CPE 325: Intro to Embedded Computer System

Lab 05

Subroutines, Passing Parameters, Hardware Multiplier, Interrupts in C

Submitted by: Hannah Kelley

Date of Experiment: September 24, 2025

Report Deadline: October 8, 2025

Demonstration Deadline: October 8, 2025

Theory

Topic 1: Subroutines

a) In MSP430 assembly, a subroutine is a reusable block of code called with CALL and ended with RET, which returns execution by popping the stored return address from the stack. When called, the MSP430 automatically pushes the return address, and the programmer must preserve any modified registers (typically R4–R15) with PUSH and POP to avoid corrupting the caller's data. Arguments and return values are usually passed through registers such as R12 or R15. Since the stack grows downward and stores both return addresses and saved registers, maintaining stack balance is essential—every push must have a matching pop. Errors like missing a RET, not restoring registers, or mismanaging the stack can cause crashes, making stack discipline and register preservation critical when using subroutines.

Topic 2: Interrupt Vector

a) In MSP430, the interrupt vector is a table at the top of memory where each entry holds the address of the Interrupt Service Routine (ISR) for a specific interrupt. When an interrupt occurs, the CPU automatically saves the current PC and status register on the stack, then loads the ISR address from the vector table. The ISR must end with RETI, which restores the saved state and returns to normal execution. Correctly assigning ISR addresses to their vector locations is essential to ensure the right code runs when an interrupt is triggered.

Topic 3: Hardware Multiplier

a) The MSP430 includes a built-in hardware multiplier module that performs multiplication faster and more efficiently than software routines. It supports operations like unsigned, signed, and multiply-accumulate by writing operands into dedicated registers such as MPY, MPYS, MAC, or MACS. After the two operands are loaded, the result is automatically placed in the RESLO and RESHI registers, without needing manual shifting or looping. Using the hardware multiplier saves cycles, reduces code size, and is especially useful in math-heavy operations like signal processing, exponentiation, or array calculations. Properly selecting the correct operand registers and retrieving the full result is key to using it correctly.

Topic 4: Passing Parameters

Parameters in MSP430 assembly can be passed in three common ways: registers, the stack, or fixed memory locations.

a) The most efficient and widely used method is passing through registers, where arguments are placed in registers like R12–R15 before calling a subroutine, and results are often returned the same way.
b) A second method is using the stack, where values are pushed before the CALL and accessed relative to the stack pointer inside the subroutine—this is useful when many parameters are needed or registers are limited.
c) The third approach is storing parameters in memory, such as global variables or predefined memory addresses, which the subroutine accesses directly; this method is slower but useful for large data or shared values.

Results & Observation

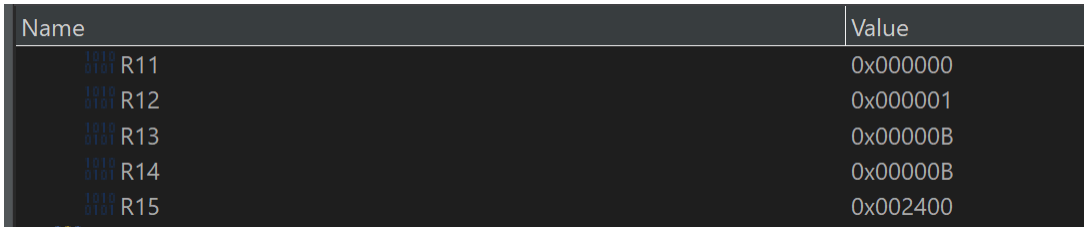
Program 1:

Program Description:

The goal of this lab is to write an MSP430 assembly program that computes the sum of all elements in an integer array after raising each element to a specified positive exponent. The program must use two separate subroutines to perform the calculation: SW_Mult, which implements multiplication using the shift-and-add software algorithm, and HW_Mult, which uses the MSP430's hardware multiplier. The array, its size, and the exponent are initialized in the main program, and both subroutines receive these parameters. For each array element, a loop raises the value to the given exponent by repeated multiplication, and the powered result is added to a running total. The input array must contain at least six hard-coded values in the range –8 to +8, and only exponents of 2 or higher need to be supported. After both implementations run, the number of clock cycles used by each must be measured and compared to determine which method is more efficient and why.

