# ECE 375 Lab 4

Data Manipulation and the LCD Display

Lab Time: Tuesday 10-12

Miles Van de Wetering
Mathew Popowski

TA Signature

## 1 Introduction

In this lab, we learned to work with multiple registers at once in order to perform arithmetic operations on numbers that were larger than 8 bytes. Specifically, we performed 16 byte addition and subtraction, and 24 byte multiplication (we only did the extra-credit portion, since it was described as being eligible for full credit).

## 2 Internal Register Definitions and Constants

We used r16 for a multipurpose register, r0, 1, 2, 25 to store the high bytes multiplication result (the B operand registers were used for the low bytes). r19, 20, and 23 were used to store the 2-3 bytes used for one arithmetic operand (A). r21, 22, and 25 were used to store the 2-3 bytes used for the second operand (B). r15 was used to represent the case when addition spilled over into another register. AddParam1Loc ($0101) is the address in data memory where we stored the A operand for addition, AddParam2Loc was where we stored the second operand, and so forth. Each address left space for the appropriate number of bytes for higher-size arithmetic.

## 3 Program Initialization

To initialize our program we directed the stack pointer to the end of the program memory section, allowing us to push and pop from the stack as we change scopes (we did very little of this since storing all parameters and results in different registers made demo-ing to our TA much more straightforward). In a real application, this would allow us to reuse registers without concern for scope.

## 4 Main Program

In our main program, we retrieved the A and B operands repeatedly from the appropriate sections of data memory, performed the math operations via function calls, and stored the results in the appropriate sections of data memory. Really, our main function was a very straightforward series of load -¿ call -¿ store operations.

## 5 Add16

In this function, we added first the two low bytes of the operands together, then performed an add-with-carry with the upper bytes, and finally stored the addover if necessary.

## 6 Sub16

This function worked just like adding, except there was no possibility of a carryover (and it used subtraction instead of addition).

## 7 Mul24

This function was significantly more in-depth than the other two, and required some concentration to grasp in it's entirety. Essentially, the first operand is stored in the lower 3 result registers, then repeatedly right-shifted. Each time it's right-shifted, if a 1 was shifted off (if the carry was set) then we perform an add to the upper 3 result registers with the second operand. This is repeated 24 times (24 byte multiplication = 24 repetitions).

## 8 Stored Program Data

We stored OperandD, E, and F for use in the COMPOUND function.

## 9 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 an overflow flag. For example, adding FF to FF will cause an overflow if working with unsigned arithmetic (the result cannot be represented in 8 bits).

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?

   This allows us to reference these data by name, and to reserve multiple bytes at a time.

## 10 Difficulties

We didn't have any major difficulties with this lab.

## 11 Conclusion

All in all, we learned a lot from this lab. Once again we got to make use of functions to create nice, re-useable (or it would have been if we added push/pops) code. Furthermore, we got practice reading/writing bits in the 'assembly' sort of order - lowest digits in the lowest bytes - which is interesting and not always easy to reason about.

