CPE 325: Intro to Embedded Computer System

Lab 05

Subroutines, Passing Parameters, and Hardware Multiplier

Submitted by: Anshika Sinha

Date of Experiment: 02/05/2025

Report Deadline: 04/15/2025

Demonstration Deadline: 04/16/2025

Theory

Topic 1: Subroutines

a) A subroutine is a block of code that performs a specific task and can be reused multiple times within a program.

b) Subroutines avoid code duplication, saving memory space. Only one copy of the subroutine is stored in memory, and programs branch to its location when needed.

c) A CALL instruction branches to the subroutine's starting location. A RETURN instruction sends execution back to the calling program.

   a. The CALL instruction pushes the PC onto the stack, updates the stack pointer, and branches to the subroutine.

   b. The RETURN instruction pops the return address from the stack into the PC, resuming execution from the correct location.

d) The return address of the main program or parent subroutine must be saved for returning correctly. MSP430 uses a link register or stack for storing return addresses.

e) Subroutine Nesting: A subroutine can call another subroutine. Each nested call must save the previous return address before overwriting it. Uses a stack to manage multiple return addresses.

Topic 2: Passing parameters: When calling a subroutine, parameters need to be passed between the caller and the subroutine.

a) Passing Parameters via Registers
   a. Efficient and straightforward method.
   b. Parameters are stored in general-purpose registers.
   c. Since registers are used for passing parameters, they do not need to be pushed/popped from the stack unless other temporary registers are modified.

b) Passing Parameters via Memory

   a. Parameters are stored in specific memory locations.

   b. Subroutine accesses these memory locations to read/write parameters.

   c. Used when registers are insufficient or when data needs to be available beyond subroutine execution.

   d. Requires memory read/write operations, which can slow down execution compared to register passing.

c) Passing Parameters via the Stack

   a. Used when there are many parameters and registers are insufficient.
   b. Parameters are pushed onto the stack before calling the subroutine.
   c. The subroutine retrieves input parameters from the stack based on stack organization.

Topic 3: Hardware multiplier

a) The MSP430 includes a hardware multiplier to efficiently perform multiplication operations. Standard multiplication using instructions can be slow, but the hardware multiplier speeds up calculations with fewer instructions.
b) It supports 16 x 16 bit multiplication for signed and unsigned numbers, with or without an accumulator.
c) The twelve available registers determine the multiplication type. OP2 holds the second operand. The result is stored in RES0 to RES3. SUMEXT works with RESLO and RESHI for 16×16-bit multiplication.

Results & Observation

Program 1:

Program Description:

This MSP430 program calculates the dot product of two given arrays. The two input arrays are initialized with 6 values each, ranging between -32 and +32. The dot product is found by summing up the products of the corresponding elements in the array. SW_Mult implements the Shift-and-Add Multiplication Algorithm to manually compute the product of two numbers using bitwise operations. It iterates through both arrays, computing products using shifts and conditional additions. Then, it accumulates the result into a dot product sum. HW_Mult uses the MSP430's hardware multiplier to perform fast multiplication. Iterates through the arrays, multiplying elements using the dedicated multiplication registers, MPY and OP2. The HW_Mult result is stored in register R13 and the SW_Mult result is stored in register R12.

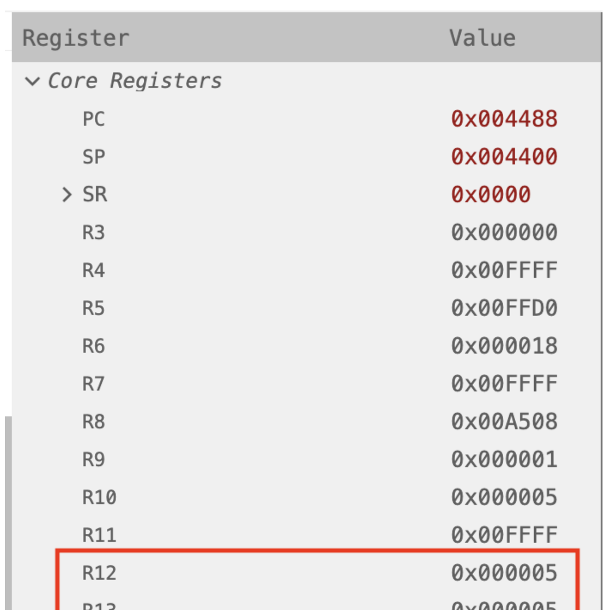
Program Output:

