ECE 375 LAB 5

Introduction to AVR Development Tools

Lab Time: Tuesday 7-9

Aaron Rito

Introduction

Lab 5 uses Atmel Studio and assembly to perform mathematical operations outside the typical design of the micro controller. 16 bit and and subtract, a 24 bit multiplication, and a 24 bit compound statement.

Program Overview

The program performs mathematical operations.

Initialization Routine

The program initializes the at128 chip , and then runs a routine to initialize the stack pointer and direct access registers. Then a short loop is executed to place the strings stored in program memory, to reserved space in data memory, where the LCD functions can access it directly.

Main Routine

The main routine simply runs the functions in order: Loadvals, add16, sub16, mul24, compound

LoadVALs Function

The LoadVals function manages loading all the operands into memory. This is almost the same process as using lpm, but is less confusing for me.

ADD16 Function

ADD16 adds 2 16 bit numbers by first adding the lower byte, then adding a carry bit to the sum of the high byte, and keeping track of a 3rd carry register in case the high byte addition overflows.

Sub16 function

SUB16 subtracts 2 16 bit numbers in the same fashion as the add16, but if the low byte number is 2's compliment (negative), the carry flag is set and subtracted from the high byte.

MUL24 function

MUL 24 multiplies 2 24 bit numbers. It iterates through the memory locations that have the operands in them, multiply them, then shifts the results to increasing registers in memory, adding the carry bits to the results. This is in a process loop that requires 3 iterations, to cycle through the 24 bit operands. The result is a 48 bit number, that is placed in memory in big Endian, for some reason not designed by me.

Study 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 is two's compliment overflow bit. This flag would be set when adding two, two's compliment numbers like 1111111 and 1111111.

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? You can control the address of the results and manage the operands easily. It's not really very different. You still are allocating a certain amount of memory space for said address if you make room in-between. I suppose it's easier because you can see how many bytes are allocated specifically.

Difficulties

My add and subtract functions are not portable, as I was sick and didn't have the time I normally would to spend on this lab. I also didn't write as many comments as usual, due to the sickness. I did manage to get everything working on time, and that's enough for this time. The MUL24 function places the results in big endian for some reason.

Conclusion

When using a micro-controller for large number calculations, you must write routines to handle the overflows from the data registers only being 8 bits. While these operations are certainly possible on a 128 chip, the code can create a lot of overhead and memory usage, if not managed properly. Multiplication requires a result location that is twice as big as the operands. If multiplying 24 bits, you need a 48 bit result location.

