        ldi ZH, high(OperandC)   ; load OperandC location to Z
        ldi ZL, low(OperandC)
        ; Shift Z to prepare for program memory access:
        lsl ZH
        lsl ZL
        adc ZH, zero ; shift carry from lower byte to upper byte
        ldi YH, high(SUB16_OP1) ; Load OP1 location into Y
        ldi YL, low(SUB16_OP1)   ; ($0114)

        ldi oloop, 2     ; load oloop with 2 to loop 2 times.
subloadloop1:
        lpm mpr, Z+ ; load mpr from Z, inc Z
        st Y+, mpr   ; store mpr to Y, inc Y
        dec oloop        ; decrement oloop to run loop 2 times
        brne subloadloop1
        ; Operand C is now loaded to ADD16_OP1

        ldi ZH, high(OperandD)   ; load OperandD location to Z
        ldi ZL, low(OperandD)
        ; Shift Z to prepare for program memory access:
        lsl ZH
        lsl ZL
        adc ZH, zero ; shift carry from lower byte to upper byte
        ldi YH, high(SUB16_OP2) ; Load OP2 location into Y
        ldi YL, low(SUB16_OP2)   ; ($0116)

        ldi oloop, 2     ; load oloop with 2 to loop 2 times.
subloadloop2:
        lpm mpr, Z+ ; load mpr from Z, inc Z
        st Y+, mpr   ; store mpr to Y, inc Y
        dec oloop        ; decrement oloop to run loop 2 times
        brne subloadloop2
        ; Both operands should be loaded into program mem now!

        pop oloop    ; pop regs from stack
        pop iloop
        pop mpr
        pop ZL
        pop ZH
        pop YL
        pop YH
```

```

        ret                         ; End a function with 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
        ; Already done in LOADCOMPOUND
        ; Perform subtraction to calculate G - H:
        rcall SUB16
        ; Setup the ADD16 function with SUB16 result and operand I
        ; Operand I already loaded to ADD16_OP2 by LOADCOMPOUND
        ; Just load SUB16_Result into ADD16_OP1:
        ldi XL, low(SUB16_Result)
        ldi XH, high(SUB16_Result)
        ldi YL, low(ADD16_OP1)
        ldi YH, high(ADD16_OP1)
        ld mpr, X+
        st Y+, mpr
        ld mpr, X
        st Y, mpr
        ; Perform addition next to calculate (G - H) + I:
        rcall ADD16
        ; Setup the MUL24 function with ADD16 result as both operands:
        ldi XL, low(ADD16_Result)
        ldi XH, high(ADD16_Result)
        ldi YL, low(MUL24_OP1)
        ldi YH, high(MUL24_OP1)
        ldi ZL, low(MUL24_OP2)
        ldi ZH, high(MUL24_OP2)
        ld mpr, X+
        st Y+, mpr
        st Z+, mpr
        ld mpr, X+
        st Y+, mpr
        st Z+, mpr
        ld mpr, X
```

```
        st Y, mpr
        st Z, mpr
        ; Perform multiplication to calculate ((G - H) + I)^2:
        rcall MUL24
        ret                     ; End a function with RET
;-----------------------------------------------------------
; Func: LOADCOMPOUND
; Desc: Loads the numbers needed for the compound, as well
;       as clearing the result locations from previous
;       functions first.
;-----------------------------------------------------------
LOADCOMPOUND:
        ; Execute the function here
        push YH ; push regs to stack
        push YL
        push ZH
        push ZL
        push mpr
        push iloop
        push oloop

        rcall CLRRES ; Clear result memory locations

        ; Uses OperandG, OperandH, and OperandI
        ; as ( ( G - H ) + I )^2
        ; Meaning SUB16 with G and H
        ; Then ADD16 with the result and I
        ; Then MUL24 where both operands are the result
        ; So G and H need to be loaded to
        ; SUB16_OP1 and SUB16_OP2 respectively
        ; "I" will be loaded to ADD16_OP2 due to where
        ; it visually is in the equation,
        ; although it doesn't matter too much

        ldi ZH, high(OperandG)   ; load OperandG location to Z
        ldi ZL, low(OperandG)
        ; Shift Z to prepare for program memory access:
        lsl ZH
        lsl ZL
        adc ZH, zero ; shift carry from lower byte to upper byte
        ldi YH, high(SUB16_OP1) ; Load OP1 location into Y
        ldi YL, low(SUB16_OP1)   ; ($0114)

        ldi oloop, 2     ; load oloop with 2 to loop 2 times.
comploadloop1:
        lpm mpr, Z+ ; load mpr from Z, inc Z
```

