CPE 323 Introduction to Embedded Computer Systems:
MSP430: Assembly Language and C

Instructor: Dr Aleksandar Milenkovic
Lecture Notes
Outline

- Assembly Language Programming
  - Adding two 32-bit numbers (decimal, integers)
  - Counting characters ‘E’

- Subroutines
  - CALL&RETURN
  - Subroutine Nesting
  - Passing parameters
  - Stack and Local Variables

- C and the MSP430
Assembly Language Programming: 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: 0x68adf0ac

Approach
- Input numbers: storage, placement in memory
- Results: storage (ABSOLUTE ASSEMBLER)
- Main program: initialization, program loops
- Decimal addition, integer addition
/*---------------------------------------------*/
* Program    : Program demonstrates addition of 32-bit decimal and integer numbers
* Input      : Input integers are lint1 and lint2 (constants in flash)
* Output     : Results are stored in lsumd (decimal sum) and lsumi (int sum)
* Written by : A. Milenkovic
* Date       : September 10, 2008
* Description: MSP430 IAR EW; Demonstration of the MSP430 assembler
*----------------------------------------------------*/

#include "msp430.h" ; #define controlled include file

NAME    main ; module name
PUBLIC  main ; make the main label visible
             ; outside this module
ORG 0xF000
lint1:  DC32 0x45678923
lint2:  DC32 0x23456789

ORG     0xFFFFE
DC16    main ; set reset vector to 'init' label

ORG 0x0400
lsumd:  DS32 1 ; equivalent to DS 4
lsumi:  DS32 1 ; equivalent to DS 4

ORG 0xE000
Decimal/Integer Addition of 32-bit Numbers (cont’d)

main:   NOP                             ; main program
        MOV.W  #WDTPW+WDTHOLD,&WDTCTL  ; Stop watchdog timer
        MOV #lint1, R4                 ; pointer to first 4-byte decimal number
        MOV #lsumd, R8
        MOV #2, R5
        CLRC

ldeca:  MOV 4(R4), R7                  ; decimal addition
        DADD @R4+, R7
        MOV R7, 0(R8)
        ADD #2, R8
        DEC R5
        JNZ ldeca

        MOV #lint1, R4
        MOV #lsumd, R8
        MOV #2, R5
        CLRC

lia:    MOV 4(R4), R7                  ; decimal addition
        ADDC @R4+, R7
        MOV R7, 4(R8)
        ADD #2, R8
        DEC R5
        JNZ lia

        JMP $                           ; jump to current location '$'
        ; (endless loop)

END
Assembly Language Directives

ORG 0xF000
b1:   DB   5       ; allocates a byte in memory and initialize ; it
       with constant 5;
       ; equivalent to DC8 5
b2:   DB   -122    ; allocates a byte with constant -122
b3:   DB   10110111b ; binary value of a constant
b4:   DB   0xA0    ; hexadecimal value of a constant
b5:   DB   123q    ; octal value of a constant
EVEN         ; move a location pointer to the first even address
    tf    EQU 25
w1:   DW   32330    ; allocates a a word size constant in memory;
       ; equivalent to DC16 32330
w2:   DW   -32000
dw1:  DL   100000    ; allocates a long word size constant in memory;
       ; equivalent to DC32 100000
dw2:  DL   -10000
dw3:  DL   0xFFFFFFFF
dw4:  DL   tf
s1:   DB 'ABCD'     ; allocates 4 bytes in memory with string ABCD
s2:   DB "ABCD"     ; allocates 5 bytes in memory with string ABCD
       ; and \0 character at the end
Assembly Language Directives (cont’d)

ORG 0x0200

v1b DS 1 ; allocates a byte in memory; equivalent to DS8
v2b DS 1 ; allocates a byte in memory;
v3w DS 2 ; allocates a word of 2 bytes in memory;
          ; equivalent to DS8 2 or DS16
v4b DS32 4 ; allocates a buffer of 4 long words;
           ; 4x4=16 bytes in memory
Assembly Language Programming: 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

- Six parallel ports: x=1,2,3,4,5,6
- 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

- P1OUT
- P1DIR
- P1IN
Count Characters ‘E’

* Program : Counts the number of characters E in a string
* Input   : The input string is the myStr
* Output  : The port one displays the number of E's in the string
* Written by : A. Milenkovic
* Date    : August 14, 2008
* Description: MSP430 IAR EW; Demonstration of the MSP430 assembler

