CPE 323
Intro to Embedded Computer Systems
Assembly Language Programming (Subroutines)

Aleksandar Milenkovic

milenka@uah.edu

.

Admin

The Case for Subroutines: An Example

- Problem
  - Sum up elements of two integer arrays
  - Display results on P2OUT&P1OUT and P4OUT&P3OUT
- Example
  - arr1 .int 1, 2, 3, 4, 1, 2, 3, 4 ; the first array
  - arr2 .int 1, 1, 1, 1, -1, -1, -1 ; the second array
  - Results
    - P2OUT&P1OUT=0x000A, P4OUT&P3OUT=0x0001
- Approach
  - Input numbers: arrays
  - Main program (no subroutines): initialization, program loops

Sum Up Two Integer Arrays (ver1)

```
;-------------------------------------------------------------------------------
; File        : Lab5_D1.asm (CPE 325 Lab5 Demo code)
; Function    : Finds a sum of two integer arrays
; Description: The program initializes ports,
;              sums up elements of two integer arrays and
;              display sums on parallel ports
; Input       : The input arrays are signed 16-bit integers in arr1 and arr2
; Output      : P1OUT&P2OU displays sum of arr1, P3OUT&P4OUT displays sum of arr2
; Author      : A. Milenkovic, milenkovic@computer.org
; Date        : September 14, 2008
;-------------------------------------------------------------------------------
            .cdecls C,LIST,"msp430.h"       ; Include device header file

;-------------------------------------------------------------------------------
            .def    RESET                   ; Export program entry-point to
                                            ; make it known to linker.
;-------------------------------------------------------------------------------
            .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
StopWDT:    mov.w   #WDTPW|WDTHOLD,&WDTCTL  ; Stop watchdog timer
```

Sum up two integer arrays (ver1)

```
;-------------------------------------------------------------------------------
; Main code here
;-------------------------------------------------------------------------------
main:
```

Sum up two integer arrays (ver1)

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

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

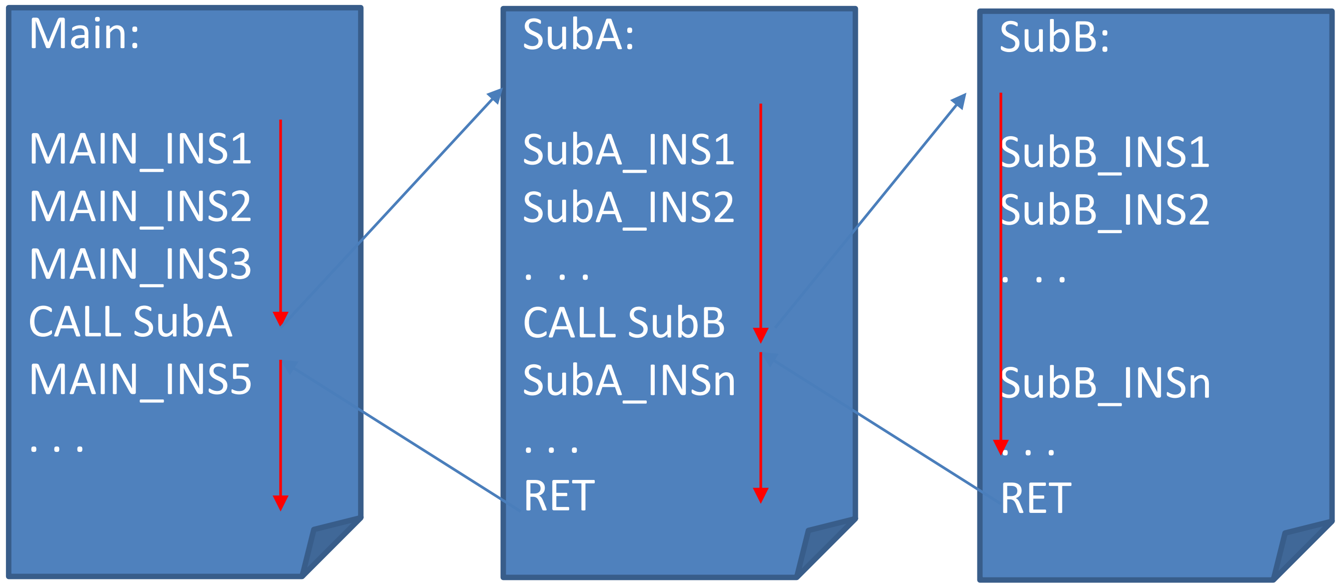
Subroutines

- A particular sub-task is performed many times on different data values
- Frequently used subtasks are known as subroutines
- Subroutines: How do they work?
  - Only one copy of the instructions that constitute the subroutine is placed in memory
  - Any program that requires the use of the subroutine simply branches to its starting location in memory
  - Upon completion of the task in the subroutine, the execution continues at the next instruction in the calling program