## 12 Source Code

Base Code:

```
;***********************************************************
;*
;* Lab5
;*
;* Enter the description of the program here
;*
;* This is the skeleton file for Lab 5 of ECE 375
;*
;***********************************************************
;*
;*  Author: Mathew
;*    Date: 10/25/16
;*
;***********************************************************

.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 ALow = r19 ; A variable
.def AHi = r20 ; Another variable
.def BLow = r21
.def BHi = r22
.def ATop=r23
.def BTop=r24
.def RTop=r25
.def addover = r15


.equ AddParam1Loc = $0101
.equ AddParam2Loc = $0103

.equ SubParam1Loc = $0105
.equ SubParam2Loc = $0107

.equ MultParam1Loc = $0109
.equ MultParam2Loc = $010C

.equ AddResult = $0110
.equ SubResult = $0112
.equ MultResult = $0114
```

```
.equ CmpdResult = $011A
;***********************************************************
;* 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
LDI mpr, LOW(RAMEND)
OUT SPL, mpr
LDI mpr, HIGH(RAMEND)
OUT SPH, mpr ; Init the 2 stack pointer registers

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
LDS ALow, AddParam1Loc
LDS AHi, AddParam1Loc+1

LDS BLow, AddParam2Loc
LDS BHi, AddParam2Loc+1

rcall ADD16

STS AddResult, ALow
STS AddResult+1, AHi
STS AddResult+2, addover

; Enter values 0xA2FF and 0xF477 into data memory
; locations where ADD16 will get its inputs from
; Call ADD16 function to test its correctness
; (calculate A2FF + F477)

; Observe result in Memory window

; Setup the SUB16 function direct test
LDS ALow, SubParam1Loc
LDS AHi, SubParam1Loc+1

LDS BLow, SubParam2Loc
LDS BHi, SubParam2Loc+1

rcall SUB16

STS SubResult, ALow
STS SubResult+1, AHi
; Enter values 0xF08A and 0x4BCD into data memory
; locations where SUB16 will get its inputs from

; Call SUB16 function to test its correctness
; (calculate F08A - 4BCD)

; Observe result in Memory window

; Setup the MUL24 function direct test
LDS ALow, MultParam1Loc
LDS AHi, MultParam1Loc+1
LDS ATop, MultParam1Loc+2

LDS BLow, MultParam2Loc
LDS BHi, MultParam2Loc+1
LDS BTop, MultParam2Loc+2

rcall MUL24

STS MultResult, BLow
STS MultResult+1, BHi
STS MultResult+2, BTop
STS MultResult+3, RLo
STS MultResult+4, RHi
STS MultResult+5, RTop

; Enter values 0xFFFFFF and 0xFFFFFF into data memory
; locations where MUL24 will get its inputs from
; Call MUL24 function to test its correctness
; (calculate FFFFFF * FFFFFF)

; Observe result in Memory window

; Call the COMPOUND function
rcall COMPOUND
STS CmpdResult, BLow
STS CmpdResult+1, BHi
STS CmpdResult+2, BTop
STS CmpdResult+3, RLo
STS CmpdResult+4, RHi
STS CmpdResult+5, RTop
; 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:
; Save variable by pushing them to the stack
; Execute the function here
ADD ALow, BLow
ADC AHi, BHi
ADC addover, zero

; Restore variable by popping them from the stack in reverse order
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
; Execute the function here
```

```
SUB ALow, BLow
SBC AHi, BHi
; Restore variable by popping them from the stack in reverse order
ret ; End a function with RET



;-----------------------------------------------------------
; Func: MUL24
; Desc: Multiplies two 24-bit numbers and generates a 48-bit
; result.
;-----------------------------------------------------------
MUL24:
clc
ldi mpr, 25
Mult: BRCS AddFirst
Rejoin: ror rTop
ror rhi
ror rlo
ror BTop
ror BHi
ror BLow
dec mpr
brne Mult
rjmp MULEND

AddFirst: ADD rlo, ALow
ADC rhi, AHi
ADC rTop, ATop
rjmp Rejoin

MULEND: 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:
LDI ALow, low(OperandD)
LDI AHi, high(OperandD)
LDI BLow, low(OperandE)
LDI BHi, high(OperandE)
rcall ADD16
LDI BLow, low(OperandF)
LDI BHi, high(OperandF)
rcall SUB16
mov BLow, ALow
mov BHi, AHi
mov AddOver, BTop
rcall MUL24
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.

;***********************************************************
;* Stored Program Data
;***********************************************************

; Enter any stored data you might need here

OperandD:
.DB 0x51, 0xFD ; test value for operand D
OperandE:
.DB 0xFF, 0x1E ; test value for operand E
OperandF:
.DB 0xFF, 0xFF ; test value for operand F

.dseg
.org $0100
addrA: .byte 2
addrB: .byte 2
LAddrP: .byte 4

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