CPE 325: Embedded Systems Laboratory

Lab05

Subroutines, Passing Parameters, and Hardware Multiplier

Submitted by: Esther Shore

Date of Experiment: June 14, 2023

Report Deadline: June 21, 2023

Demonstration Deadline: June 26, 2023

**Introduction**

This lab introduces subroutines in MSP430 assembly programming in addition to passing parameters to those subroutines. Three different methods of passing parameters are examined, particularly focusing on utilizing the stack. Hardware and software methods of multiplication are used and compared in terms of performance. Reading and writing from memory is also practiced.

**Theory Topics**

1. Subroutines

Subroutines are modules of code that are written to be reusable blocks that improve organization and flow of a program to perform a specific function. They are reached by a CALL instruction. At the end of a subroutine a RET instruction is used to return to the program that called it. Within a subroutine, registers and a stack can also be used to improve efficiency of data management. Examples of subroutines are HW\_linear and SW\_linear, which each perform the subtask of writing an output array to memory using a specific technique.

1. Passing parameters

Three different ways data can be input to a subroutine are through registers, stack by value, and stack by reference. For passing parameters through registers, values are loaded into registers in the main program and accessed through those same registers within the subroutine. Stack by value involves pushing actual parameter values to the stack and accessing them in reference to the stack pointer within the subroutine. In contrast, stack by reference involves pushing the address of a parameter.

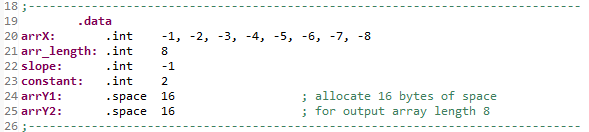
**Program 1**

***Program Description:***

This program involves evaluating the equation Y = mX + C for a given integer array X and assuming m and C are integer constants. The inputs are written to memory in the main driver program and passed into HW\_linear and SW\_linear subroutines through a stack. The HW\_linear subroutine utilizes the hardware multiplier on the MSP430 whereas the SW\_linear subroutine utilizes the shift-and-add multiplication algorithm. The resultant array is written to memory through each subroutine.

***Program Output:***

Input:



Output:



arrY1 begins at 0x2416, arrY2 begins at 0x2426

Assuming the clock frequency for the MSP430 is 1MHz, that means it goes through 1,000,000 clock cycles per second.

HW\_linear:

215 clock cycles

21 clock cycles per element => 1000000/21 = 47619 elements per second

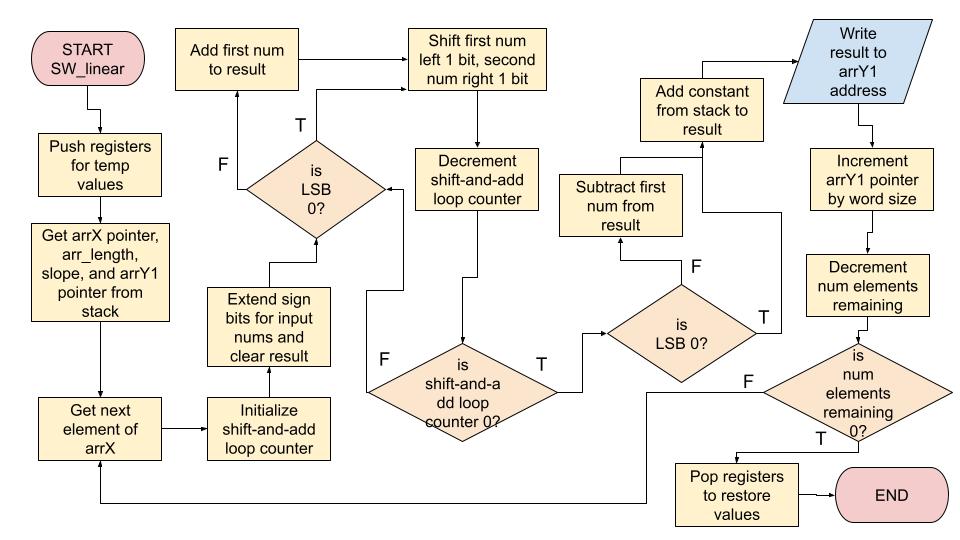
SW\_linear:

791 clock cycles total

92 clock cycles per element => 100000/92 = 10869 elements per second

Referring to the timing measurements gathered for each function, the hardware implementation appears to be significantly more efficient than the software implementation. This is likely due to the fact that the hardware implementation is utilizing the dedicated hardware on the MSP430 that is designed and optimized for multiplication. There were also less instructions required for hardware multiplication compared to its software counterpart.