Subroutines (cont’d)

- CALL instruction:
  perform the branch to subroutines
  - SP <= SP – 2 ; allocate a word on the stack for return address
  - M[SP] <= PC ; push the return address (current PC) onto the stack
  - PC <= TargetAddress ; the starting address of the subroutine is moved into PC
- RET instruction:
  the last instruction in the subroutine
  - PC <= M[SP] ; pop the return address from the stack
  - SP <= SP + 2 ; release the stack space

Subroutine Nesting



```
Main:

MAIN_INS1
MAIN_INS2
MAIN_INS3
CALL SubA
MAIN_INS5
. . .
```

```
SubA:

SubA_INS1
SubA_INS2
. . .
CALL SubB
SubA_INSn
. . .
RET
```

```
SubB:

SubB_INS1
SubB_INS2
. . .

SubB_INSn
. . .
RET
```

Mechanisms for Passing Parameters

- Through registers
- Through stack
  - By value
    - Actual parameter is transferred
    - If the parameter is modified by the subroutine, the “new value” does not affect the “old value”
  - By reference
    - The address of the parameter is passed
    - There is only one copy of parameter
    - If parameter is modified, it is modified globally

Subroutine: SUMA_RP

- Subroutine for summing up elements of an integer array
- Passing parameters through registers
  - `R12 - starting address of the array`
  - `R13 - array length`
  - `R14 - display id`
    `(0 for P2&P1, 1 for P4&P3)`

Subroutine: SUMA_RP

```
;-------------------------------------------------------------------------------
; File       : Lab5_D2_RP.asm (CPE 325 Lab5 Demo code)
; Function   : Finds a sum of an input integer array
; Description: suma_rp is a subroutine that sums elements of an integer array
; Input      : The input parameters are:
;                    R12 -- array starting address
;                    R13 -- the number of elements (>= 1)
;                    R14 -- display ID (0 for P1&P2 and 1 for P3&P4)
; Output     : No output
; Author     : A. Milenkovic, milenkovic@computer.org
; Date       : September 14, 2008
;------------------------------------------------------------------------------
            .cdecls C,LIST,"msp430.h"      ; Include device header file

            .def suma_rp

            .text
```

Subroutine: SUMA_RP

```
suma_rp:
            push.w  R7                ; save the register R7 on the stack
            clr.w   R7                ; clear register R7 (keeps the sum)
lnext:      add.w   @R12+, R7         ; add a new element
            dec.w   R13               ; decrement step counter
            jnz     lnext             ; jump if not finished
            bit.w   #1, R14           ; test display ID
            jnz     lp34              ; jump on lp34 if display ID=1
            mov.b   R7, P1OUT         ; display lower 8-bits of the sum on P1OUT
            swpb    R7                ; swap bytes
            mov.b   R7, P2OUT         ; display upper 8-bits of the sum on P2OUT
            jmp     lend              ; skip to end
lp34:       mov.b   R7, P3OUT         ; display lower 8-bits of the sum on P3OUT
            swpb    R7                ; swap bytes
            mov.b   R7, P4OUT         ; display upper 8-bits of the sum on P4OUT
lend:       pop     R7                ; restore R7
            ret                       ; return from subroutine

            .end
```

Main (ver2): Call suma_rp

```
;-------------------------------------------------------------------------------
; Main code here
;-------------------------------------------------------------------------------
main
```

Subroutine: SUMA_SP

- Subroutine for summing up elements of an integer array
- Passing parameters through the stack
  - The calling program prepares input parameters on the stack

