ECE 375 Lab 5

Data Manipulation & the LCD

Lab Time: Wednesday 10a-12n

Robert Detjens

David Headrick

TA Signature

Introduction

In this lab, we implemented three large number functions for the ATMega128: ADD16, SUB16, and MUL26. the AVR instruction set can only work with 8 bits at a time, so working with multi-byte values needs to be done in 1-byte chunks. This means the functions written in this lab have to coordinate multiple 8 bit instructions to operate on 16 or 24 bits at a time.

Program Overview

Within the ADD16 function, data is loaded in from program memory. Next, the lower 8 bits are added using ADD. Then, the high byte is added with ADC, which will also bring in the carry bit from the first add. Finally, the high and low results of these two operations is stored into the result in data memory.

The SUB16 function works very much the same as the ADD16 function, except using SUB and SBC for subtracting the 8 bit high and low values.

The MUL24 function uses the shift-and-add challenge algorithm to perform a multiplication on two 24 bit numbers. The result of this function was stored in a 42 bit space in data memory.

Additional Questions

- Although this lab dealt with unsigned numbers, the ATmega128 microcontroller has features 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 an indication that there was a two's complement overflow. For example, if two positive, signed numbers were added together and the result were negative, then the V flag in the status register would become set due to the two's complement overflow during addition.

  This 8 bit signed binary addition example would set the V flag:

  ```
    0b 01100000
  + 0b 01000000
  = 0b 10100000
  ```

- 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?

  One advantage to using the `.BYTE` directive vs. many `.EQU` directives is that human error can interfere with correct address reservation. If you accidentally declare the wrong address, it could write over the data of the previous operand. With the `.ORG` and `.BYTE` directives, the assembler automatically declares the correct addresses for you, given the number of bytes you need. This reduces the risk of error.

Difficulties

One difficulty in this lab was organizing the memory for MUL24. Many different locations in memory were involved, so we had to sit down and write out a plan for where it all was going to end up.

We also were not clearing the addition overflow byte when there was no overflow, so any calculations after one that overflowed into the third byte would have that bit set regardless if the current calculation overflowed or not.

Conclusion

In conclusion, we have learned the basics of large number operations on the ATMega128 by writing multiple large number functions in assembly. In addition, we also learned how to better manage our memory to efficiently store and compute large numbers.