***Program Flowchart:***



**Figure 1:** SW\_linear Subroutine Using Shift-and-Add Multiplication Flowchart

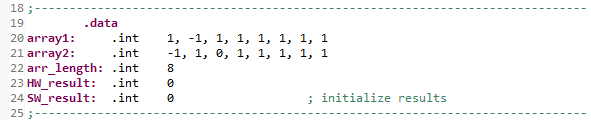
**Program 2 (Bonus)**

***Program Description:***

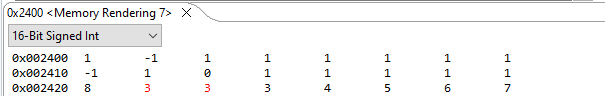
This program takes two input arrays of integers and calculates the dot product using software and hardware, each in their own subroutine. The inputs are pushed to the stack and accessed in reference to the stack pointer within the subroutines. For the software subroutine, the shift-and-add multiplication algorithm was used and for the hardware subroutine, the hardware multiplier was used. Both dot product results are then written to memory based on the addresses passed through the stack. Since the dot product result is a 16-bit integer, the elements in the input arrays must be limited to 8-bits.

***Program Output:***

Input:



Output:



HW\_result is located at 0x2422, SW\_result is located at 0x2424

**Conclusion**

I did not encounter any major issues in this lab although writing in assembly using a stack definitely posed a bit of a learning curve at first. I learned a lot in this lab and became much more comfortable with MSP430 assembly programming as well as interpreting instructions following a stack as it is built and collapsed.

***Appendix:***

**Table 1:** Program 1 Source Code

| ; ------------------------------------------------------------------------------  ; File: Lab05\_P1.asm  ; Function: Write linear Y coordinate array to memory  ; Description: This program performs multiplication of each element in X array  ; with slope using hardware then adds constant, then repeats  ; using software  ; Input: Pointers to input and output arrays, array length on stack  ; Output: Output array written to memory  ; Author(s): Esther Shore  ; Date: June 21, 2023  ; ------------------------------------------------------------------------------  .cdecls C, LIST, "msp430.h" ; Include device header file  ;-------------------------------------------------------------------------------  .def RESET ; Export program entry-point to  ; make it known to linker.  .ref HW\_linear  .ref SW\_linear  ;-------------------------------------------------------------------------------  .data  arrX: .int -1, -2, -3, -4, -5, -6, -7, -8  arr\_length: .int 8  slope: .int -1  constant: .int 2  arrY1: .space 16  arrY2: .space 16 ; allocate 16 bytes of space for arr len 8  ;-------------------------------------------------------------------------------  .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 stack pointer  mov.w #WDTPW|WDTHOLD,&WDTCTL ; Stop watchdog timer  ;-------------------------------------------------------------------------------  ; Main loop here  ;-------------------------------------------------------------------------------  main:  push #arrX ; push the starting address of arrX  push arr\_length ; push the number of elements  push slope ; push slope  push constant ; push constant (y-intercept)  push #arrY1 ; push starting address of arrY1  call #HW\_linear ; call HW\_linear subroutine  add #2, SP ; decrement SP by one word  push #arrY2 ; push starting address of arrY2  call #SW\_linear ; call SW\_linear subroutine  add #10, SP ; collapse stack  jmp $ ; infinite loop  nop  ;-------------------------------------------------------------------------------  ; Stack Pointer definition  ;-------------------------------------------------------------------------------  .global \_\_STACK\_END  .sect .stack  ;-------------------------------------------------------------------------------  ; Interrupt Vectors  ;-------------------------------------------------------------------------------  .sect ".reset" ; MSP430 RESET Vector  .short RESET  .end |
| --- |

**Table 2:** HW\_linear Code

| ; ------------------------------------------------------------------------------  ; File: HW\_linear.asm  ; Function: Write Y array to memory  ; Description: This program performs multiplication of each element in X array  ; with slope using hardware then adds constant  ; Input: Pointers to input and output arrays, array length on stack  ; Output: Output array written to memory  ; Author(s): Esther Shore  ; Date: June 21, 2023  ; ------------------------------------------------------------------------------  .cdecls C, LIST, "msp430.h" ; Include device header file  .def HW\_linear  ;-------------------------------------------------------------------------------  .text ; Assemble into program memory.  ;-------------------------------------------------------------------------------  HW\_linear:  push R4  push R5  push R6  push R7  push R8 ; push registers to stack  mov.w 20(SP), R4 ; arrX pointer on stack, R4  mov.w 18(SP), R5 ; arr\_length on stack, R5  mov.w 16(SP), &MPY ; slope on stack, second num to multiply  mov.w 12(SP), R8 ; arrY1 pointer on stack, R8  lnext: mov.w @R4+, R6 ; move element of array to R6 then increment  mov.w R6, &OP2 ; R6, first num to multiply  nop  nop  nop ; three clock cycles required for 16x16 hardware multiplication  mov.w RESLO, R7 ; move result to R7  add.w 14(SP), R7 ; add constant on stack to R7  mov.w R7, 0(R8) ; write result to arrY1 address  add.w #2, R8 ; increment arrY1 pointer by word size  dec.w R5 ; decrement arr\_length for loop control  jnz lnext ; if arr\_length not zero, get next element  pop R8 ; pop registers off stack  pop R7  pop R6  pop R5  pop R4  ret  .end |
| --- |

