CPE 323: The MSP430 Assembly Language Programming

Aleksandar Milenkovic

Electrical and Computer Engineering
The University of Alabama in Huntsville

milenka@ece.uah.edu

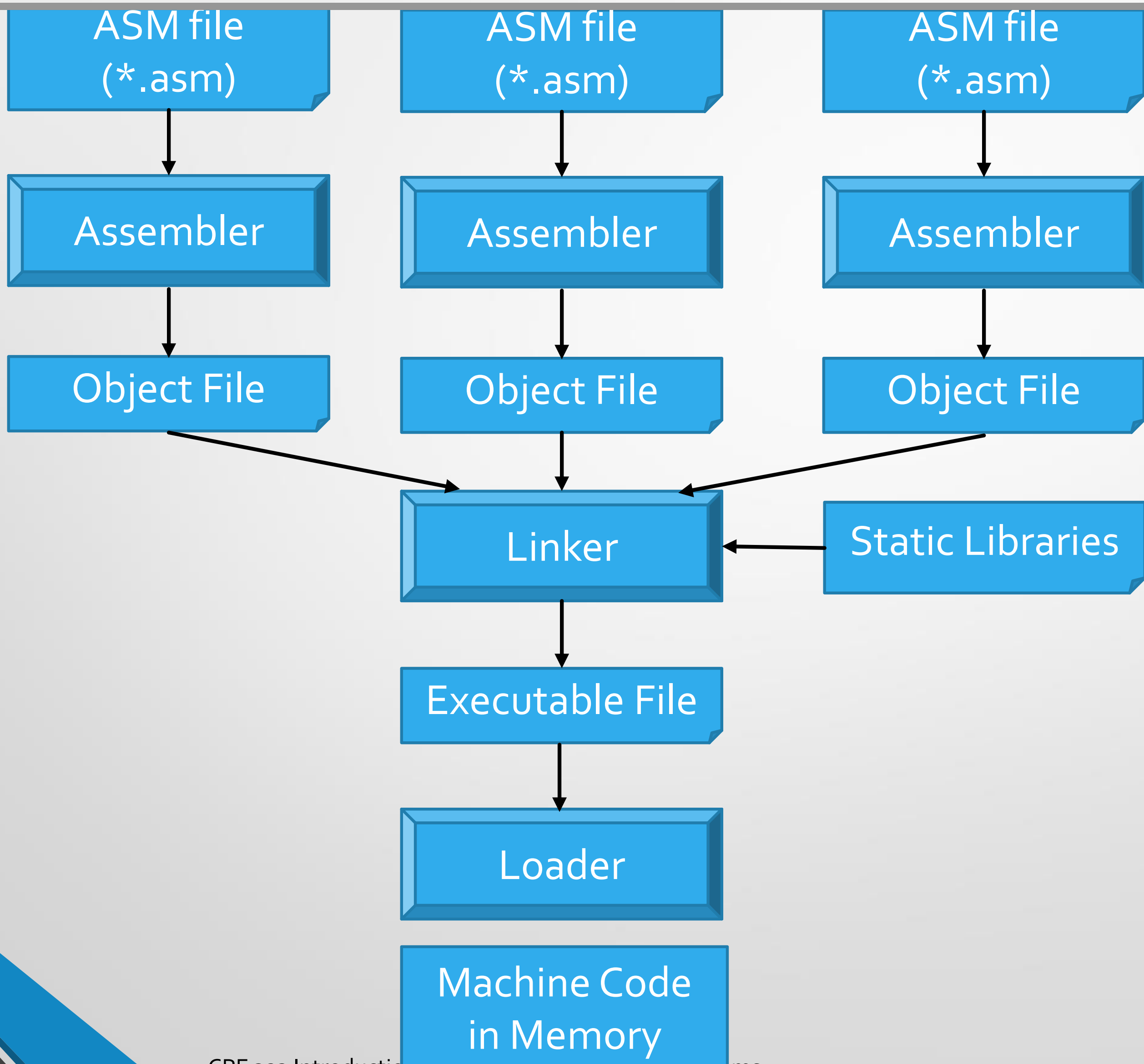
http://www.ece.uah.edu/~milenka

Outline

- Introduction
- Assembly language directives
- SUMI/SUMD
  - Adding two 32-bit numbers (decimal, integers)
- CountEs: Counting characters ‘E’
- Subroutines
  - CALL&RETURN
  - Subroutine Nesting
  - Passing parameters
  - Stack and Local Variables
- Performance