Source Code

```
;***********************************************************
;*
;*  Robert_Detjens_and_David_Headrick_Lab5_sourcecode.asm
;*
;*
;*
;*  This is the skeleton file for Lab 5 of ECE 375
;*
;***********************************************************
;*
;*   Author: Robert Detjens
;*   David Headrick
;*     Date: 10/27/21
;*
;***********************************************************

.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

;-----------------------------------------------------------
; Interrupt Vectors
;-----------------------------------------------------------
.org    $0000
  rjmp  INIT    ; Reset interrupt

.org    $0046

;-----------------------------------------------------------
; Program Initialization
;-----------------------------------------------------------
INIT:
    ; Initialize the Stack Pointer
    ldi         mpr,low(RAMEND)
    out         SPL, mpr
    ldi         mpr,high(RAMEND)
    out         SPH, mpr

  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)
; source addr in prog memory
ldi   ZL,   low(ADD16_D1 << 1)
ldi   ZH,   high(ADD16_D1 << 1)
; store in data memory
ldi   YL,   low(ADD16_OP1)
ldi   YH,   high(ADD16_OP1)
; store two bytes
lpm     mpr,  Z+
st      Y+,   mpr
lpm     mpr,  Z+
st      Y+,   mpr
; store the second operand (directly contiguous)
lpm     mpr,  Z+
st      Y+,   mpr
lpm     mpr,  Z+
st      Y+,   mpr

nop             ; Check load ADD16 operands (Set Break point here #1)
rcall   ADD16   ; (calculate FCBA + FFFF)
nop             ; Check ADD16 result (Set Break point here #2)

; Setup the SUB16 function direct test
; source addr in prog memory
ldi   ZL,   low(SUB16_D1 << 1)
ldi   ZH,   high(SUB16_D1 << 1)
; store in data memory
ldi   YL,   low(SUB16_OP1)
ldi   YH,   high(SUB16_OP1)
; store two bytes
lpm     mpr,  Z+
st      Y+,   mpr
lpm     mpr,  Z+
st      Y+,   mpr
; store two more bytes for the second operand (directly contiguous)
lpm     mpr,  Z+
st      Y+,   mpr
lpm     mpr,  Z+
st      Y+,   mpr

nop             ; Check load SUB16 operands (Set Break point here #3)
rcall   SUB16   ; (calculate FCB9 - E420)
nop             ; Check SUB16 result (Set Break point here #4)
```

```
  ; Setup the MUL24 function direct test

  ; source addr in prog memory
  ldi   ZL,   low(MUL24_D1 << 1)
  ldi   ZH,   high(MUL24_D1 << 1)
  ; store in data memory
  ldi   YL,   low(MUL24_OP1)
  ldi   YH,   high(MUL24_OP1)
  ; store three bytes
  lpm     mpr,  Z+
  st      Y+,   mpr
  lpm     mpr,  Z+
  st      Y+,   mpr
  lpm     mpr,  Z+
  st      Y+,   mpr
  ; discard padding byte
  lpm     mpr,  Z+
  ; load addr of dest (contiguous in program memory)
  ldi   YL,   low(MUL24_OP2)
  ldi   YH,   high(MUL24_OP2)
  ; store second operand
  lpm     mpr,  Z+
  st      Y+,   mpr
  lpm     mpr,  Z+
  st      Y+,   mpr
  lpm     mpr,  Z+
  st      Y+,   mpr

  nop             ; Check load MUL24 operands (Set Break point here #5)
  rcall   MUL24   ; (calculate FFFFFF * FFFFFF)
  nop             ; Check MUL24 result (Set Break point here #6)

  nop             ; Check load COMPOUND operands (Set Break point here)
  rcall   COMPOUND; (calculate )
  nop             ; Check COMPOUND result (Set Break point here)

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  A     ; Save A register
  push  B     ; Save B register
  push  mpr
  push  XH     ; Save X-ptr
  push  XL
  push  YH     ; Save Y-ptr
  push  YL
  push  ZH     ; Save Z-ptr
  push  ZL

  ; Load beginning address of first operand into X
  ldi  XL, low(ADD16_OP1)
  ldi  XH, high(ADD16_OP1)

  ; Load beginning address of second operand into Y
  ldi  YL, low(ADD16_OP2)
  ldi  YH, high(ADD16_OP2)

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


  ; add low bytes
  ld    A,    X+
  ld    B,    Y+
  add   A,    B
  st    Z+,   A
  ; add high bytes with carry
