ECE 375 Lab 5

Large Number Arithmetic

**Lab Time: Tuesday 8-10**

Zachary DeVita

# Introduction

The purpose of this lab is to become familiarized with arithmetic using large number. In cases, where the arithmetic operands are larger than the bits that the microcontroller can process in a single cycle, there needs to be a process for manipulating registers to produce the correct result. Arithmetic of numbers which are too large to be represented with only 8 bits requires a more complex system of temporarily storing values in multiple registers.

Using the example code as well as the software tools provided, we were able to write an assembly language program to conduct an addition, subtraction, and multiplication of two 16 bit values. After the program was written, it was converted to hex using AVRStudio, and then uploaded to the AVR board using the Universal Programmer.

# Program Overview

The lab has been broken up into two separate programs which are responsible for different requirements for the lab. Both programs contain functions for 16 bit addition, 16 bit subtraction, and the 24 bit multiplication. One program is only used for the test cases which were given in the main of the program template. The other program is used to implement the combination function of the program.

The programs essentially operate in the same fashion. Hex values are stored into the data memory of ATmega128 microcontroller to be used as operands for the arithmetic. Depending on whether the addition, subtraction, or multiplication function is used to manipulate the data will determine the result. The result from the function is also saved into data memory at different locations as the operands in order to preserve the data. The latter version of the program has the combination function which utilizes all three of the aforementioned functions to find the result of ((A-B)+C)2, where the A, B, & C are different 16 bit operands.

## Preprocessor Routine

Prior to entering main(), the definition file for the ATmega128 is included. There are many registers defined for various purposes as well. These include the multipurpose register, a zero register, several registers to hold custom flags, inner and outer loop counters, and a register to hold the carry value associated with the shift of an 8 bit register holding the first part of a 16 bit operand.

Also before main(), but after the preprocessor setup is the start of code segment commands, and the program initialization. The stack pointer is also initialized, and the zero register is cleared.

## Main Routine

Once inside of main() there are the test cases. This is where operands are stored into data memory, and where arithmetic functions are called to manipulate the data. There is one test case for each individual function here, and the results are then listed and checked for accuracy.

## ADD16

The ADD16 subroutine will take two 16 bit operands and add them together resulting in a 24 bit result. Initially in the function, register values are pushed onto the stack to preserve their initial values. The X and Y registers are used here to point at the individual operands, while the Z register is used to point to the specific memory address where the result will be stored. The appropriate locations in memory for these registers to point to is set up at the start of the function. The low byte of each operand is then loaded into registers A and B, and then these values are added together with their result stored in data memory. The high byte of each operand is then loaded into the same registers, and an addition with carry is done. This result is stored next to the previous result. After that, the function has completed and the original register values are popped off the stack.

## SUB16

The SUB16 subroutine will take two 16 bit operands and find their difference which results in a 16 bit value. Initially in the function, register values are pushed onto the stack to preserve their initial values. The X and Y registers are used here to point at the individual operands, while the Z register is used to point to the specific memory address where the result will be stored. The appropriate locations in memory for these registers to point to is set up at the start of the function. The low byte of each operand is then loaded into registers A and B, and then these values are subtracted with the result stored in data memory. The high byte of each operand is then loaded into the same registers, and a subtraction with carry is done. This result is stored next to the previous result. After that, the function has completed and the original register values are popped off the stack.

## MUL24

The MUL24 subroutine will take two 24 bit operands and find their product which results in a 48 bit value. Initially in the function, register values are pushed onto the stack to preserve their initial values. For this function we load the multiplicand operand into registers to be used as a constant for the subroutine. Two sets of three consecutive registers are then used to hold the result from the function. The first set is initially set to zero, and the second set initially holds the multiplier operand for the function. For each of the iterations of the loop, the values held in the six registers shift to the right, acting as though they were a single register. At the end of the shift the carry flag is checked, and, if it is present, then the multiplicand is summed with the upper 16 bits of the result registers. The number of iterations of the loop is equal to the number of bits being multiplied, so, after 24 iterations, the result is produced. The result is stored into data memory when the subroutine has finished.

## COMBO

The COMBO subroutine is set up to simply call the three aforementioned subroutines for a set of three operands. The second operand is subtracted from the first operand using the SUB16 subroutine, and the result is stored into data memory. The third operand is summed with that result using the ADD16 subroutine. The result from that is then stored into a different location in order to retain the original results. The result from the addition is then multiplied by itself using the MUL24 function, and this final 48 bit result is then stored in an additional location in data memory.

