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 $((G - H) + I)^2$.

BlockDiagram L4.jpg

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 5 of ECE 375
;*
;*    Author: Astrid Delestine & Lucas Plaisted
;*      Date: Enter Date
;*
;***********************************************************

.include "m32U4def.inc"           ; Include definition file

;************************************************************
;* Variable and Constant Declarations
;************************************************************
.def    mpr = r16                 ; Multi-Purpose Register
.def    waitcnt = r17                 ; Wait Loop Counter
.def    ilcnt = r18               ; Inner Loop Counter
.def    olcnt = r19               ; Outer Loop Counter
.def    hlcnt = r15               ; Hit Left Counter
.def    hrcnt = r14               ; Hit Right Counter
;.def   count = r20               ; needed for LCD binToASCII

.equ    WTime = 50                ; Time to wait in wait loop

.equ    WskrR = 4                 ; Right Whisker Input Bit
.equ    WskrL = 5                 ; Left Whisker Input Bit
.equ    EngEnR = 5                ; Right Engine Enable Bit
.equ    EngEnL = 6                ; Left Engine Enable Bit
.equ    EngDirR = 4               ; Right Engine Direction Bit
.equ    EngDirL = 7               ; Left Engine Direction Bit

;//TAKEN FROM LAB3

.equ    lcdL1 = 0x00              ; Make LCD Data Memory locations constants
.equ    lcdH1 = 0x01
.equ    lcdL2 = 0x10              ; lcdL1 means the low part of line 1's location
.equ    lcdH2 = 0x01              ; lcdH2 means the high part of line 2's locatio
.equ    lcdENDH = 0x01            ; as it sounds, the last space in data mem
.equ    lcdENDL = 0x1F            ; for storing lcd text

;//END TAKEN FROM LAB3

.equ    strSize = 4;

```

```
;///////////////////////////////////////////////////////////////
;These macros are the values to make the TekBot Move.
;///////////////////////////////////////////////////////////////

.equ	MovFwd = (1<<EngDirR|1<<EngDirL)	; Move Forward Command
.equ	MovBck = $00				; Move Backward Command
.equ	TurnR = (1<<EngDirL)			; Turn Right Command
.equ	TurnL = (1<<EngDirR)			; Turn Left Command
.equ	Halt = (1<<EngEnR|1<<EngEnL)		; Halt Command

;***********************************************************
;*	Start of Code Segment
;***********************************************************
.cseg							; Beginning of code segment

;***********************************************************
;*	Interrupt Vectors
;***********************************************************
.org	$0000					; Beginning of IVs
		rjmp	INIT			; Reset interrupt

		; Set up interrupt vectors for any interrupts being used


		; This is just an example:
;.org	$002E					; Analog Comparator IV
;		rcall	HandleAC		; Call function to handle interrupt
;		reti					; Return from interrupt
.org	$0002 ;INT0
		rcall	HitRight		;RIGHT WHISKER
		reti
.org	$0004 ;INT1
		rcall	HitLeft		;LEFT WHISKER
		reti
;.org	$0006 ;INT2
.org	$0008 ;INT3
		rcall	ClearCounters		;CLEAR COUNTERS
		reti
;.org	$000E ;INT6

.org	$0056					; End of Interrupt Vectors

;***********************************************************
;*	Program Initialization
;***********************************************************
```

```
INIT:
	; Initialize the Stack Pointer (VERY IMPORTANT!!!!)
		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

	; Initialize Port B for output
		ldi	mpr, $FF		; Set Port B Data Direction Register
		out	DDRB, mpr		; for output
		ldi	mpr, $00		; Initialize Port B Data Register
		out	PORTB, mpr		; so all Port B outputs are low

	; Initialize Port D for input
		ldi	mpr, $00		; Set Port D Data Direction Register
		out	DDRD, mpr		; for input
		ldi	mpr, $FF		; Initialize Port D Data Register
		out	PORTD, mpr		; so all Port D inputs are Tri-State



	;init the LCD
		rcall LCDInit
		rcall LCDBacklightOn
		rcall LCDClr
		rcall toLCD

		rcall ClearCounters


	; Initialize external interrupts
		; Set the Interrupt Sense Control to falling edge
		ldi mpr, 0b10001010
		sts EICRA, mpr;