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

Assembly Programming Flow



THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

Assembly Directives

- Assembly language directives tell the assembler to
  - Set the data and program at particular addresses in address pace
  - Allocate space for constants and variables
  - Define synonyms
  - Include additional files
  - …
- Typical directives
  - Equate: assign a value to a symbol
  - Origin: set the current location pointer
  - Define space: allocate space in memory
  - Define constant: allocate space for and initialize constants
  - Include: loads another source file

LaCASA

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

ASM430 Section Control Directives

- CCStudio ASM430 has three predefined sections into which various parts of a program are assembled
  - .bss: Uninitialized data section
  - .data: Initialized data section
  - .text: Executable code section

Description	ASM430 (CCS)	A430 (IAR)
Reserve size bytes in the uninitialized sect.	.bss	-
Assemble into the initialized data section	.data	RSEG const
Assemble into a named initialized data sect.	.sect	RSEG
Assemble into the executable code	.text	RSEG code
Reserve space in a named (uninitialized) section	.usect	-
Align on byte boundary	.align 1	-
Align on word boundary	.align 2	EVEN

Examples

```
; IAR
          RSEG DAT16_N      ; switch to DATA segment
          EVEN              ; make sure it starts at even address
MyWord:   DS 2              ; allocate 2 bytes / 1 word
MyByte:   DS 1              ; allocate 1 byte

; CCS Assembler (Example #1)
MyWord:   .usect ".bss", 2, 2   ; allocate 1 word
MyByte:   .usect ".bss", 1      ; allocate 1 byte

; CCS Assembler (Example #2)
          .bss   MyWord,2,2 ; allocate 1 word
          .bss   MyByte,1   ; allocate 1 byte
```

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

Constant Initialization Directives

Description	ASM430 (CCS)	A430 (IAR)
Initialize one or more successive bytes or text strings	.byte or .string	DB
Initialize 32-bit IEEE floating-point	.float	DF
Initialize a variable-length field	.field	-
Reserve size bytes in the current location	.space	DS
Initialize one or more 16-bit integers	.word	DW
Initialize one or more 32-bit integers	.long	DL

LaCASA

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

Directives: Dealing with Constants

```
b1:         .byte   5          ; allocates a byte in memory and initialize it with 5
b2:         .byte   -122       ; allocates a byte with constant -122
b3:         .byte   10110111b  ; binary value of a constant
b4:         .byte   0xA0       ; hexadecimal value of a constant
b5:         .byte   123q       ; octal value of a constant
tf:         .equ 25
```

Word view of Memory

Label	Address	Memory[15:8]	Memory[7:0]
b1	0x3100	0x86	0x05
b3	0x3102	0xA0	0xB7
b5	0x3104	--	0x53

Byte view of Memory

Label	Address	Memory[7:0]
b1	0x3100	0x05
b2	0x3101	0x86
b3	0x3102	0xB7
b4	0x3103	0xA0
b5	0x3104	0x53

Directives: Dealing with Constants

```
...
w1:         .word   21          ; allocates a word constant in memory;

w2:         .word  -21
w3:         .word tf
dw1:        .long  100000       ; allocates a long word size constant in memory;
                                ; 100000 (0x0001_86A0)
dw2:        .long 0xFFFFFFEA
```

Label	Address	Memory[15:8]	Memory[7:0]
w1	0x3106	0x00	0x15
w2	0x3108	0xFF	0xEB
w3	0x310A	0x00	0x19
dw1	0x310C	0x86	0xA0
	0x310E	0x00	0x01
dw2	0x3110	0xFF	0xEA
	0x3112	0xFF	0xFF

Directives: Dealing with Constants

```
s1:        .byte 'A', 'B', 'C', 'D' ; allocates 4 bytes in memory with string ABCD
s2:        .byte "ABCD", ' ' ; allocates 5 bytes in memory with string ABCD + NULL
```