I. With input arrays:

ARR1: .int -1, 1, 1, 1, 1, 1

ARR2: .int 2, 2, 2, 2, -4, 5

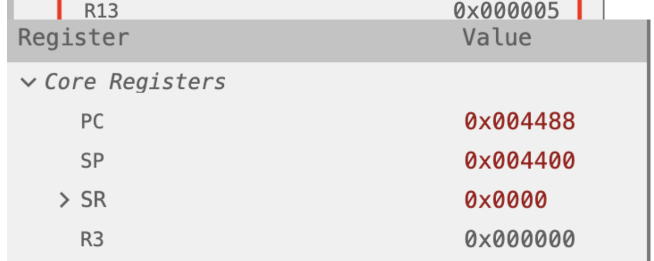
Dot product result: 0x0005



Register	Value
Core Registers	
PC	0x004488
SP	0x004400
SR	0x0000
R3	0x000000
R4	0x00FFFF
R5	0x00FFD0
R6	0x000018
R7	0x00FFFF
R8	0x00A508
R9	0x000001
R10	0x000005
R11	0x00FFFF
R12	0x000005
R13	0x000005

Figure 01: Program 1 output with first set of inputs

II. With input arrays:



Register	Value
Core Registers	
PC	0x004488
SP	0x004400
SR	0x0000
R3	0x000000

ARR1: .int -1, 1, 1, 1, 1, 0

ARR2: .int 2, 2, 2, 2, -4, 5

Dot product result: 0x0000

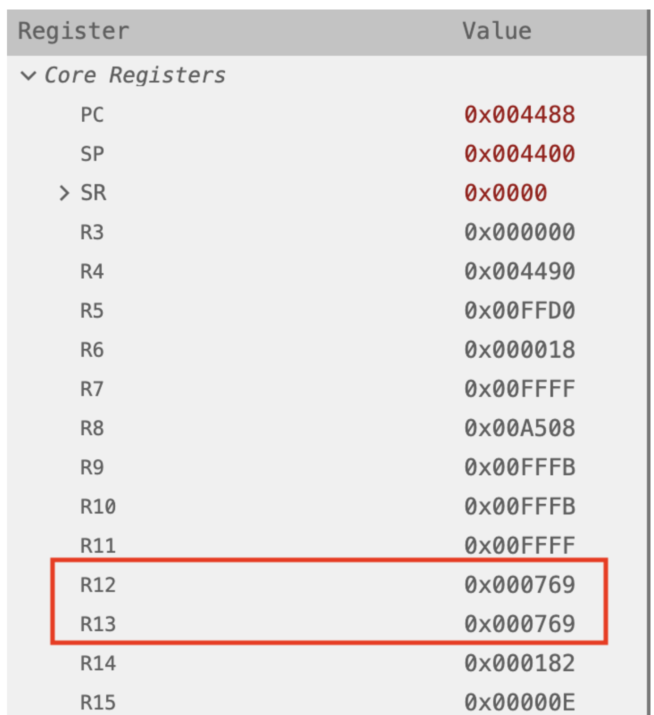
Figure 02: Program 1 output with second set of inputs

III. With input arrays:

ARR1: .int 1,2,4,7,31,-29,-5

ARR2: .int 1,2,4,7,31,-29,-5

Dot product result: 0x0769



Register	Value
Core Registers	
PC	0x004488
SP	0x004400
SR	0x0000
R3	0x000000
R4	0x004490
R5	0x00FFD0
R6	0x000018
R7	0x00FFFF
R8	0x00A508
R9	0x00FFFB
R10	0x00FFFB
R11	0x00FFFF
R12	0x000769
R13	0x000769
R14	0x000182
R15	0x00000E

Figure 03: Program 1 output with third set of inputs

For SW_Mult, each multiplication takes multiple cycles since loop iterations depend on bit count. Additional cycles are required for shifting and conditional branches. This program's SW multiplier took 944 clock cycles to compute the dot product for the first set of inputs.

HW_Mult uses the MSP430 hardware multiplier and the multiplication result is ready in a few clock cycles. It took this program's HW multiplier subroutine 147 clock cycles to compute the dot product for the first set of inputs. This subroutine is much faster than the software multiplier subroutine.