**Table 3:** SW\_linear Source Code

| ; ------------------------------------------------------------------------------  ; File: SW\_linear.asm  ; Function: Write Y array to memory  ; Description: This program performs multiplication of each element in X array  ; with slope using software then adds constant  ; Input: Pointers to input and output arrays, array length on stack  ; Output: Output array written to memory  ; Author(s): Esther Shore  ; Date: June 21, 2023  ; ------------------------------------------------------------------------------  .cdecls C, LIST, "msp430.h" ; Include device header file  .def SW\_linear  ;-------------------------------------------------------------------------------  .text ; Assemble into program memory.  ;-------------------------------------------------------------------------------  SW\_linear:  push R4  push R5  push R6  push R7  push R8  push R9  push R10 ; push registers to stack  mov.w 24(SP), R4 ; arrX pointer on stack, R4  mov.w 22(SP), R9 ; arr\_length on stack, R9  mov.w 20(SP), R5 ; slope on stack, R5  mov.w 16(SP), R8 ; arrY2 pointer on stack, R8  lnext: mov.w @R4+, R6 ; move element of array to R6 then increment  mov.w #8, R10 ; constant 8 bits for size of inputs for 16 bit result  sxt.w R6 ; extend sign bit for R6  sxt.w R5 ; extend sign bit for R5  clr.w R7 ; clear result register R7  loop: bit.w #1, R5 ; bitwise AND to check if LSB is 1  jz cont ; if LSB is 0, then jump to cont  add.w R6, R7 ; if LSB is 1, add  cont: add.w R6, R6 ; left shift by 1 bit same as multiply by 2  rrc.w R5 ; right shift by 1 bit  dec.w R10 ; decrement bit num for loop control  jnz loop ; while not zero, continue to loop  bit.w #1, R5 ; check if LSB is 1  jz endl ; if LSB is 0, jump to endl  sub.w R6, R7 ; if LSB is 1, subtract R7 = R7 - R6  endl: add.w 18(SP), R7 ; add constant on stack to R7  mov.w R7, 0(R8) ; write result to arrY1 address  add.w #2, R8 ; increment arrY1 pointer by word size  dec.w R9 ; decrement arr\_length for loop control  jnz lnext ; if arr\_length not zero, get next element  pop R10 ; pop registers off stack  pop R9  pop R8  pop R7  pop R6  pop R5  pop R4  ret  .end |
| --- |

**Table 4:** Program 2 Source Code

| ; ------------------------------------------------------------------------------  ; File: Lab05\_P2.asm  ; Function: Write dot product of two arrays to memory  ; Description: This program performs multiplication of each element in two  ; array using hardware then adds to result, repeats with software  ; Input: Pointers to input arrays, array length, and pointer to result  ; in memory on stack  ; Output: Dot product result written to memory  ; Author(s): Esther Shore  ; Date: June 21, 2023  ; ------------------------------------------------------------------------------  .cdecls C, LIST, "msp430.h" ; Include device header file  ;-------------------------------------------------------------------------------  .def RESET ; Export program entry-point to  ; make it known to linker.  .ref HW\_dotprod  .ref SW\_dotprod  ;-------------------------------------------------------------------------------  .data  array1: .int 1, -1, 1, 1, 1, 1, 1, 1  array2: .int -1, 1, 0, 1, 1, 1, 1, 1  arr\_length: .int 8  HW\_result: .int 0  SW\_result: .int 0 ; initialize results  ;-------------------------------------------------------------------------------  .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 stack pointer  mov.w #WDTPW|WDTHOLD,&WDTCTL ; Stop watchdog timer  ;-------------------------------------------------------------------------------  ; Main loop here  ;-------------------------------------------------------------------------------  main:  push #array1 ; push starting address of array1  push #array2 ; push starting address of array2  push arr\_length ; push the number of elements  push #HW\_result ; push address of HW\_result  call #HW\_dotprod ; call HW\_dotprod subroutine  add #2, SP ; decrement SP by one word  push #SW\_result ; push address of SW\_result  call #SW\_dotprod ; call SW\_dotprod subroutine  add #8, SP ; collapse stack  jmp $ ; infinite loop  nop  ;-------------------------------------------------------------------------------  ; Stack Pointer definition  ;-------------------------------------------------------------------------------  .global \_\_STACK\_END  .sect .stack  ;-------------------------------------------------------------------------------  ; Interrupt Vectors  ;-------------------------------------------------------------------------------  .sect ".reset" ; MSP430 RESET Vector  .short RESET  .end |
| --- |