Label	Address	Memory[15:8]	Memory[7:0]
s1	0x3114	0x42	0x41
	0x3116	0x44	0x43
s2	0x3118	0x42	0x41
	0x311A	0x44	0x43
	0x311C	--	0x00
	0x311E		

Table of Symbols

- Created by the assembler (think about this as a table of synonyms)

Symbol	Value [hex]
b1	0x3100
b2	0x3101
b3	0x3102
b4	0x3103
b5	0x3104
tf	0x0019
w1	0x3106
w2	0x3108
w3	0x310A
dw1	0x310C
dw2	0x3110
s1	0x3114
s2	0x3118

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

Directives: Variables in RAM

```
.bss v1b,1,1        ; allocates a byte in memory, equivalent to DS 1
.bss v2b,1,1        ; allocates a byte in memory
.bss v3w,2,2        ; allocates a word of 2 bytes in memory
.bss v4b,8,2        ; allocates a buffer of 2 long words (8 bytes)
.bss vx,1,1
```

Label	Address	Memory[15:8]	Memory[7:0]
v1b	0x1100	--	--
v3w	0x1102	--	--
v4b	0x1104	--	--
	0x1106	--	--
	0x1108	--	--
	0x110A	--	--
vx	0x110C		

Symbol	Value [hex]
v1b	0x1100
v2b	0x1101
v3w	0x1102
v4b	0x1104
vx	0x110C

LaCASA

Decimal/Integer Addition of 32-bit Numbers

- Problem
  - Write an assembly program that finds a sum of two 32-bit numbers
    - Input numbers are decimal numbers (8-digit in length)
    - Input numbers are signed integers in two's complement
- Data:
  - lint1: `DC32 0x45678923`
  - lint2: `DC32 0x23456789`
  - Decimal sum: `0x69135712`
  - Integer sum: `0x68ac31ac`
- Approach
  - Input numbers: storage, placement in memory
  - Results: storage (ABSOLUTE ASSEMBLER)
  - Main program: initialization, program loops
  - Decimal addition, integer addition

Decimal/Integer Addition of 32-bit Numbers

```
;-------------------------------------------------------------------------------
; File        : LongIntAddition.asm
; Function    : Sums up two long integers represented in binary and BCD
; Description: Program demonstrates addition of two operands lint1 and lint2.
;               Operands are first interpreted as 32-bit decimal numbers and
;               and their sum is stored into lsumd;
;               Next, the operands are interpreted as 32-bit signed integers
;               in two's complement and their sum is stored into lsumi.
; Input       : Input integers are lint1 and lint2 (constants in flash)
; Output      : Results are stored in lsumd (decimal sum) and lsumi (int sum)
; Author      : A. Milenkovic, milenkovic@computer.org
; Date        : August 24, 2018
;-------------------------------------------------------------------------------
            .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.
;-------------------------------------------------------------------------------
```

Decimal/Integer Addition of 32-bit Numbers (cont'd)

```
lint1:.long 0x45678923

lint2:.long 0x23456789

;-------------------------------------------------------------------------------

;-------------------------------------------------------------------------------

lsumd:.usect ".bss", 4,2 ; allocate 4 bytes for decimal result

lsumi:.usect ".bss", 4,2 ; allocate 4 bytes for integer result

;-------------------------------------------------------------------------------

RESET:      mov.w   #__STACK_END,SP         ; Initialize stack pointer

StopWDT:    mov.w   #WDTPW|WDTHOLD,&WDTCTL  ; Stop watchdog timer

;-------------------------------------------------------------------------------
```

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

Decimal/Integer Addition of 32-bit Numbers (cont'd)

```
;-------------------------------------------------------------------------------
; Main code here
;-------------------------------------------------------------------------------
            clr.w   R2                      ; clear status register
            mov.w   lint1, R8               ; get lower 16 bits from lint1 to R8
            dadd.w  lint2, R8               ; decimal addition, R8 + lower 16-bit of lint2
            mov.w   R8, lsumd               ; store the result (lower 16-bit)
            mov.w   lint1+2, R8             ; get upper 16 bits of lint1 to R8
            dadd.w  lint2+2, R8             ; decimal addition
            mov.w   R8, lsumd+2             ; store the result (upper 16-bit)
            mov.w   lint1, R8               ; get lower 16 bite from lint1 to R8
            add.w   lint2, R8               ; integer addition
            mov.w   R8, lsumi               ; store the result (lower 16 bits)
            mov.w   lint1+2, R8             ; get upper 16 bits from lint1 to R8
            addc.w  lint2+2, R8             ; add upper words, plus carry
            mov.w   R8, lsumi+2             ; store upper 16 bits of the result

           jmp $                            ; jump to current location '$'
                                            ; (endless loop)
```