	; Configure the External Interrupt Mask
		ldi mpr, 0b0000_1011 ; x0xx_0000 ; all disabled
		out EIMSK, mpr;
	; Turn on interrupts
		; NOTE: This must be the last thing to do in the INIT function
	sei ; Turn on interrupts

;***********************************************************
;*	Main Program
;***********************************************************
MAIN:					; The Main program
```

```

        ldi     mpr, MovFwd         ; Load Move Forward Command
        out     PORTB, mpr

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

;***********************************************************
;*  Functions and Subroutines
;***********************************************************

;-----------------------------------------------------------
;   You will probably want several functions, one to handle the
;   left whisker interrupt, one to handle the right whisker
;   interrupt, and maybe a wait function
;------------------------------------------------------------

;-----------------------------------------------------------
; Func: Template function header
; Desc: Cut and paste this and fill in the info at the
;       beginning of your functions
;-----------------------------------------------------------
ClearCounters:                          ; Begin a function with a label

        ; Save variable by pushing them to the stack

        ; Execute the function here
        clr     hrcnt               ; sets hlcnt and hrcnt to zero by doing an xor
        clr     hlcnt               ; operation with themself

        push ZL                 ; Save vars to stack
        push ZH
        push XL
        push XH
        push mpr
        push ilcnt

        ldi   ZL , low(STRING_BEG<<1)       ; Sets ZL to the low bits
                                    ; of the first string location
        ldi   ZH , high(STRING_BEG<<1)      ; Sets ZH to the first
                                    ; of the first string location
        ldi   XH , lcdH1
        ldi   XL , lcdL1
        ldi   ilcnt , 16

CCl1: ; While ilcnt != zero 1
```

```
        lpm   mpr, Z+
        st    X+ , mpr
        dec   ilcnt
        brne CCl1

        ldi ZL, low(STRING2_BEG<<1)
        ldi ZH, high(STRING2_BEG<<1)
        ;z is already pointing at the second string due to how memory is stored
        ldi   XH , lcdH2
        ldi   XL , lcdL2
        ldi   ilcnt , 16

CCl2: ; While ilcnt != zero 2
        lpm   mpr, Z+
        st    X+ , mpr
        dec   ilcnt
        brne CCl2

        rcall LCDWrite

        pop ilcnt
        pop mpr
        pop XH
        pop XL
        pop ZH
        pop ZL                  ; Pop vars off of stack
        ; Restore variable by popping them from the stack in reverse order

        ret                     ; End a function with RET


;-----------------------------------------------------------
; Func: toLCD
; Desc: Takes various info and pushes it to the LCD
;       *HL#:0
;       *HR#:0
;-----------------------------------------------------------
toLCD:
        push ZL             ; Save vars to stack
        push ZH
        push XL
        push XH
        push mpr
        push ilcnt

        ; Sets ZL to the low bits of the first string location
```

```
        ldi  ZL , low(STRING_BEG<<1)
        ldi  ZH , high(STRING_BEG<<1)
        ;points to the data location where LCD draws from
        ldi  XH , lcdH1
        ldi  XL , lcdL1
        ldi  ilcnt , 4

Line1Loop: ; While ilcnt != zero
        lpm  mpr, Z+
        st   X+ , mpr
        dec  ilcnt
        brne Line1Loop
        //end loop

        mov mpr, hlcnt; copies the counter to mpr

        rcall Bin2ASCII; Takes a value in MPR and outputs the ascii equivilant
        ;convineintly X is currently pointing where I would like this number to


        ldi ZL, low(STRING2_BEG<<1)
        ldi ZH, high(STRING2_BEG<<1)

        ldi  XH , lcdH2
        ldi  XL , lcdL2
        ldi  ilcnt , 4

Line2Loop: ; While ilcnt != zero 2
        lpm  mpr, Z+
        st   X+ , mpr
        dec  ilcnt
        brne Line2Loop


        mov mpr, hrcnt;
        rcall Bin2ASCII

        rcall LCDWrite






        pop ilcnt
        pop mpr
```

```
        pop XH
        pop XL
        pop ZH
        pop ZL                  ; Pop vars off of stack


        ret
;-----------------------------------------------------------
; Sub:  HitRight
; Desc: Handles functionality of the TekBot when the right whisker
;       is triggered.
;-----------------------------------------------------------
HitRight:
        push    mpr             ; Save mpr register
        push    waitcnt             ; Save wait register
        in      mpr, SREG       ; Save program state
        push    mpr             ;