Program Flowchart:

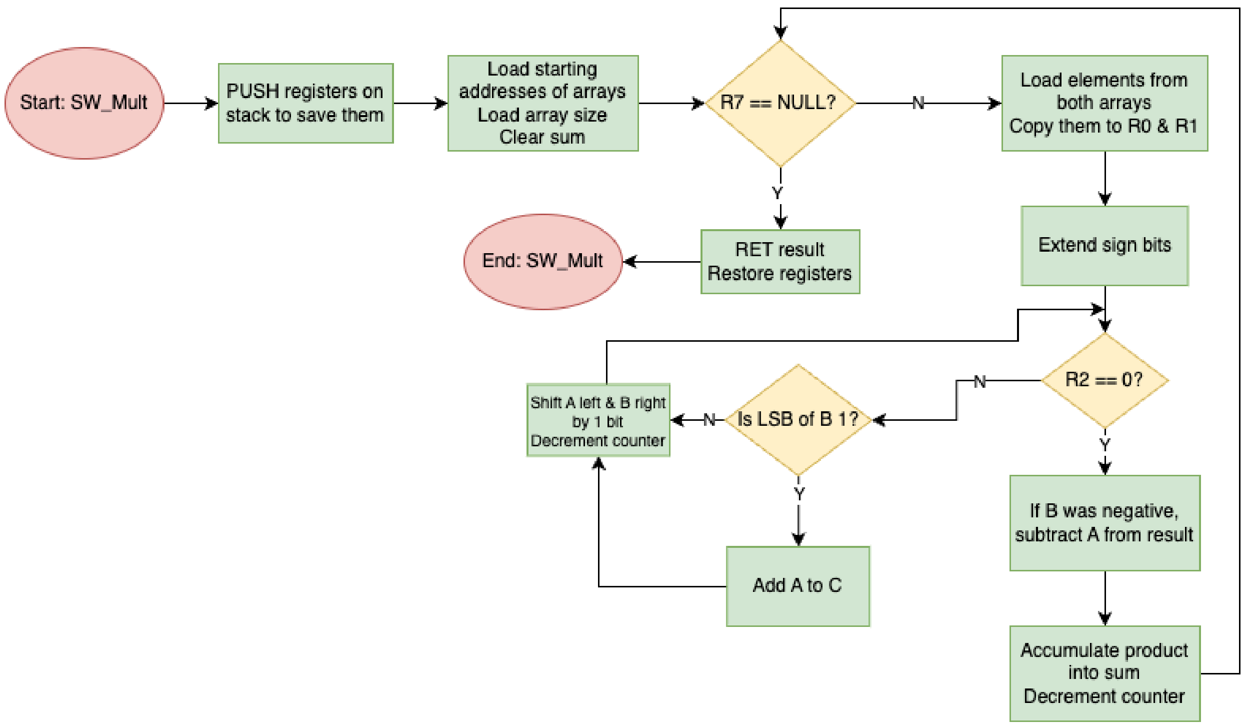


Figure 04: SW_Mult subroutine flowchart

Report Questions:

1. **How do you pass parameters to a subroutine using stack? Explain how you extract parameters that you pass using this technique.**

Parameters are passed to the subroutine through the stack by pushing those parameters, like the starting addresses of the arrays and the length of the array, onto the stack. These parameters are extracted by the subroutine from the stack by using indexed addressing mode (e.g. mov 8(SP), R6.

Conclusion

In conclusion, this program calculates the dot product of two given arrays using a software multiplier and a hardware multiplier. The hardware multiplier is much faster due to its reliance on hardware. The software multiplier produces the same results, although it takes a lot more clock cycles. This lab demonstrates the use of subroutines to execute tasks, passing parameters through the stack, and the capabilities of the hardware multiplier.