**Table 5:** HW\_dotprod Source Code

| ; File: HW\_dotprod.asm  ; Function: Calculate dot product  ; Description: This program performs multiplication of each element in two  ; array using hardware then adds to result  ; Input: Pointers to input arrays, array length, and pointer to result  ; in memory on stack  ; Output: Dot product result written to memory  ; Author(s): Esther Shore  ; Date: June 21, 2023  ; ------------------------------------------------------------------------------  .cdecls C, LIST, "msp430.h" ; Include device header file  .def HW\_dotprod  ;-------------------------------------------------------------------------------  .text ; Assemble into program memory.  ;-------------------------------------------------------------------------------  HW\_dotprod:  push R4  push R5  push R6  push R7  push R8  push R9 ; push registers to stack  mov.w 20(SP), R4 ; array1 pointer on stack, R4  mov.w 18(SP), R5 ; array2 pointer on stack, R5  mov.w 16(SP), R6 ; arr\_length on stack, R6  mov.w 14(SP), R7 ; HW\_result address on stack, R7  clr.w R9 ; clear results register R9  lnext: mov.w @R4+, &MPY ; move element of array1 to R6 then increment  mov.w @R5+, &OP2 ; move element of array2 to R7 then increment  nop  nop  nop ; three clock cycles required for 16x16 hardware multiplication  mov.w RESLO, R8 ; product to R8  add.w R8, R9 ; add product to result R9  mov.w R9, 0(R7) ; write result to HW\_result address  dec.w R6 ; decrement arr\_length for loop control  jnz lnext ; if arr\_length not zero, get next element  pop R9 ; pop registers off stack  pop R8  pop R7  pop R6  pop R5  pop R4  ret  .end |
| --- |

**Table 6:** SW\_dotprod Source Code

| ; ------------------------------------------------------------------------------  ; File: SW\_dotprod.asm  ; Function: Calculate dot product  ; Description: This program performs multiplication of each element in two  ; array using software then adds to result  ; Input: Pointers to input arrays, array length, and pointer to result  ; in memory on stack  ; Output: Dot product result written to memory  ; Author(s): Esther Shore  ; Date: June 21, 2023  ; ------------------------------------------------------------------------------  .cdecls C, LIST, "msp430.h" ; Include device header file  .def SW\_dotprod  ;-------------------------------------------------------------------------------  .text ; Assemble into program memory.  ;-------------------------------------------------------------------------------  SW\_dotprod:  push R4  push R5  push R6  push R7  push R8  push R9  push R10  push R11 ; push registers to stack  mov.w 24(SP), R4 ; array1 pointer on stack, R4  mov.w 22(SP), R10 ; array2 pointer on stack, R5  mov.w 20(SP), R9 ; arr\_length on stack, R9  mov.w 18(SP), R8 ; SW\_result address on stack, R8  lnext: mov.w @R4+, R6 ; move element of array1 to R6 then increment  mov.w @R10+, R5 ; move element of array2 to R5 then increment  mov.w #8, R11 ; inputs are 8 bits for 16 bit result  sxt.w R6 ; extend sign bit for R6  sxt.w R5 ; extend sign bit for R5  clr.w R7 ; clear result register R7  loop: bit.w #1, R5 ; bitwise AND to check if LSB is 1  jz cont ; if lSB is 0, then jump to cont  add.w R6, R7 ; if LSB is 1, add  cont: add.w R6, R6 ; left shift by 1 bit same as multiply by 2  rrc.w R5 ; right shift by 1 bit  dec.w R11 ; decrement bit num for loop control  jnz loop ; while not zero, continue to loop  bit.w #1, R5 ; check if LSB is 1  jz endl ; if LSB is 0, jump to endl  sub.w R6, R7 ; if LSB is 1, subtract R7 = R7 - R6  endl: add.w R7, 0(R8) ; add result to SW\_result address  dec.w R9 ; decrement arr\_length for loop control  jnz lnext ; if arr\_length not zero, get next element  pop R11 ; pop registers off stack  pop R10  pop R9  pop R8  pop R7  pop R6  pop R5  pop R4  ret  .end |
| --- |