Main (ver3): Call suma_sp (Pass Through Stack)

```
;-------------------------------------------------------------------------------
; Main code here
;-------------------------------------------------------------------------------
main:       bis.b   #0xFF,&P1DIR               ; configure P1.x as output
            bis.b   #0xFF,&P2DIR               ; configure P2.x as output
            bis.b   #0xFF,&P3DIR               ; configure P3.x as output
            bis.b   #0xFF,&P4DIR               ; configure P4.x as output

            push    #arr1                      ; push the address of arr1
            push    #8                         ; push the number of elements
            push    #0                         ; push display id
            call    #suma_sp
            add.w   #6,SP                      ; collapse the stack
            push    #arr2                      ; push the address of arr1
            push    #7                         ; push the number of elements
            push    #1                         ; push display id
            call    #suma_sp
            add.w   #6,SP                      ; collapse the stack

            jmp     $

arr1:       .int     1, 2, 3, 4, 1, 2, 3, 4    ; the first array
arr2:       .int     1, 1, 1, 1, -1, -1, -1    ; the second array
```

Address	Stack
0x0800	OTOS
0x07FE	#arr1
0x07FC	0008
0x07FA	0000
0x07F8	Ret. Addr.

Subroutine: SUMA_SP

```
;-------------------------------------------------------------------------------
; File       : Lab5_D3_SP.asm (CPE 325 Lab5 Demo code)
; Function   : Finds a sum of an input integer array
; Description: suma_sp is a subroutine that sums elements of an integer array
; Input      : The input parameters are on the stack pushed as follows:
;                 starting addrress of the array
;                 array length
;                 display id
; Output     : No output
; Author     : A. Milenkovic, milenkovic@computer.org
; Date       : September 14, 2008
;------------------------------------------------------------------------------
            .cdecls C,LIST,"msp430.h"       ; Include device header file

            .def    suma_sp

            .text
```

Subroutine: SUMA_SP (cont'd)

```
suma_sp:
                                            ; save the registers on the stack
        push     R7                         ; save R7, temporal sum
        push     R6                         ; save R6, array length
        push     R4                         ; save R5, pointer to array
        clr.w    R7                         ; clear R7
        mov.w    10(SP), R6                 ; retrieve array length
        mov.w    12(SP), R4                 ; retrieve starting address
lnext:  add.w    @R4+, R7                   ; add next element
        dec.w    R6                         ; decrement array length
        jnz      lnext                      ; repeat if not done
        mov.w    8(SP), R4                  ; get id from the stack
        bit.w    #1, R4                     ; test display id
        jnz      lp34                       ; jump to lp34 display id = 1
        mov.b    R7, P1OUT                       ; lower 8 bits of the sum
to P1OUT
        swpb     R7                         ; swap bytes
        mov.b    R7, P2OUT                      ; upper 8 bits of the sum to
P2OUT
        jmp      lend                       ; jump to lend
lp34:   mov.b    R7, P3OUT                  ; lower 8 bits of ths sum to P3OUT
        swpb     R7                         ; swap bytes
        mov.b    R7, P4OUT                  ; upper 8 bits of the sum to P4OUT
lend:   pop      R4                         ; restore R4
        pop      R6                         ; restore R6
        pop      R7                         ; restore R7
        ret                                 ; return

        .end
```

Address	Stack
0x0800	OTOS
0x07FE	#arr1
0x07FC	0008
0x07FA	0000
0x07F8	Ret. Addr.
0x07F6	(R7)
0x07F4	(R6)
0x07F2	(R4)

The Stack and Local Variables

- Subroutines often need local workspace
- We can use a fixed block of memory space – static allocation – but:
  - The code will not be relocatable
  - The code will not be reentrant
  - The code will not be able to be called recursively
- Better solution: dynamic allocation
  - Allocate all local variables on the stack
  - STACK FRAME = a block of memory allocated by a subroutine to be used for local variables
  - FRAME POINTER = an address register used to point to the stack frame