Source Code

```
;***********************************************************
;*
;*       Lab 5  ECE 375
;*
;***********************************************************
;*
;*       Author: Aaron Rito
;*         Date: 2/14/17
;*
;***********************************************************


.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
                ldi             mpr, low(RAMEND)        ; initialize Stack Pointer
                out             SPL, mpr
                ldi             mpr, high(RAMEND)
                out             SPH, mpr

                clr             zero

.org    $0046                                   ; End of Interrupt Vectors
```

```
;-----------------------------------------------------------
; Main Program
;-----------------------------------------------------------
MAIN:                                                ; The Main program
                              call LOAD_VALS
                              ;nop
                              ;call ADD16
                              ;nop
                              ;rcall MUL24
                              ;nop
                              ;rcall SUB16
                              rcall COMPOUND


DONE:   rjmp    DONE                  ; Create an infinite while loop to signify the
                                                              ; end of the program.


;*********************************************************
;*      Functions and Subroutines
;*********************************************************
LOAD_VALS:
        ldi r19, 0xA2
                        ldi r20, 0xFF
                        ldi     XL, low(ADD16_OP1)     ; Load low byte of address
                        ldi     XH, high(ADD16_OP1)    ; Load high byte of address
                        ST      X+, r19
                        ST      X+, r20
                        ldi     YL, low(ADD16_OP2)     ; Load low byte of address
                        ldi     YH, high(ADD16_OP2)    ; Load high byte of address
                        ldi r19, 0xF4
                        ldi r20, 0x77
```

```
        ST      Y+, r19
        ST      Y+, r20
        ldi     XL, low(SUB16_OP1)     ; Load low byte of address
        ldi     XH, high(SUB16_OP1)    ; Load high byte of address
        ldi r19, 0xF0
        ldi r20, 0x8A
        ST      X+, r19
        ST      X+, r20
        ldi     YL, low(SUB16_OP2)     ; Load low byte of address
        ldi     YH, high(SUB16_OP2)    ; Load high byte of address
        ldi r19, 0x4B
        ldi r20, 0xCD
        ST      Y+, r19
        ST      Y+, r20
        ldi r20, 6
        ldi     YL, low(addrA) ; Load low byte of address
        ldi     YH, high(addrA)        ; Load high byte of address
        ldi r19, 0xFF
        load_loop:
                ST Y+, r19
                DEC r20
                BRNE load_loop
        ldi             XL, low(OperandD)      ; Load low byte of address
ldi             XH, high(OperandD)     ; Load high byte of address
ldi             YL, low(OperandE)      ; Load low byte of address
ldi             YH, high(OperandE)     ; Load high byte of address
ldi             ZL, low(OperandF)      ; Load low byte of address
ldi             ZH, high(OperandF)     ; Load high byte of address
ldi             r16, $FD
ldi             r17, $51
ldi             r18, $1E
ldi             r19, $FF
```

```
                ldi             r20, $FF
                ldi             r21, $FF
                ST      X+, r16
                ST      X, r17
                ST      Y+, r18
                ST      Y, r19
                ST      Z+, r20
                ST      Z, r21
                        ret
;-------------------------------------------------------------
; Func: ADD16
; Desc: Adds two 16-bit numbers and generates a 24-bit number
;               where the high byte of the result contains the carry
;               out bit.
;-------------------------------------------------------------
ADD16:
                ; Load beginning address of first operand into X

                ldi             XL, low(ADD16_OP1)     ; Load low byte of address
                ldi             XH, high(ADD16_OP1)    ; Load high byte of address
                ldi             YL, low(ADD16_OP2)     ; Load low byte of address
                ldi             YH, high(ADD16_OP2)    ; Load high byte of address
                ldi             ZL, low(ADD16_Result)  ; Load low byte of address
                ldi             ZH, high(ADD16_Result) ; Load high byte of address
                adiw    X, 0x02
                adiw    Y, 0x02
                clr             r20
                clr             r21
                clr             r22
                LD              r16, -X
                LD              r17, -Y
                ADD             r16,r17
```

```
                ADC             r21, zero
                MOV             r20, r16
                LD              r16, -X
                LD              r17, -Y
                ADD             r16, r17
                ADC             r22, zero
                ADD             r21, r16
                ADC             r22, zero
                ST              Z+, r22
                ST              Z+, r21
                ST              Z+, r20

                ret                                     ; End a function with RET

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

                ldi             XL, low(SUB16_OP1)      ; Load low byte of address
                ldi             XH, high(SUB16_OP1)     ; Load high byte of address
                ldi             YL, low(SUB16_OP2)      ; Load low byte of address
                ldi             YH, high(SUB16_OP2)     ; Load high byte of address