LaCASA

Decimal/Integer Addition of 32-bit Numbers (cont'd)

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

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

Version 2: Decimal/Integer Addition of 32-bit Numbers (cont'd)

```
; Decimal addition
            mov.w   #lint1, R4          ; pointer to lint1
            mov.w   #lsumd, R8          ; pointer to lsumd
            mov.w   #2, R5              ; R5 is a counter (32-bit=2x16-bit)
            clr.w   R10                 ; clear R10
lda:        mov.w   4(R4), R7           ; load lint2
            mov.w   R10, R2             ; load original SR
            dadd.w  @R4+, R7            ; decimal add lint1 (with carry)
            mov.w   R2, R10             ; backup R2 in R10
            mov.w   R7, 0(R8)           ; store result (@R8+0)
            add.w   #2, R8              ; update R8
            dec.w   R5                  ; decrement R5
            jnz     lda                 ; jump if not zero to lda
```

Version 2: Decimal/Integer Addition of 32-bit Numbers (cont'd)

```
; Integer addition
        mov.w   #lint1, R4          ; pointer to lint1
        mov.w   #lsumi, R8          ; pointer to lsumi
        mov.w   #2, R5              ; R5 is a counter (32-bit=2x16-bit)
        clr.w   R10                 ; clear R10
lia:    mov.w   4(R4), R7           ; load lint2
        mov.w   R10, R2             ; load original SR
        addc.w  @R4+, R7            ; decimal add lint1 (with carry)
        mov.w   R2, R10             ; backup R2 in R10
        mov.w   R7, 0(R8)           ; store result (@R8+0)
        add.w   #2, R8              ; update R8
        dec.w   R5                  ; decrement R5
        jnz     lia                 ; jump if not zero to lia

        jmp     $                   ; jump to current location '$'
                                    ; (endless loop)
```

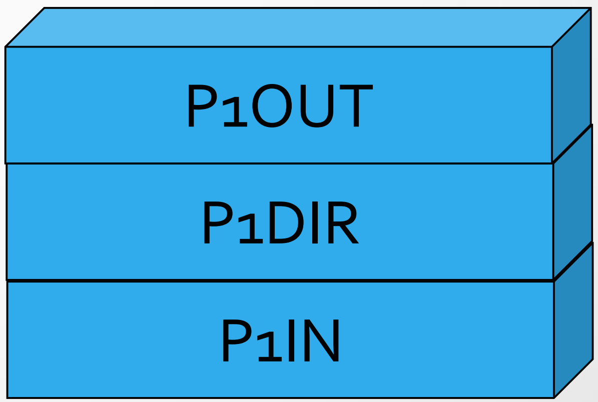
Count Characters ‘E’

- Problem
  - Write an assembly program that processes an input string to find the number of characters ‘E’ in the string
  - The number of characters is “displayed” on the port 1 of the MSP430
- Example:
  - mystr=“HELLO WORLD, I AM THE MSP430!”, ‘’
  - P1OUT=0x02
- Approach
  - Input string: storage, placement in memory
  - Main program: initialization, main program loop
  - Program loop: iterations, counter, loop exit
  - Output: control of ports

Programmer’s View of Parallel Ports

- Parallel ports: x=1,2,3,4,5, …
- Each can be configured as:
  - Input: PxDIR=0x00 (default)
  - Output: PxDIR=0xFF
- Writing to an output port:
  - PxOUT=x02
- Reading from an input port:
  - My_port=P1IN

