Embedded Systems Lab

CPE 325-02

Introduction to Subroutines

By: David Thornton

Lab Date:

Lab Due:

Demonstration Due:

Introduction

This lab introduces the concept of subroutines with the MSP430.

Theory

Topic 1: Subroutines

- "In computer programming, a subroutine is a sequence of program instructions that performs a specific task, packaged as a unit. This unit can then be used in programs wherever that particular task should be performed." - Source

Topic 2: Passing parameters

- "To pass parameters to a subroutine, the calling program pushes them on the stack in the reverse order so that the last parameter to pass is the first one pushed, and the first parameter to pass is the last one pushed. This way the first parameter is on top of the stack and the last one is at the bottom of the stack." - Source

Topic 3: Hardware multipliers

- "The hardware multiplier is realized as each other 16 bit peripheral module, and not integrated into the CPU. The CPU is unchanged through all configurations, and the instruction set is not modified. It takes no extra cycle for multiplication. Both operands are loaded into the multiplier's register and the result can be accessed immediately after loading the second operand." - Source

Lab Assignment

1. Write an assembly program that passes a base number b (value should be other than 0 and 1) to a subroutine calc_power. This subroutine should populate two arrays in memory with b^1, b^2, b^3, b^4, b^5. This means your subroutine should compute the first 5 powers of a parameter passed into it. [One of the arrays is populated with results using hardware multiplier and the other using software multiplier.] You must pass b to calc_power using a register of your choice. You may also want to pass the address of your result. Pass these addresses using stack. In order to compute the powers, you need multiplication operations. For this you must implement two additional sub routines as defined below in Q2.

2. One of the subroutines that you need to implement is SW_Mult that uses the Shift-and-Add multiplication algorithm. The algorithm is described in the provided pdf file. Another subroutine that you need to implement is HW_Mult. It should use Hardware Multiplier to multiply numbers. Both of these subroutines should take in input numbers through stack and return the result of multiplication in one of the registers.

   Hints:
   a. You may want to allocate space for results in main. You can allocate space using **".bss"** for both results using hardware multiplier and software multiplier.
   b. In subroutine calc_power, you can repeatedly call SW multiplier subroutine to populate the memory space allocated for software based results. Then, you can use a similar approach to populate memory space allocated for hardware based results.

3. Measure the number of clock cycles used by each subroutine for a small range of values. Comment on the efficiency of each subroutine.

4. **Bonus (up to 15pts)**: Create a subroutine that converts a string of at least a five-digit number to its numerical value by using the Hardware Multiplier and stores the result in a variable in the memory. The subroutine needs to receive the base address of the string and the address of the variable through the program stack. For full credit you need to use the accumulator. If the accumulator is not used only up to 10pts will be rewarded.

Observations

It is quite clear that the software approach to this problem is nearly three times slower (in terms of clock cycles) than the hardware approach (per appendices 7 and 8). The hardware approach is not only simpler, has few lines of code, it also is far less complicated to understand. The calc_power subroutine is moderately efficient, but could be improved. [Source for clock cycles]

This image displays the result for b = 3.

```
0x002400  0003 0009 001B 0051 00F3 0003 0009 001B 0051
0x002412  00F3 0003 0001 A375 EAFF BA6C 92BD DADD 0F4C
```

Conclusion

This lab was successful in introducing me to subroutines.

[Demo link]