                ldi             ZL, low(SUB16_Result)   ; Load low byte of address
                ldi             ZH, high(SUB16_Result) ; Load high byte of address
                adiw    X, 0x02
                adiw    Y, 0x02
                LD              r16, -X
                LD              r17, -Y
```

```
                SUB             r16, r17
                MOV             r20, r16
                LD              r16, -X
                LD              r17, -Y
                SBC             r16, r17
                MOV             r21, r16
                ST              Z+, r21
                ST              Z+,     r20

                ret                                             ; End a function with RET

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



;-----------------------------------------------------------
; Func: COMPOUND
; Desc: Computes the compound expression ((D - E) + F)^2
;               by making use of SUB16, ADD16, and MUL24.
;
;               D, E, and F are declared in program memory, and must
;               be moved into data memory for use as input operands.
;
;               All result bytes should be cleared before beginning.
;-----------------------------------------------------------
COMPOUND:

                ; Setup SUB16 with operands D and E
```

```
; Perform subtraction to calculate D - E
ldi             XL, low(SUB16_OP1)     ; Load low byte of address
ldi             XH, high(SUB16_OP1)    ; Load high byte of address
ldi             YL, low(SUB16_OP2)     ; Load low byte of address
ldi             YH, high(SUB16_OP2)    ; Load high byte of address
ldi             ZL, low(SUB16_Result)  ; Load low byte of address
ldi             ZH, high(SUB16_Result) ; Load high byte of address
adiw    X, 0x02
adiw    Y, 0x02
LD              r16, -X
LD              r17, -Y
SUB             r16, r17
MOV             r20, r16
LD              r16, -X
LD              r17, -Y
SBC             r16, r17
MOV             r21, r16
ST              Z+, r21
ST              Z+,     r20
; Setup the ADD16 function with SUB16 result and operand F
; Perform addition next to calculate (D - E) + F
ldi             XL, low(SUB16_Result)  ; Load low byte of address
ldi             XH, high(SUB16_Result) ; Load high byte of address
ldi             YL, low(ADD16_OP2)     ; Load low byte of address
ldi             YH, high(ADD16_OP2)    ; Load high byte of address
ldi             ZL, low(ADD16_Result)  ; Load low byte of address
ldi             ZH, high(ADD16_Result) ; Load high byte of address
adiw    X, 0x02
adiw    Y, 0x02
clr             r20
clr             r21
clr             r22
```

```
        LD              r16, -X
        LD              r17, -Y
        ADD             r16,r17
        ADC             r21, zero
        MOV             r20, r16
        LD              r16, -X
        LD              r17, -Y
        ADD             r16, r17
        ADC             r22, zero
        ADD             r21, r16
        ADC             r22, zero
        ST              Z+, r22
        ST              Z+, r21
        ST              Z+, r20
        ldi             XL, low(addrA) ; Load low byte
        ldi             XH, high(addrA)        ; Load high byte
        ST              X+, r22
        ST              X+, r21
        ST              X+, r20
        ldi             YL, low(addrB) ; Load low byte
        ldi             YH, high(addrB)        ; Load high byte
        ST              Y+, r22
        ST              Y+, r21
        ST              Y+, r20
        ; Setup the MUL24 function with ADD16 result as both operands
        ; Perform multiplication to calculate ((D - E) + F)^2
        call MUL24

        ret

MUL24:
        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, 3               ; 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 third 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

                ret                                             ; End a function with RET

;-----------------------------------------------------------
; Func: Template function header
; Desc: Cut and paste this and fill in the info at the
;               beginning of your functions
;-----------------------------------------------------------
FUNC:                                           ; Begin a function with a label
                ; Save variable by pushing them to the stack

                ; Execute the function here

                ; Restore variable by popping them from the stack in reverse order
                ret                                             ; End a function with RET



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

; Enter any stored data you might need here
```

```
;*********************************************************
;*      Data Memory Allocation
;*********************************************************

.dseg
.org    $0100                       ; data memory allocation for MUL16 example
addrA:  .byte 3
addrB:  .byte 3
LAddrP: .byte 6

; 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
ADD16_Result:
               .byte 3                      ; allocate three bytes for ADD16 result
.org    $0120                       ; data memory allocation for operands
SUB16_OP1:
               .byte 2                      ; allocate two bytes for first operand of ADD16
SUB16_OP2:
               .byte 2                      ; allocate two bytes for second operand of ADD16
SUB16_Result:
               .byte 3                      ; allocate three bytes for ADD16 result
.org    $0130
OperandD:
               .byte 2                      ; test value for operand D
OperandE:
```

```
                .byte 2                 ; test value for operand E
OperandF:
                .byte 2
CMP_RST:
                .byte 6                         ; test value for operand F
```