  ld    A,    X+
  ld    B,    Y+
  adc   A,    B
  st    Z+,   A
  ; store extra bit if carry
  brcc  ADD_nocarry
  ldi   mpr,  1
  st    Z,    mpr  ; if carry, set overflow in next byte
```

```
  jmp   ADD_end
  ADD_nocarry:
  ldi   mpr,  0
  st    Z,    mpr  ; if not carry, clear overflow in next byte
  ADD_end:

  pop  ZL
  pop  ZH
  pop  YL
  pop  YH
  pop  XL
  pop  XH
  pop  mpr
  pop  B
  pop  A

  ret

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

  push  A    ; Save A register
  push  B    ; Save B register
  push  mpr
  push  XH    ; Save X-ptr
  push  XL
  push  YH    ; Save Y-ptr
  push  YL
  push  ZH    ; Save Z-ptr
  push  ZL

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

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

  ; Load beginning address of result into Z
```

```
  ldi  ZL, low(SUB16_Result)
  ldi  ZH, high(SUB16_Result)

  ; Execute the function

  ; subract low bytes
  ld    A,    X+
  ld    B,    Y+
  sub   A,    B
  st    Z+,   A
  ; subtract high bytes with carry borrow
  ld    A,    X+
  ld    B,    Y+
  sbc   A,    B
  st    Z+,   A

  pop  ZL
  pop  ZH
  pop  YL
  pop  YH
  pop  XL
  pop  XH
  pop  mpr
  pop  B
  pop  A

  ret


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

  ; make sure result is zero'd before doing computation
  ldi  ZL, low(MUL24_Result)
  ldi  ZH, high(MUL24_Result)
  ldi   OLoop,  6
  MUL24_init_result:
    clr     mpr
    st      Z+,   mpr
```

```
  dec     OLoop
  brne    MUL24_init_result

; make sure op1 shiftspace is also zero'd
ldi  XL, low(MUL24_OP1) + 3
ldi  XH, high(MUL24_OP1)
ldi   OLoop,  3
MUL24_init_shift:
  clr     mpr
  st      X+,   mpr
  dec     OLoop
  brne    MUL24_init_shift

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

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

; Load beginning address of result into Z
ldi  ZL, low(MUL24_Result)
ldi  ZH, high(MUL24_Result)


ldi   OLoop,  24
MUL24_loop:
  ; shift current LSB out of op2
  rcall   LSR24

  ; add op1 to total if that shifted bit is 1
  brcc    MUL24_noadd
    rcall   ADD48
  MUL24_noadd:

  ; shift op1 left
  rcall   LSL48

  ; do this 24 times, one for each bit on op2
  dec     OLoop
  brne    MUL24_loop

ret
```

```
;-----------------------------------------------------------
; Func: LSR24
; Desc: Shifts one 24-bit value in memory right one bit.
;-----------------------------------------------------------
LSR24:
  ; Execute the function here


  push OLoop
  push mpr
  push  XH    ; Save X-ptr
  push  XL

  ; Load address of highest bute of operand into X
  ldi  XL, low(MUL24_OP2) + 3
  ldi  XH, high(MUL24_OP2)

  clc
  ldi   OLoop,  3
  LSR24_loop:
    ; shift three bytes (using rotate to propagate carry)
    ld    mpr,  -X
    ror   mpr
    st    X,   mpr
    dec   OLoop
    brne  LSR24_loop

  pop  XL
  pop  XH
  pop mpr
  pop OLoop

  ret



;-----------------------------------------------------------
; Func: LSL48
; Desc: Shifts one 24-bit value in memory right one bit.
;-----------------------------------------------------------
LSL48:
  ; Execute the function here
```

```
  push OLoop
  push mpr
  push  XH    ; Save X-ptr
  push  XL

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

  clc
  ldi   OLoop,  6
  LSL48_loop:
    ; shift three bytes (using rotate to propagate carry)
    ld    mpr,  X
    rol   mpr
    st    X+,   mpr
    dec   OLoop
    brne  LSL48_loop

  pop  XL
  pop  XH
  pop mpr
  pop OLoop

  ret


;-----------------------------------------------------------
; Func: ADD48
; Desc: Multiplies two 24-bit numbers in X and Y and stores
;       the result in Z.
;-----------------------------------------------------------
ADD48:
  ; Execute the function here

  push OLoop
  push A
  push B
  push mpr
  push  XH    ; Save X-ptr
  push  XL
  push  YH    ; Save Y-ptr
  push  YL
```