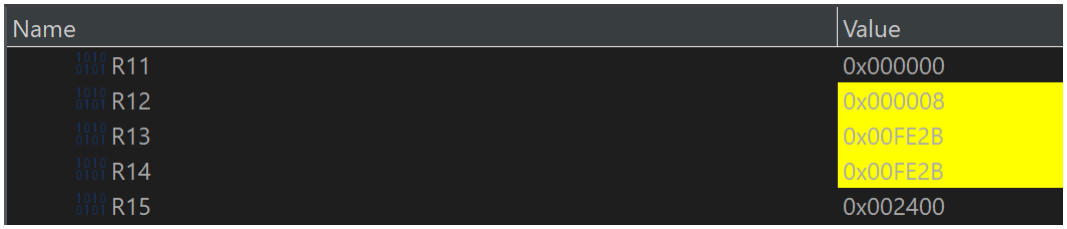
No flowchart has been included for this problem because the Lab 5 assignment document did not require it.

Program Output:



Name	Value
R11	0x000000
R12	0x000001
R13	0x00000B
R14	0x00000B
R15	0x002400

Figure 01: Program 1 Results (R12 and R13) with Array = [2, -2, 1, 2, 1, 1]



Name	Value
R11	0x000000
R12	0x000008
R13	0x00FE2B
R14	0x00FE2B
R15	0x002400

Figure 02: Program 1 Results (R12 and R13) with Array = [-8, 2, 3, 1, -1, 2]

Report Questions:

1. **Which subroutine is more efficient and why?**
   After testing both the sw_mult and hw_mult subroutines with different input arrays of at least size six the hw_mult routine performed more efficiently. Both arrays were tested with the following arrays: A1 = [-8, 2, 3, 1, -1, 2], A2 = [2, 4, 6, -3, -5, 1], and A3 = []. The subroutine

sw_mult performed the calculation in 1,865, 1,867, and 1,891 clock cycles respectively. The subroutine sw_mult performed the calculation in 290, 290, and290 clock cycles respectively. These results demonstrate that hw_mult is significantly more efficient. This is expected, as the hardware multiplier utilizes built-in registers to perform operations directly, avoiding the iterative algorithm used by sw_mult.

Program 2:

Program Description:

The objective of this lab is to develop an MSP430 assembly program that uses interrupt-driven input from two switches, SW1 and SW2, to control two LEDs, LED1 and LED2. At startup, both LEDs must be turned off. SW1 is responsible for toggling LED2: each press changes its state from off to on or on to off. This behavior must persist across multiple presses, with each press detected through an interrupt. SW2 triggers a different response—when pressed, LED1 must blink exactly four times at a frequency of 8 Hz, after which LED2 is toggled once. All switch handling must rely solely on interrupts rather than polling, ensuring responsive and efficient interaction between the switches and LEDs.

No program output has been included because all outputs are the state of LEDs on the lab board. No flowchart has been included because it was not required by the Lab 5 Assignment document.

Appendix

Table 01: Program 1 Main Function

```
;-------------------------------------------------------------------------------
; File:       main.asm
; Description: raise all elements of an array to the same power then add those
;               results together. Multiplication is done with software and hardware
;              multiplication
; Board:      5529
; Input:      array of 6+ elements, size of array, and exponent
; Output:
; Author:     Hannah Kelley
; Date:       September 24, 2025
;-------------------------------------------------------------------------------
        .cdecls C,LIST,"msp430.h"       ; Include device header file

;-------------------------------------------------------------------------------
        .def    RESET                   ; Export program entry-point to
                                        ; make it known to linker.
                                        .ref sw_mult
                                        .ref hw_mult

        .data
array:      .word -8, 2, 3, 1, -1, 2                ; array of siged ints (16 bits)
size:                   .word 6                                 ; number of elements
exponent:.word 3                                                ; exponent
        .text
;-------------------------------------------------------------------------------
        .text                           ; Assemble into program memory.
        .retain                         ; Override ELF conditional linking
                                        ; and retain current section.
        .retainrefs                     ; And retain any sections that have
                                        ; references to current section.
;-------------------------------------------------------------------------------
RESET       mov.w  #__STACK_END,SP          ; Initialize stackpointer
StopWDT     mov.w  #WDTPW|WDTHOLD,&WDTCTL   ; Stop watchdog timer
;-------------------------------------------------------------------------------
; Main loop here
;-------------------------------------------------------------------------------
main:
                                        ; sw_mult call
                                        mov.w #array, R15
                                        mov.w size, R14
                                        mov.w exponent, R11

                                        push R15
                                        push R14
                                        push R11
```

```
                            call #sw_mult                   ; R13 = sw_mult result
                            add #6, SP                              ; clean up stack
                            ; hw_mult call
                            mov.w #array, R15
                            mov.w size, R14
                            mov.w exponent, R12
                            sub #2, SP                      ; make space for result
                            push R15
                            push R14
                            push R12
                            call #hw_mult
                            mov 6(SP), R14                  ; R12 = hw_mult result
                            add #8, SP                              ; clean up stack
                            jmp $
                            nop
;-------------------------------------------------------------------------------
; Stack Pointer definition
;-------------------------------------------------------------------------------
        .global __STACK_END
        .sect   .stack

;-------------------------------------------------------------------------------
; Interrupt Vectors
;-------------------------------------------------------------------------------
        .sect   ".reset"            ; MSP430 RESET Vector
        .short  RESET
```