```
        st Y+, mpr  ; store mpr to Y, inc Y
        dec oloop       ; decrement oloop to run loop 2 times
        brne comploadloop1
        ; Operand G is now loaded to SUB16_OP1

        ldi ZH, high(OperandH)  ; load OperandD location to Z
        ldi ZL, low(OperandH)
        ; Shift Z to prepare for program memory access:
        lsl ZH
        lsl ZL
        adc ZH, zero ; shift carry from lower byte to upper byte
        ldi YH, high(SUB16_OP2) ; Load OP2 location into Y
        ldi YL, low(SUB16_OP2)  ; ($0116)

        ldi oloop, 2    ; load oloop with 2 to loop 2 times.
comploadloop2:
        lpm mpr, Z+ ; load mpr from Z, inc Z
        st Y+, mpr  ; store mpr to Y, inc Y
        dec oloop       ; decrement oloop to run loop 2 times
        brne comploadloop2
        ; Operand H now loaded to SUB16_OP2

        ldi ZH, high(OperandI)  ; load OperandD location to Z
        ldi ZL, low(OperandI)
        ; Shift Z to prepare for program memory access:
        lsl ZH
        lsl ZL
        adc ZH, zero ; shift carry from lower byte to upper byte
        ldi YH, high(ADD16_OP2) ; Load OP2 location into Y
        ldi YL, low(ADD16_OP2)  ; ($0112)

        ldi oloop, 2    ; load oloop with 2 to loop 2 times.
comploadloop3:
        lpm mpr, Z+ ; load mpr from Z, inc Z
        st Y+, mpr  ; store mpr to Y, inc Y
        dec oloop       ; decrement oloop to run loop 2 times
        brne comploadloop3
        ; Operand I now loaded to ADD16_OP2

        pop oloop   ; pop regs from stack
        pop iloop
        pop mpr
        pop ZL
        pop ZH
        pop YL
        pop YH
```

```

        ret                         ; End a function with RET

;-----------------------------------------------------------
; Func: CLRRES
; Desc: Clears the memory locations of the results for
;       ADD16, SUB16, and MUL24
;-----------------------------------------------------------
CLRRES:
        push XL
        push XH
        push YL
        push YH
        push ZL
        push ZH

        ; Load beginning address of ADD16 result to Z
        ldi     XL, low(ADD16_Result)   ; Load low byte of address
        ldi     XH, high(ADD16_Result)  ; Load high byte of address
        ; Load beginning address of SUB16 result to Y
        ldi     YL, low(SUB16_Result)   ; Load low byte of address
        ldi     YH, high(SUB16_Result)  ; Load high byte of address
        ; Load beginning address of MUL24 result to Z
        ldi     ZL, low(MUL24_Result)
        ldi     ZH, high(MUL24_Result)

        ; Write zeros to all result locations
        st X+, zero
        st X+, zero
        st X, zero  ; Three bytes for ADD16 result
        st Y+, zero
        st Y, zero  ; Two for SUB16
        st Z+, zero
        st Z+, zero
        st Z+, zero
        st Z+, zero
        st Z+, zero
        st Z, zero  ; And SIX for MUL24

        pop ZH
        pop ZL
        pop YH
        pop YL
        pop XH
        pop XL

```

```
        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

            ; Restore all registers in reverves order
            pop     iloop
            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
        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:   ;$0110
         .byte 2      ; allocate two bytes for first operand of ADD16
ADD16_OP2:   ;$0112
         .byte 2      ; allocate two bytes for second operand of ADD16
SUB16_OP1:   ;$0114
         .byte 2      ; allocate two bytes for first operand of SUB16
SUB16_OP2:   ;$0116
         .byte 2      ; allocate two bytes for second operand of SUB16
MUL24_OP1:   ;$0118
         .byte 3      ; allocate three bytes for first operand of MUL24
MUL24_OP2:   ;$011B
         .byte 3      ; allocate three bytes for second operand of MUL24


.org     $0120        ; data memory allocation for results
ADD16_Result:
         .byte 3      ; allocate three bytes for ADD16 result
.org     $0130
SUB16_Result:
         .byte 2      ; allocate two bytes for SUB16 result
.org     $0140
MUL24_Result:
         .byte 6      ; allocate six bytes for MUL24 result

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