Port Registers



THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

Count Characters ‘E’

```
;-------------------------------------------------------------------------------
; File       : Lab4_D1.asm (CPE 325 Lab4 Demo code)
; Function   : Counts the number of characters E in a given string
; Description: Program traverses an input array of characters
;              to detect a character 'E'; exits when a NULL is detected
; Input      : The input string is specified in myStr
; Output     : The port P1OUT displays the number of E's in the string
; Author     : A. Milenkovic, milenkovic@computer.org
; Date       : August 14, 2008
;-------------------------------------------------------------------------------
        .cdecls C,LIST,"msp430.h"       ; Include device header file

;-------------------------------------------------------------------------------
        .def    RESET                   ; Export program entry-point to
                                        ; make it known to linker.
myStr:  .string "HELLO WORLD, I AM THE MSP430!", ''
;-------------------------------------------------------------------------------
        .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
```


LaCASA

Count Characters ‘E’ (cont’d)

```
;-------------------------------------------------------------------------------
; Main loop here
;-------------------------------------------------------------------------------
main:   bis.b   #0FFh,&P1DIR            ; configure P1.x output
        mov.w   #myStr, R4              ; load the starting address of the string into R4
        clr.b   R5                      ; register R5 will serve as a counter
gnext:  mov.b   @R4+, R6                ; get a new character
        cmp     #0,R6                   ; is it a null character
        jeq     lend                    ; if yes, go to the end
        cmp.b   #'E',R6                 ; is it an 'E' character
        jne     gnext                   ; if not, go to the next
        inc.w   R5                      ; if yes, increment counter
        jmp     gnext                   ; go to the next character

lend:   mov.b   R5,&P1OUT               ; set all P1 pins (output)
        bis.w   #LPM4,SR                ; LPM4
        nop                             ; required only for Debugger

;-------------------------------------------------------------------------------
; Stack Pointer definition
;-------------------------------------------------------------------------------
        .global __STACK_END
        .sect   .stack
;-------------------------------------------------------------------------------
; Interrupt Vectors
;-------------------------------------------------------------------------------
         .sect   ".reset"               ; MSP430 RESET Vector
         .short  RESET
         .end
```

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
```

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

Sum up two integer arrays (ver1)

```
;-------------------------------------------------------------------------------
; 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
            ; load the starting address of the array1 into the register R4
            mov.w   #arr1, R4
            ; load the starting address of the array1 into the register R4
            mov.w   #arr2, R5
            ; Sum arr1 and display
            clr.w   R7                      ; Holds the sum
            mov.w       #8, R10                 ; number of elements in arr1
lnext1:     add.w       @R4+, R7                ; get next element
            dec.w   R10
            jnz     lnext1
            mov.b   R7, P1OUT               ; display sum of arr1
            swpb    R7
            mov.b   R7, P2OUT
```


LaCASA

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

Sum up two integer arrays (ver1)

```
 ; Sum arr2 and display
            clr.w   R7                      ; Holds the sum
            mov.w   #7, R10                 ; number of elements in arr2
lnext2:     add.w   @R5+, R7                ; get next element
            dec.w   R10
            jnz     lnext2
            mov.b   R7, P3OUT               ; display sum of arr1
            swpb    R7
            mov.b   R7, P4OUT
            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