Appendix

Table 01: Program 1 main source code

```
;-------------------------------------------------------------------------------
; * File:        main.asm
; * Function:    Calculate dot product of two arrays
; * Description: This MSP430 program calculates the dot product of two given
arrays.
; The two input arrays are initialized with 6 values each, ranging between -32
and +32.
;
; * Input:       Two arrays with at least 6 values ranging from -32 to +32
; * Output:      Pass the result on program stack
; * Author(s):   Anshika Sinha
; * Date:        April 13, 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
;-------------------------------------------------------------------------------
            .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
            mov.w   #WDTPW|WDTHOLD,&WDTCTL  ; Stop watchdog timer

;-------------------------------------------------------------------------------
; Main loop here
;-------------------------------------------------------------------------------ma
in:
            ; CALL HW_Mult
            push    #ARR1             ; Push address of 1st array onto stack
            push    #ARR2             ; Push address of 2nd array onto stack
            push    #6                ; Push array length onto stack
            call    #HW_Mult
            add     #6, SP            ; Clean up stack

            ; CALL SW_Mult
            push    #ARR1             ; Push address of 1st array onto stack
            push    #ARR2             ; Push address of 2nd array onto stack
            push    #6                ; Push array length onto stack
            call    #SW_Mult
            add     #6, SP

            jmp     $                       ; Infinite loop
            nop                             ; Required for debug

.data
```

```
ARR1:      .int -1, 1, 1, 1, 1, 0
ARR2:      .int 2, 2, 2, 2, -4, 5

;-------------------------------------------------------------------------------
; Stack Pointer definition
;-------------------------------------------------------------------------------
            .global __STACK_END
            .sect   .stack

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

Table 02: Program 1 SW_Mult subroutine code

```
; -------------------------------------------------------------------------------
; File:          Lab5_SW_Mult.asm
; Function:      Software multiplication subroutine
; Description:   Computes the dot product of two given arrays using the Shift
;                 and Add multiplication algorithm.
; Input:         Calling the subroutine
; Output:        The result is returned through the stack
; Author:        Anshika Sinha
; Date:          April 13, 2025
; -------------------------------------------------------------------------------
        .cdecls C, LIST, "msp430.h"     ; Include device header file
; -------------------------------------------------------------------------------
        .def    SW_Mult
        .text

SW_Mult:
    push R4               ; Save registers onto stack
    push R5
    push R6
    push R7
    push R8
    push R9
    push R10
    push R11

    mov 22(SP), R4      ; R4 = pointer to array1
    mov 20(SP), R5      ; R5 = pointer to array2
    mov 18(SP), R6      ; R6 = length
    clr R12             ; R12 = result

SW_Loop:
    jz SW_End       ; If length is 0, end loop

    mov @R4+, R9        ; Load array1 element into R7 (multiplicand)
```

```
    mov @R5+, R10       ; Load array2 element into R8 (multiplier)
    clr R11            ; Clear product accumulator
    ;mov R7, R9         ; Copy multiplicand to R9
    ;mov R8, R10        ; Copy multiplier to R10
    mov #16, R7        ; Loop counter

SW_ShiftAdd:
    bit #1, R10
    jz  SW_Shift

    add R9, R11        ; If LSB = 1, add R9 (multiplicand)

SW_Shift:
    rla R9             ; Shift multiplicand left
    rra R10            ; Shift multiplier right
    dec R7
    jnz SW_ShiftAdd

    bit #1, R8
    jz SW_Result
    sub R9, R11        ; If multiplier was negative, subtract

SW_Result:
    add R11, R12       ; Accumulate result
    dec R6
    jmp SW_Loop

SW_End:
    pop R11             ; Restore registers
    pop R10
    pop R9
    pop R8
    pop R7
    pop R6
    pop R5
    pop R4
    ret                 ; Return result
    .end
```

Table 03: Program 1 HW_Mult subroutine code

```
;-----------------------------------------------------------------------------
; File:         Lab5_HW_Mult.asm
; Function:     Hardware Multiplication subroutine
; Description:  Calculates dot product using the hardware multiplier
; Input:        Calling the subroutine
; Output:       The result is returned through the stack
; Author:       Anshika Sinha
; Date:         April 13, 2025
;-----------------------------------------------------------------------------
        .cdecls C, LIST, "msp430.h"     ; Include device header file
;-----------------------------------------------------------------------------
        .def    HW_Mult
        .text
```

```
HW_Mult:
    push R4              ; Save registers onto stack
    push R5
    push R6

    mov 12(SP), R4      ; R4 = array1 pointer
    mov 10(SP), R5       ; R5 = array2 pointer
    mov 8(SP), R6       ; R6 = array length
    clr R13             ; Clear result register

HW_Loop:
    mov @R4+, R9       ; Load element from array1 into R9
    mov @R5+, R10       ; Load element from array2 into R10

    mov R9, MPY       ; Load element from array1 into MPY
    mov R10, OP2       ; Load element from array2 into OP2

    nop
    nop
    nop
    add RESLO, R13      ; Add lower result to sum
    dec R6
    jnz HW_Loop

DotHW_End:
    pop R6               ; Restore registers
    pop R5
    pop R4
    ret                  ; Return result
    .end
```