Table 02: Program 1 sw_mult Function

```
;-------------------------------------------------------------------------------
; File:         sw_mult.asm
; Description: multiply numbers using the shift and add algorithm
; Board:        5529
; Input:           base address of array, exponent, and length of array
; Output:          sigma(array[i]^exponent)
; Author:       Hannah Kelley
; Date:         September 24, 2025
;-------------------------------------------------------------------------------
                .cdecls C,LIST,"msp430.h"       ; Include device header file
                .def sw_mult
;-------------------------------------------------------------------------------
sw_mult:                        push R4 ; push whatever registers we use
                                push R5
                                push R6
                                push R7
                                push R8
                                push R9
                                push R10
                                push R11
                                push R12
                                clr R13         ; R13 = accumulator/result
```

```
                                mov 24(SP), R4  ; R4 = base address of array
                                mov 20(SP), R6  ; R6 = exponent
                                mov 22(SP), R5  ; R5 = length of array
outerloop:
                                cmp #0, R5      ; compare length with 0
        jz outerend; if length = 0 we have reached the end of the array
                                mov @R4+, R7    ; R7 = base
                                mov R7, R12     ; R12 = current power result
                                mov R6, R11     ; R11 = exponent counter
                                dec R11         ; exponent counter--
                                jz elementdone
innerloop:
                                mov R12, R8     ; R8 = multiplicand
                                mov R7, R9      ; R9 = multiplier
                                clr R10         ; R10 = running product
                                mov #16, R14    ; R14 = bit counter
mulloop:
                                bit #1, R9      ; test LSB of multipler (R9)
                                jz skipadd      ; if LSB = 0 do not add
                add R8, R10     ; if LSB = 1, running product += multiplicand
skipadd:                         rla R8         ; multiplicand *= 2
                                rra R9          ; multiplier /= 2
                                dec R14
                                jnz mulloop

                                mov R10, R12    ; update power result
                                dec R11         ; exponent counter--
                                jnz innerloop
elementdone:
                                add R12, R13           ; R13 += R12
                                dec R5                         ; length--
                                jmp outerloop
outerend:
                                pop R12
                                pop R11
                                pop R10
                                pop R9
                                pop R8
                                pop R7
                                pop R6
                                pop R5
                                pop R4
                                ret
                                .end
```

Table 03: Program 1 hw_mult Function

```
/*-------------------------------------------------------------------------
 * File:        Lab01_P2.c
 * Function:    ...
 * Description: ...
 * Input:       ...
 * Output:      ...
 * Author(s):   Your Name
 * Date:        ...
```

```
;-------------------------------------------------------------------------
; File:        hw_mult.asm
; Description: multiply numbers using hte hardware multiplier
; Board:       5529
; Input:          base address of array, exponent, and length of array
; Output:         product
; Author:      Hannah Kelley
; Date:        September 24, 2025
;-------------------------------------------------------------------------
               .cdecls C,LIST,"msp430.h"       ; Include device header file
               .def hw_mult
;-------------------------------------------------------------------------
hw_mult:
                push R4                 ; save registers
                push R5
                push R6                 ; will use R6 for exponent counter
                push R7                 ; will hold temp result
                mov 14(SP), R4          ; array pointer
                mov 12(SP), R5          ; array length
                mov 10(SP), R6          ; exponent (power)
                clr R14                 ; accumulator = 0
PowLoop:
                mov @R4+, R7            ; load next array element into R7
                mov R7, R12             ; base value into R12
                mov R6, R11          ; copy exponent into R11 (loop counter)
                dec R11                 ; exponent-1 multiplications needed
                jz PowDone              ; if exponent = 1, skip inner loop
PowInnerLoop:   mov R7, MPY             ; load base into MPY
                mov R12, OP2            ; multiply current result by base
                nop
                nop
                nop
                mov RESLO, R12          ; store multiplication result
                dec R11
                jnz PowInnerLoop
PowDone:        add R12, R14            ; add powered value into accumulator
                dec R5
                jnz PowLoop
                mov R14, 16(SP)
                pop R7
                pop R6
                pop R5
                pop R4
                ret
                .end
```