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

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


LaCASA

Subroutines

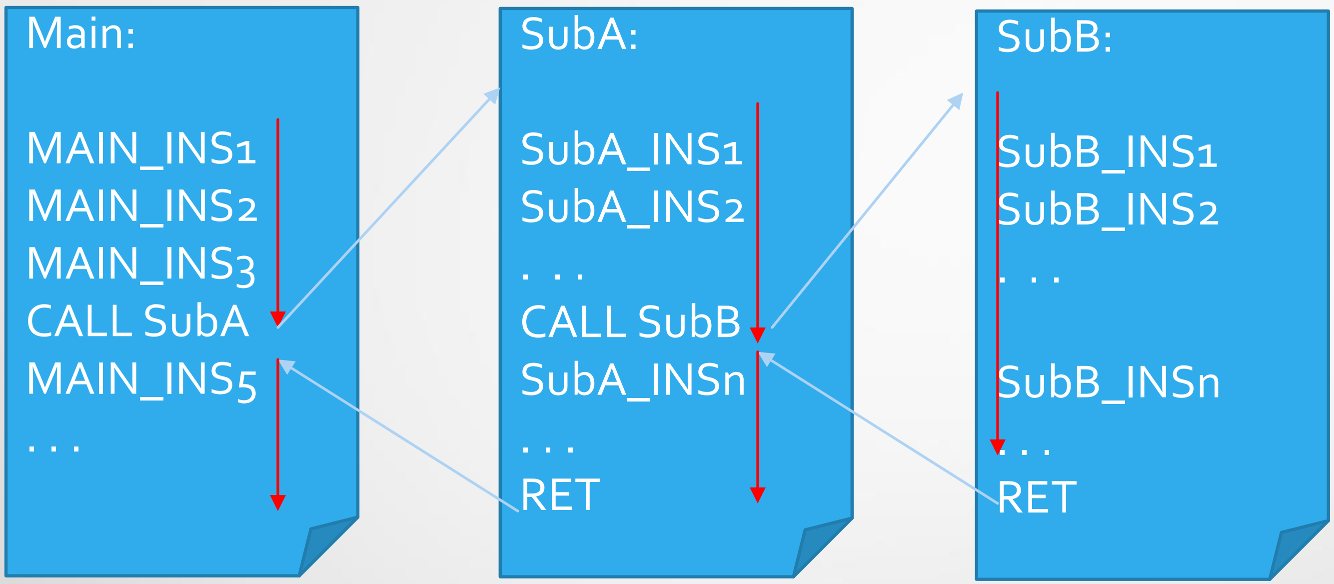
- A particular sub-task is performed many times on different data values
- Frequently used subtasks are known as subroutines
- Subroutines: How do they work?
  - Only one copy of the instructions that constitute the subroutine is placed in memory
  - Any program that requires the use of the subroutine 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

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

Subroutine Nesting



THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

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

LaCASA

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
```

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

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
```

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

Main (ver2): Call suma_rp

```
;-------------------------------------------------------------------------------
; 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

            mov.w   #arr1, R12              ; put address into R12
            mov.w   #8, R13                 ; put array length into R13
            mov.w   #0, R14                 ; display #0 (P1&P2)
            call    #suma_rp

            mov.w   #arr2, R12              ; put address into R12
            mov.w   #7, R13                 ; put array length into R13
            mov.w   #1, R14                 ; display #0 (P3&P4)
            call    #suma_rp
            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
```

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

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

LaCASA

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

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.

LaCASA

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

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
```


LaCASA

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

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)

LaCASA

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

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

LaCASA

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

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
```


LaCASA

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

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)

R12 → 0x07F6

SP → 0x0731

LaCASA

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

Performance

- Performance: how fast a task can be completed
- Performance(X) = 1/ExecutionTime(X)
- ET: ExecutionTime

$$ET = IC \cdot CPI \cdot CCT = \frac{IC \cdot CPI}{CF}$$

  - IC: Instruction Count – the number of instructions executed in the program
  - CPI: Cycles Per Instruction – the average number of clock cycles it takes to execute an instruction
  - CCT: Clock Cycle Time – the duration of one processor clock cycle
  - CF: Clock Frequency (1/CCT)

LaCASA

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

Performance: An Example

```
RESET:      mov.w   #__STACK_END,SP          ; 4cc
StopWDT:    mov.w   #WDTPW|WDTHOLD,&WDTCTL   ; 5cc
            push    R14                      ; 3 cc (table 3.15)
            mov.w   SP, R14                  ; 1 cc
            mov.w   #aend, R6                ; 2 cc
            mov.w   R6, R5                   ; 1 cc
            sub.w   #arr1, R5                ; 2 cc
            sub.w   R5, SP                   ; 1 cc
lnext:      dec.w   R6                       ; 1 cc  x 9
            dec.w   R14                      ; 1 cc  x 9
            mov.b   @R6, 0(R14)              ; 4 cc  x 9
            dec.w   R5                       ; 1 cc x 9
            jnz     lnext                    ; 2 cc x 9
            jmp     $

arr1    .byte   1, 2, 3, 4, 5, 6, 7, 8, 9
aend
        .end
TOTAL NUMBER OF CLOCK CYLES:          4+5+3+1+2+1+2+1+9x(1+1+4+1+2) = 19+9x9 = 100 cc
TOTAL NUMBER OF INSTRUCITONS           8+9x5 = 53 instructions
CPI                                   100/53 = 1.88 cc/instruction
```


LaCASA