# Additional Questions

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

The ‘V’ flag is the two’s complement overflow indicator, and it is used when the contents of the register after an operation results in $80. The flag indicates that there was an arithmetic overflow in an operation, and that the signed two’s complement result would not fit in the number of bits allocated for the register. An example of this could be:

127+120 which equals 247

247 is represented in binary as 11110111

In two’s complement this binary is equivalent to -9

Because this value is too large to be represented in 8 bit two’s complement, the two’s complement overflow flag is set.

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

The BYTE directive reserves memory resources in the SRAM or EEPROM, and can be referred to by its label where it’s instantiated. The directive takes one parameter, which is the number of bytes to reserve. The directive can only be used within a Data Segment. This is essentially the assembly language equivalent of a variable with a predefined size, and can be used like a variable by using its label as the reference. Variables are defined with the BYTE directive, and constants are declared with a EQU directive.

# Conclusion

The requirements for this lab were to write an AVR assembly program which could do large number arithmetic. To be more precise, The program’s objective was to conduct 16 bit addition, 16 bit subtraction, 24 bit multiplication, and another subroutine to conduct a combination of the three aforementioned subroutines. The program was an overall success. Utilizing the simple skeleton program, we were able to make all functions operate correctly. The lab was extraordinarily difficult, but may have helped me to learn the language more effectively than previous labs. Hopefully I can take away from this lab what I needed to, and maybe future labs will be more enjoyable with the knowledge I gained from this lab.

# Source Code

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;\*

;\* Large Number Arithmetic

;\*

;\* This is the skeleton file for Lab 5 of ECE 375

;\*

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;\*

;\* Author: Zachary DeVita

;\* Date: 10/25/2016

;\*

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

.include "m128def.inc" ; Include definition file

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;\* Internal Register Definitions and Constants

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

.def mpr = r16 ; Multipurpose register

.def rlo = r7 ; Low byte of MUL result

.def rhi = r8 ; High byte of MUL result

.def zero = r2 ; Zero register, set to zero in INIT, useful for calculations

.def A = r0 ; A variable

.def B = r1 ; Another variable

.def flag1 = r19

.def flag2 = r20

.def flag3 = r3

.def carry\_reg = r23

.def WL = r25

.def WH = r24

.def oloop = r17 ; Outer Loop Counter

.def iloop = r18 ; Inner Loop Counter

.def i = r22

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;\* Start of Code Segment

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

.cseg ; Beginning of code segment

;-----------------------------------------------------------

; Interrupt Vectors

;-----------------------------------------------------------

.org $0000 ; Beginning of IVs

rjmp INIT ; Reset interrupt

.org $0046 ; End of Interrupt Vectors

;-----------------------------------------------------------

; Program Initialization

;-----------------------------------------------------------

INIT: ; The initialization routine

; Initialize Stack Pointer

ldi mpr, low(RAMEND)

out SPL, mpr ; Load SPL with low byte of RAMEND

ldi mpr, high(RAMEND)

out SPH, mpr ; Load SPH with high byte of RAMEND

clr zero ; Set the zero register to zero, maintain

; these semantics, meaning, don't load anything

; to it.

;-----------------------------------------------------------

; Main Program

;-----------------------------------------------------------

MAIN: ; The Main program

; Setup the ADD16 function direct test

ldi XL, 0x00

ldi XH, 0x01

; Enter values 0xA2FF and 0xF477 into data memory

; locations where ADD16 will get its inputs from

ldi mpr, 0xA2 ; Hi

st X+, mpr ; store contents of X register to data memory

ldi mpr, 0xFF ; Lo

st X+, mpr

ldi mpr, 0xF4 ; Hi

st X+, mpr

ldi mpr, 0x77 ; Lo

st X+, mpr ; store contents of Y register to data memory

rcall ADD16 ; Call ADD16 function to test its correctness

; (calculate A2FF + F477)

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

; Answer is 0x19776

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

; Observe result in Memory window

; Setup the SUB16 function direct test

adiw XH:XL, 3

; Enter values 0xF08A and 0x4BCD into data memory

; locations where SUB16 will get its inputs from

ldi mpr, 0xF0 ; Hi

st X+, mpr ; store contents of X register to data memory\

ldi mpr, 0x8A ; Lo

st X+, mpr

ldi mpr, 0x4B ; Hi

st X+, mpr

ldi mpr, 0xCD ; Lo

st X+, mpr ; store contents of Y register to data memory

rcall SUB16 ; Call SUB16 function to test its correctness

; (calculate F08A - 4BCD)