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

  ; Load beginning address of second operand / dest into Y
  ldi  YL, low(MUL24_Result)
  ldi  YH, high(MUL24_Result)

  clc
  ldi   OLoop,  6
  ADD48_loop:
    ; add all 6 bytes
    ld    A,    X+
    ld    B,    Y
    adc   A,    B
    st    Y+,   A
    dec   OLoop
    brne  ADD48_loop

  ; store extra bit if carry
  brcc  ADD48_nocarry
    ldi   mpr,  1
    st    Y,    mpr  ; if carry, set overflow in next byte
  ADD48_nocarry:

  pop  YL
  pop  YH
  pop  XL
  pop  XH
  pop mpr
  pop B
  pop A
  pop OLoop

  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

  ; source addr in prog memory
  ldi   ZL,   low(OperandD << 1)
  ldi   ZH,   high(OperandD << 1)
  ; store in data memory
  ldi   YL,   low(SUB16_OP1)
  ldi   YH,   high(SUB16_OP1)
  ; store two bytes
  lpm     mpr,  Z+
  st      Y+,   mpr
  lpm     mpr,  Z+
  st      Y+,   mpr
  ; store two more bytes for the second operand (directly contiguous)
  lpm     mpr,  Z+
  st      Y+,   mpr
  lpm     mpr,  Z+
  st      Y+,   mpr

  ; do D - E
  rcall   SUB16



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

  ; load SUB16_Result into ADD16_OP1
  ; source addr in data memory
  ldi   ZL,   low(SUB16_Result)
  ldi   ZH,   high(SUB16_Result)
  ; store in data memory
  ldi   YL,   low(ADD16_OP1)
  ldi   YH,   high(ADD16_OP1)
  ; store two bytes
  ld      mpr,  Z+
  st      Y+,   mpr
  ld      mpr,  Z+
```

```
st      Y+,   mpr
; load OperandF into ADD16_OP2
; source addr in prog memory
ldi    ZL,    low(OperandF << 1)
ldi    ZH,    high(OperandF << 1)
; store two more bytes for the second operand (directly contiguous)
lpm     mpr,  Z+
st      Y+,   mpr
lpm     mpr,  Z+
st      Y+,   mpr

; do (ans) + F
rcall   ADD16



; Setup the MUL24 function with ADD16 result as both operands
; Perform multiplication to calculate ((D - E) + F)^2

; copy ADD16_Result into MUL24_OP1
ldi    ZL,    low(ADD16_Result)
ldi    ZH,    high(ADD16_Result)
; store in data memory
ldi    YL,    low(MUL24_OP1)
ldi    YH,    high(MUL24_OP1)
; store three bytes
ld      mpr,  Z+
st      Y+,   mpr
ld      mpr,  Z+
st      Y+,   mpr
ld      mpr,  Z+
st      Y+,   mpr

; copy ADD16_Result into MUL24_OP2
ldi    ZL,    low(ADD16_Result)
ldi    ZH,    high(ADD16_Result)
; store in data memory
ldi    YL,    low(MUL24_OP2)
ldi    YH,    high(MUL24_OP2)
; store three bytes
ld      mpr,  Z+
st      Y+,   mpr
ld      mpr,  Z+
st      Y+,   mpr
```

```
  ld      mpr,  Z+
  st      Y+,   mpr

  ; do (ans) * (ans)
  rcall   MUL24


  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


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

; Enter any stored data you might need here

; ADD16 operands
ADD16_D1:
  .DW 0xFCBA
ADD16_D2:
  .DW 0xFFFF

; SUB16 operands
SUB16_D1:
  .DW 0xFCB9
SUB16_D2:
  .DW 0xE420

; MUL24 operands
MUL24_D1:
  .DB 0xFF, 0xFF, 0xFF, 0 ; extra byte for 16b alignment
MUL24_D2:
  .DB 0xFF, 0xFF, 0xFF, 0 ; extra byte for 16b alignment

; 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

; SUB16 reservations

.org    $0130   ; data memory allocation for operands
SUB16_OP1:
  .byte 2
SUB16_OP2:
  .byte 2
.org    $0140   ; data memory allocation for results
SUB16_Result:
  .byte 2

; MUL24 reservations

.org    $0150   ; data memory allocation for operands
MUL24_OP1:
  .byte 3      ; allocate three bytes for first operand
  .byte 3      ; allocate three more for the shift
MUL24_OP2:
  .byte 3      ; allocate three bytes for second operand

.org    $0160   ; data memory allocation for results
MUL24_Result:
  .byte 6     ; allocate 6 bytes for ADD16 result
```