Subroutine: SUMA_SPSF

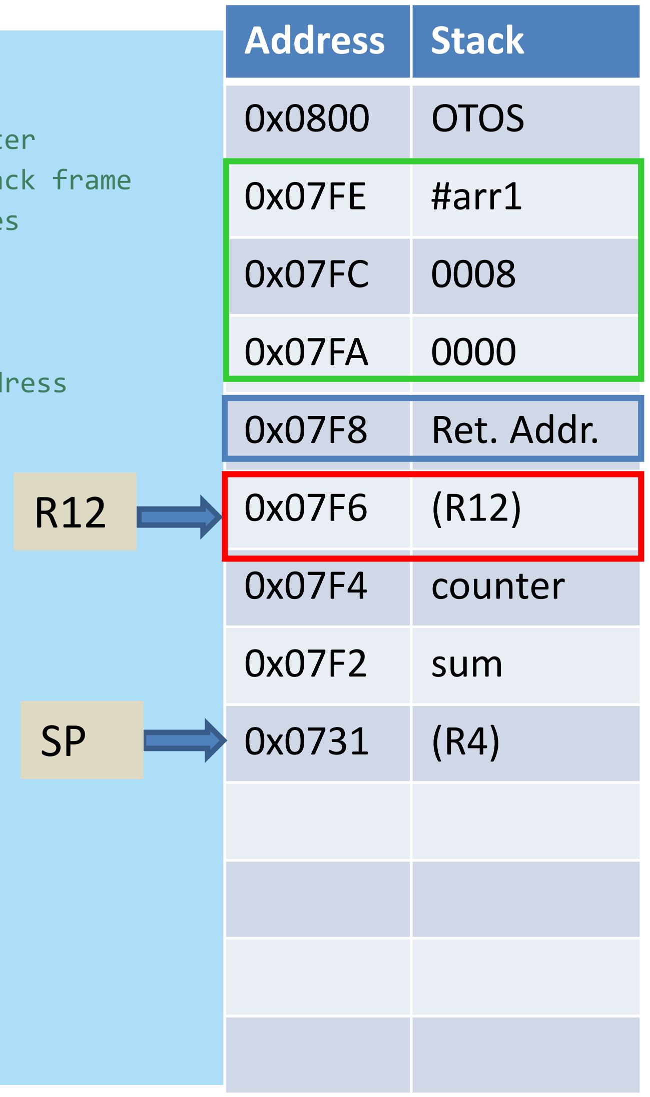
```
;-------------------------------------------------------------------------------
; File       : Lab5_D4_SPSF.asm (CPE 325 Lab5 Demo code)
; Function   : Finds a sum of an input integer array
; Description: suma_spsf is a subroutine that sums elements of an integer array.
;              The subroutine allocates local variables on the stack:
;                  counter (SFP+2)
;                  sum (SFP+4)
; Input      : The input parameters are on the stack pushed as follows:
;                  starting address of the array
;                  array length
;                  display id
; Output     : No output
; Author     : A. Milenkovic, milenkovic@computer.org
; Date       : September 14, 2008
;------------------------------------------------------------------------------
           .cdecls C,LIST,"msp430.h"       ; Include device header file

           .def    suma_spsf

            .text
```

Subroutine: SUMA_SPSF (cont’d)

```
suma_spsf:
        ; save the registers on the stack
        push    R12              ; save R12 - R12 is stack frame pointer
        mov.w   SP, R12          ; R12 points on the bottom of the stack frame
        sub.w   #4, SP           ; allocate 4 bytes for local variables
        push    R4               ; pointer register
        clr.w   -4(R12)          ; clear sum, sum=0
        mov.w    6(R12), -2(R12) ; get array length
        mov.w   8(R12), R4       ; R4 points to the array starting address
lnext:  add.w   @R4+, -4(R12)    ; add next element
        dec.w   -2(R12)          ; decrement counter
        jnz     lnext            ; repeat if not done
        bit.w   #1, 4(R12)       ; test display id
        jnz     lp34             ; jump to lp34 if display id = 1
        mov.b   -4(R12), P1OUT   ; lower 8 bits of the sum to P1OUT
        mov.b   -3(R12), P2OUT   ; upper 8 bits of the sume to P2OUT
        jmp     lend             ; skip to lend
lp34:   mov.b   -4(R12), P3OUT   ; lower 8 bits of the sum to P3OUT
        mov.b   -3(R12), P4OUT   ; upper 8 bits of the sume to P4OUT
lend:   pop     R4               ; restore R4
        add.w   #4, SP           ; collapse the stack frame
        pop     R12              ; restore stack frame pointer
        ret                      ; return
        .end
```



Address	Stack
0x0800	OTOS
0x07FE	#arr1
0x07FC	0008
0x07FA	0000
0x07F8	Ret. Addr.
0x07F6	(R12)
0x07F4	counter
0x07F2	sum
0x0731	(R4)