Table 04: Program 2 Source Code

```
;-------------------------------------------------------------------------
----
; File:        main.asm
; Description: LEDs start off. SW1 toggles LED2 on each press.
;                             SW2 makes LED1 blink 4 times at 8 Hz, then
toggles LED2.
; Board:       5529
; Input:       SW1 and SW2
; Output:      LED1 and LED2
```

```
; Author:       Hannah Kelley
; Date:         September 24, 2025
;-------------------------------------------------------------------------
            .cdecls C,LIST,"msp430.h"
;-------------------------------------------------------------------------
            .def    RESET
            .def SW1_ISR
            .def SW2_ISR

            .data
            .text
            .retain
            .retainrefs

;-------------------------------------------------------------------------
RESET       mov.w  #__STACK_END,SP            ; Initialize stackpointer
StopWDT     mov.w  #WDTPW|WDTHOLD,&WDTCTL     ; Stop watchdog timer

Setup:          ; LEDs
                bis.b #0x01,&P1DIR            ; set P1.0 as output
                bis.b #0x80,&P4DIR            ; set P4.7 as output
                bic.b #0x01,&P1OUT            ; turn P1.0 OFF
                bic.b #0x80,&P4OUT            ; turn P4.7 OFF

                        ; SW1
                bic.b #002h, &P2DIR    ; SET P2.1 as input for SW1
                bis.b #002h, &P2REN    ; Enable Pull-Up resister at P2.1
                bis.b #002h, &P2OUT    ; required for proper IO set up
                bis.b #002h, &P2IES    ; Select High-to-Low (falling edge)
                bis.b #002h, &P2IE     ; Enable P2.1 interrupt
                bic.b #002h, &P2IFG    ; Clear any pending P2.1 interrupt
flag

                ; SW2
                bic.b #002h, &P1DIR    ; SET P1.1 as input for SW2
                bis.b #002h, &P1REN    ; Enable Pull-Up resister at P1.1
                bis.b #002h, &P1OUT    ; required for proper IO set up
                bis.b #002h, &P1IES            ; Select High-to-Low
transition (falling edge)
                bis.b #002h, &P1IE             ; Enable P1.1 interrupt
                bic.b #002h, &P1IFG    ; Clear any pending P1.1 interrupt
flag

                        ; Enable global interrupts
                bis.w #GIE, SR       ; Enable Global Interrupts (GIE)

                        jmp $

; SW1_ISR Function ------------------------------------------------------
SW1_ISR:        bic.b #002h, &P2IFG    ; Clear interrupt flag
ChkSW1:         bit.b #002h, &P2IN     ; Check if S1 is pressed
                jnz S1Exit     ; If not zero,loop and check again
Debounce1:      mov.w #2000, R15               ; Set to (2000 * 10 cc )

SW120ms:        dec.w R15                               ; Decrement R15
                nop
                nop
```

```
                nop
                nop
                nop
                nop
                nop
                jnz SW120ms                     ; Delay over?
                bit.b #02h, &P2IN               ; Verify S1 is still pressed
                jnz S1Exit      ; If not, wait for S1 press
LEDon:          xor.b #0x80, &P4OUT     ; toggle LED2
SW1wait:        bit.b #002h, &P2IN      ; Test S1
                jz SW1wait              ; Wait until S1 is released
S1Exit:         reti                    ; Return from interrupt

; SW1_ISR Function ------------------------------------------------------------
SW2_ISR:        bic.b #002h, &P1IFG     ; Clear interrupt flag
ChkSW2:         bit.b #002h, &P1IN      ; Check if S2 is pressed ;
                jnz S2Exit ; If not zero, loop and check again
Debounce2:      mov.w #2000, R14                ; Set to (2000 * 10 cc )

SW220ms:        dec.w R14                       ; Decrement R14
                Nop
                nop
                nop
                nop
                nop
                nop
                jnz SW220ms                     ; Delay over?
                bit.b #02h, &P1IN               ; Verify S2 is still pressed
                jnz S2Exit      ; If not, wait for S2 press
                mov.w #8, R5
toggle:         xor.b #0x01,&P1OUT      ; toggle on LED1
                mov.w #31250, R13
delay625:       dec.w R13
                jnz delay625
                dec.w R5
                jnz toggle
                xor.b #0x80, &P4OUT
SW2wait:        bit.b #002h, &P1IN              ; Test S2
                jz SW2wait              ; Wait until S2 is released
S2Exit:         reti                            ; Return from interrupt
; Stack Pointer definition-----------------------------------------------------
            .global __STACK_END
            .sect  .stack
;Interrupt Vectors------------------------------------------------------------
            .sect  ".reset"
            .short RESET
            .sect ".int47"
            .short SW2_ISR
            .sect ".int42"
            .short SW1_ISR
                        .end
```