Appendix

Appendix 1: Lab_5.asm

```
;-------------------------------------------------------------------------------
; File: Lab_5.asm
; Description: This program calculates the first five powers of an integer b.
;                        This program uses software (SW) and hardware (HW) multiplication
;                        subroutines, and stores the result from each in an array.
; Input: Integer b in the range [-8, -1] U [2, 8] per assignment requirement
; Output: Two integer arrays containing the first five powers of b
; Author: David Thornton (dht0002@uah.edu)
; Lab Section: 2
; Date: September 30, 2020
;-------------------------------------------------------------------------------
            .cdecls C,LIST,"msp430.h"       ; Include device header file
;-------------------------------------------------------------------------------
            .def    RESET                   ; Export program entry-point to
                                            ; make it
known to linker.
            .ref     calc_power
;-------------------------------------------------------------------------------
;           Memory allocation
            .bss     swarr, 10                              ; 10 bytes for software
array
            .bss     hwarr, 10                              ; 10 bytes for hardware
array
            .data                                           ; Declare variable in
a data segment
b:          .int     8                                      ; intput into the
code
result:     .int     1                                      ; result = 1 at start of code
;-------------------------------------------------------------------------------
            .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   #__STACK_END, SP          ; Initialize stackpointer
StopWDT     mov   #WDTPW|WDTHOLD, &WDTCTL   ; Stop watchdog timer
;-------------------------------------------------------------------------------
; Main loop here
```

```
;-------------------------------------------------------------------------------
main:
                        push    #swarr                          ; push the address of the
software array to the stack
                        push    #hwarr                          ; push the address of the
hardware array to the stack
                        push    result                          ; push result to the stack
                        mov     b,      R4                                      ; put b into
R4
                        mov     #5,     R8                                      ; counter for
number of powers
                        call    #calc_power                     ; go to calc_power to do
the operations
                        jmp $                                           ; infinite loop

lend:
                        nop

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

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

Appendix 2: Lab_5_calc_power.asm

```
;-------------------------------------------------------------------------------
; File: Lab_5_calc_power.asm
; Description: This subroutine calls the SW and HW subroutines
; Input: None
; Output: None
; Author: David Thornton (dht0002@uah.edu)
; Lab Section: 2
; Date: September 30, 2020
;-------------------------------------------------------------------------------
                        .cdecls C,LIST,"msp430.h"               ; Include device header file
                        .def    calc_power
                        .ref    HW_Mult
                        .ref    SW_Mult
```

```
            .text

calc_power:
            mov 6(SP), R5                         ; SW array address
            mov 4(SP), R6                         ; HW array address
            mov 2(SP), R7                         ; result variable
            mov #1, R12                     ; B, not b
            jmp SW_Loop

SW_Loop:
            call #SW_Mult                   ; call subroutine
            incd R5                                     ; go to the
next array index
            dec R8                                ; counter--
            jnz SW_Loop                           ; jump if the counter
is not zero
            mov #5, R8                            ; reset counter for
the HW loop

HW_Loop:
            call #HW_Mult                   ; call subroutine
            incd R6                                     ; go to the
next array index
            dec R8                                ; counter--
            jnz HW_Loop                           ; jump if the counter
is not zero

lend    reti                                ; return from subroutine
            end
```

Appendix 3: Lab_5_SW_Mult.asm

```
;-------------------------------------------------------------------------------
; File: Lab_5_SW_Mult.asm
; Description: This subroutine calculates the first five powers of B using
;                    a shift and add multiplication algorithm.
; Input:
; Output:
; Author: David Thornton (dht0002@uah.edu)
; Lab Section: 2
; Date: September 30, 2020
;-------------------------------------------------------------------------------
            .cdecls C,LIST,"msp430.h"      ; Include device header file
            .def SW_Mult
            .text
```

```
SW_Mult:
                mov #16, R9                     ; max bit counter per
assignment
                mov #0, R10                     ; clear result variable (C)
                mov R4, R11                     ; copy b to A
                jmp check

check:
                mov R12, R13                        ; temp so we don't
overwrite B
                and #0x01, R13                      ; bitwise and temp
(B) and 0x01
                cmp #0x01, R13                      ; compare LSB of
temp to 1
                jne skip                            ; if LSB of B is not
1, jump to skip
                add R11, R10                    ; add A to C
                jmp skip                            ; jump to skip
subroutine

skip:
                rrc R12                                 ; rotate right
B
                rla R11                             ; rotate left A
                dec R9                                  ; bit
counter--
                jnz check                           ; jump to check if bit
counter >0
                mov R12, R13                    ; temp so we don't
overwrite B
                and #0x10, R13                      ; bitwise and temp
(B) and 0x10
                cmp #0x10, R13                      ; compare MSB of
temp to 1
                jeq neg                                 ; jump if
temp is negative
                jmp qend                            ; else jump to qend

neg:
                sub R11, R10                    ; subtract A from C
                jmp qend                            ; jump to qend

qend:
                mov R10, 0(R5)                      ; put the result (C)
into the array
                mov R10, R12                        ; put the result in B
```

```
for next loop
                  reti                                          ; return from
subroutine
                  end
```

Appendix 4: Lab_5_HW_Mult.asm