        ; Move Backwards for a second
        ldi     mpr, MovBck ; Load Move Backward command
        out     PORTB, mpr  ; Send command to port
        ldi     waitcnt, (WTime<<1) ; Shifted bit back by 1,
                                    ; making the wait time two seconds
        rcall   Wait                ; Call wait function

        ; Turn left for a second
        ldi     mpr, TurnL  ; Load Turn Left Command
        out     PORTB, mpr  ; Send command to port
        ldi     waitcnt, WTime    ; Wait for 1 second
        rcall   Wait                ; Call wait function

        ; Move Forward again
        ldi     mpr, MovFwd ; Load Move Forward command
        out     PORTB, mpr  ; Send command to port

        pop     mpr       ; Restore program state
        out     SREG, mpr     ;
        pop     waitcnt       ; Restore wait register
        pop     mpr       ; Restore mpr

        inc     hrcnt;
        rcall   toLCD;
        ;fix debounce
        ldi mpr , 0b0000_0001
        out EIFR, mpr
        ret                 ; Return from subroutine
```

```
;-----------------------------------------------------------
; Sub:	HitLeft
; Desc:	Handles functionality of the TekBot when the left whisker
;		is triggered.
;-----------------------------------------------------------
HitLeft:
		push	mpr			; Save mpr register
		push	waitcnt			; Save wait register
		in	mpr, SREG	; Save program state
		push	mpr			;

		; Move Backwards for a second
		ldi	mpr, MovBck	; Load Move Backward command
		out	PORTB, mpr	; Send command to port
		ldi	waitcnt, (WTime<<1)	; Wait for 1 second
		rcall	Wait			; Call wait function

		; Turn right for a second
		ldi	mpr, TurnR	; Load Turn Left Command
		out	PORTB, mpr	; Send command to port
		ldi	waitcnt, WTime	; Wait for 1 second
		rcall	Wait			; Call wait function

		; Move Forward again
		ldi	mpr, MovFwd	; Load Move Forward command
		out	PORTB, mpr	; Send command to port

		pop	mpr		; Restore program state
		out	SREG, mpr	;
		pop	waitcnt		; Restore wait register
		pop	mpr		; Restore mpr

		inc	hlcnt	;
		rcall	toLCD;
			;fix debounce
		ldi mpr, 0b0000_0010
		out EIFR, mpr
		ret			; Return from subroutine

;-----------------------------------------------------------
; Sub:	Wait
; Desc:	A wait loop that is 16 + 159975*waitcnt cycles or roughly
;		waitcnt*10ms.  Just initialize wait for the specific amount
;		of time in 10ms intervals. Here is the general eqaution
;		for the number of clock cycles in the wait loop:
```

```
;               (((((3*ilcnt)-1+4)*olcnt)-1+4)*waitcnt)-1+16
;----------------------------------------------------------
Wait:
        push    waitcnt         ; Save wait register
        push    ilcnt           ; Save ilcnt register
        push    olcnt           ; Save olcnt register

Loop:   ldi     olcnt, 224      ; load olcnt register
OLoop:  ldi     ilcnt, 237      ; load ilcnt register
ILoop:  dec     ilcnt           ; decrement ilcnt
        brne    ILoop           ; Continue Inner Loop
        dec     olcnt       ; decrement olcnt
        brne    OLoop           ; Continue Outer Loop
        dec     waitcnt     ; Decrement wait
        brne    Loop            ; Continue Wait loop

        pop     olcnt       ; Restore olcnt register
        pop     ilcnt       ; Restore ilcnt register
        pop     waitcnt     ; Restore wait register
        ret             ; Return from subroutine



;-----------------------------------------------------
; 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
;.org
STRING_BEG:
.DB     "HL#:0␣␣␣␣␣␣␣␣␣␣␣"      ; Declaring data in ProgMem
```

```
STRING2_BEG:
.DB		"HR#:0           "
STRING_END:


;***********************************************************
;*	Additional Program Includes
;***********************************************************
.include "LCDDriver.asm"		; Include the LCD Driver
```