; Observe result in Memory window

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

; Answer is 0xA4BD

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

; Setup the MUL24 function direct test

adiw XH:XL, 3

; Enter values 0xFFFF and 0xFFFF into data memory

; locations where SUB16 will get its inputs from

ldi mpr, 0xFF ; Hi

st X+, mpr ; store contents of X register to data memory\

ldi mpr, 0xFF ; Lo

st X+, mpr

ldi mpr, 0xFF ; Hi

st X+, mpr

ldi mpr, 0xFF ; Lo

st X+, mpr ; store contents of Y register to data memory

rcall MUL24 ; Call MUL24 function to test its correctness

; (calculate FFFFFF \* FFFFFF)

; Observe result in Memory window

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

; Answer is 0xFFFE0001

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;\* 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:

; Save variable by pushing them to the stack

push A

push B

push XL

push XH

push YL

push YH

push ZL

push ZH

clc

sbiw XH:XL, 2

; Execute the function here

movw YH:YL, XH:XL

adiw YH:YL, 2

movw ZH:ZL, YH:YL

adiw ZH:ZL, 2

ld A, -X

ld B, -Y

add B, A

st Z, B

ld A, -X

ld B, -Y

adc B, A

st -Z, B

brcc EXIT1

st -Z, XH

EXIT1:

; Restore variable by popping them from the stack in reverse order

pop ZH

pop ZL

pop YH

pop YL

pop XH

pop XL

pop B

pop A

ret ; End a function with RET

;-----------------------------------------------------------

; Func: SUB16

; Desc: Subtracts two 16-bit numbers and generates a 16-bit

; result.

;-----------------------------------------------------------

SUB16:

; Save variable by pushing them to the stack

push A

push B

push XL

push XH

push YL

push YH

push ZL

push ZH

clc

sbiw XH:XL, 2

; Execute the function here

movw YH:YL, XH:XL

adiw YH:YL, 2

movw ZH:ZL, YH:YL

adiw ZH:ZL, 2

ld A, -X

ld B, -Y

sub A, B

st Z, A

ld A, -X

ld B, -Y

sbc A, B

st -Z, A

brcc EXIT2

adiw ZH:ZL, 2

st Z, XH

EXIT2:

; Restore variable by popping them from the stack in reverse order

pop ZH

pop ZL

pop YH

pop YL

pop XH

pop XL

pop B

pop A

ret ; End a function with RET

;-----------------------------------------------------------

; Func: MUL24

; Desc: Multiplies two 24-bit numbers and generates a 48-bit

; result.

;-----------------------------------------------------------

MUL24:

; Save variable by pushing them to the stack

push XL

push XH

push YL

push YH

push ZL

push ZH

sbiw XH:XL, 4

; Execute the function here

ld WL, X+ ; multiplicand

ld WH, X+ ; multiplicand

ld ZL, X+ ; multiplier

ld ZH, X+ ; multiplier

ldi carry\_reg, 0x80

ldi i, 17

clr flag3

clr YL

clr YH

clc

LOOP24:

clr flag1

clr flag2

lsr YL

brcc NOCARRY0

inc flag1

NOCARRY0:

cp flag3, zero

breq NOCARRY1

add YL, carry\_reg

clr flag3

NOCARRY1:

lsr YH

brcc NOCARRY2

inc flag2

NOCARRY2:

cpi flag1, 0

breq NOCARRY3

add YH, carry\_reg

clr flag1

NOCARRY3:

lsr ZL

brcc NOCARRY4

inc flag1

NOCARRY4:

cpi flag2, 0

breq NOCARRY5

add ZL, carry\_reg

clr flag2

NOCARRY5:

lsr ZH

brcc NOCARRY6

inc flag2

NOCARRY6:

cpi flag1, 0

breq NOCARRY7

add ZH, carry\_reg

NOCARRY7:

dec i

cpi i, 0

breq EXIT3

cpi flag2, 0

breq NOCARRY8

add YH, WH

adc YL, WL

brcc LOOP24

inc flag3

NOCARRY8:

rjmp LOOP24

EXIT3:

; Restore variable by popping them from the stack in reverse order

st X+, YL

st X+, YH

st X+, ZL

st X+, ZH

pop ZH

pop ZL

pop YH

pop YL

pop XH

pop XL

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, B, 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:

; Save variable by pushing them to the stack

push A

push B

push XL

push XH

push YL

push YH

push ZL

push ZH

; Execute the function here

rcall SUB16

rcall ADD16

rcall MUL24

; Restore variable by popping them from the stack in reverse order

pop ZH

pop ZL

pop YH

pop YL

pop XH

pop XL

pop B

pop A

ret ; End a function with RET