```
;-------------------------------------------------------------------------------
; File: Lab_5_HW_Mult.asm
; Description: This subroutine uses a hardware multiplier to multiply numbers
; Input: Integers b, result
; Output: Integer product of b and result
; Author: David Thornton (dht0002@uah.edu)
; Lab Section: 2
; Date: September 30, 2020
;-------------------------------------------------------------------------------
                  .cdecls C,LIST,"msp430.h"     ; Include device header file
                  .def HW_Mult
                  .text

HW_Mult:
                  mov         R4, &MPY                  ; move b into multiplication
register
                  mov         R7, &OP2                  ; move result to general
purpose operator
                  nop
                  nop
                  nop
                  mov         RESLO, R7                 ; move the product to R7
(result)
                  mov         R7, 0(R6)                 ; move result into the HW
array
                  reti                                  ; return from subroutine
                  end
```

Appendix 5: calc_power table

Subroutine	Instruction	Number of Cycles	Number of Executions	Total
calc_power:	mov 6(SP), R5	3	1	3
calc_power:	mov 4(SP), R6	3	1	3

calc_power:	mov 2(SP), R7	3	1	3
calc_power:	mov #1, R12	2	1	2
calc_power:	jmp SW_Loop	2	1	2
SW_Loop:	call #SW_Mult	6	5	30
SW_Loop:	incd R5	1	5	5
SW_Loop:	dec R8	1	5	5
SW_Loop:	jnz SW_Loop	2	5	10
SW_Loop:	mov #5, R8	1	1	1
HW_Loop:	call #HW_Mult	6	5	30
HW_Loop:	incd R6	1	5	5
HW_Loop:	dec R8	1	5	5
HW_Loop:	jnz HW_Loop	2	5	10
lend	reti	5	1	5
Total CC for calc_power:				**119**

Appendix 6: SW_Mult table

Subroutine	Instruction	Number of Cycles	Number of Executions	Total
SW_Mult:	mov #16, R9	2	1	2
SW_Mult:	mov #0, R10	2	1	2
SW_Mult:	mov R4, R11	1	1	1
SW_Mult:	jmp check	2	1	2
check:	mov R12, R13	1	16	16
check:	and #0x01, R13	2	16	32
check:	cmp #0x01, R13	2	16	32
check:	jne skip	2	1	2
check:	add R11, R10	1	1	1
check:	jmp skip	2	1	2
skip:	rrc R12	1	16	16
skip:	rla R11	1	16	16
skip:	dec R9 ; counter	1	16	16
skip:	jnz check	2	16	32
skip:	mov R12, R13	1	12	12
skip:	and #0x10, R13	2	12	24

skip:	cmp #0x10, R13	2	12	24
skip:	jeq neg	2	1	2
skip:	jmp end	2	1	2
neg:	sub R11, R10	1	1	1
neg:	jmp end	2	1	2
end:	mov R10, 0(R5)	4	1	4
end:	mov R10, R12	1	1	1
end:	ret	5	1	5
Average Total CC for SW_Mult:				**249**

Appendix 7: HW_Mult table

Subroutine	Instruction	Number of Cycles	Number of Executions	Total
SW_Mult:	mov #16, R9	2	1	2
SW_Mult:	mov #0, R10	2	1	2
SW_Mult:	mov R4, R11	1	1	1
SW_Mult:	jmp check	2	1	2
check:	mov R12, R13	1	16	16
check:	and #0x01, R13	2	16	32
check:	cmp #0x01, R13	2	16	32
check:	jne skip	2	1	2
check:	add R11, R10	1	1	1
check:	jmp skip	2	1	2
skip:	rrc R12	1	16	16
skip:	rla R11	1	16	16
skip:	dec R9 ; counter	1	16	16
skip:	jnz check	2	16	32
skip:	mov R12, R13	1	12	12
skip:	and #0x10, R13	2	12	24
skip:	cmp #0x10, R13	2	12	24
skip:	jeq neg	2	1	2
skip:	jmp end	2	1	2
neg:	sub R11, R10	1	1	1
neg:	jmp end	2	1	2

end:	mov R10, 0(R5)	4	1	4
end:	mov R10, R12	1	1	1
end:	ret	5	1	5
Average Total CC for SW_Mult:				**249**