#include "msp430.h" ; #define controlled include file

NAME main ; module name
PUBLIC main ; make the main label visible
ORG 0FFFFH ; outside this module
DC16 init ; set reset vector to 'init' label
RSEG CSTACK ; pre-declaration of segment CSTACK
RSEG CODE ; place program in 'CODE' segment
Count Characters ‘E’ (cont’d)

init:   MOV     #SFE(CSTACK), SP        ; set up stack

main:   NOP                             ; main program
        MOV.W   #WDTPW+WDTHOLD,&WDTCTL  ; stop watchdog timer
        BIS.B   #0FFh,&P1DIR            ; configure P1.x output
        MOV.W   #myStr, R4              ; load the starting address of the string into the
register R4
        CLR.B   R5                      ; register R5 will serve as a counter

gnext:  MOV.B   @R4+, R6                ; get a new character
        CMP     #0,R6
        JEQ     lend                    ; go to the end
        CMP.B   #'E',R6
        JNE     gnext
        INC     R5                      ; increment counter
        JMP     gnext

lend:   MOV.B   R5,&P1OUT                ; set all P1 pins
        JMP     $                        ; jump to itself

myStr   DB "HELLO WORLD, I AM THE MSP430!" ; the string is placed on the stack
        ; the null character is automatically added after the '!

END
Outline

- **Assembly Language Programming**
  - Adding two 32-bit numbers (decimal, integers)
  - Counting characters ‘E’

- **Subroutines**
  - CALL&RETURN
  - Subroutine Nesting
  - Passing parameters
  - Stack and Local Variables

- **C and the MSP430**
The Case for Subroutines: An Example

Problem
- Sum up elements of two integer arrays
- Display results on P2OUT&P1OUT and P4OUT&P3OUT

Example
- arr1 DC16 1, 2, 3, 4, 1, 2, 3, 4 ; the first array
- arr2 DC16 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)

/*----------------------------------*/
* Program    : Find a sum of two integer arrays;
* Input      : The input arrays are signed 16-bit integers in arr1 and arr2
* Output     : Display sum of arr1 on P1OUT&P2OUT and sum of arr2 on P3OUT&P4OUT
* Modified by: A. Milenkovic, milenkovic@computer.org
* Date       : September 14, 2008
* Description: MSP430 IAR EW; Demonstration of the MSP430 assembler
*----------------------------------*/

#include "msp430.h" ; #define controlled include file

NAME    main                    ; module name
PUBLIC  main                    ; make the main label visible
ORG     OFFFEh
DC16    init                    ; set reset vector to 'init' label
RSEG    CSTACK                  ; pre-declaration of segment
RSEG    CODE                    ; place program in 'CODE' segment
init:   MOV    #SFE(CSTACK), SP ; set up stack
Sum up two integer arrays (ver1)

main:   NOP                             ; main program
        MOV.W #WDTPW+WDTHOLD,&WDTCTL  ; Stop watchdog timer
        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     #arr1, R4               ; load the starting address of the array1 into the register R4
        CLR     R7                      ; Holds the sum
        MOV     #8, R10                 ; number of elements in arr1

        lnext1: ADD     @R4+, R7                ; get next element
                 DEC     R10
                 JNZ     lnext1
        MOV.B   R7, P1OUT               ; display sum of arr1
                 SWPB    R7
        MOV.B   R7, P2OUT

        ; Sum arr2 and display
        MOV     #arr2, R4
        CLR     R7
        MOV     #7, R10                    ; number of elements in arr2

        lnext2: ADD     @R4+, R7                ; get next element
                 DEC     R10
                 JNZ     lnext2
        MOV.B   R7, P3OUT               ; display sum of arr1
                 SWPB    R7
        MOV.B   R7, P4OUT

        JMP     $                          ; End of main routine

arr1    DC16     1, 2, 3, 4, 1, 2, 3, 4     ; the first array
arr2    DC16     1, 1, 1, 1, -1, -1, -1, -1     ; the second array

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 instructions: perform the branch to subroutines
- RETURN instruction: the last instruction in the subroutine
Subroutine Nesting
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

/*------------------------------------------------------------------------------
* Program  : Subroutine for that sums up elements of an integer array
* Input    : The input parameters are passed through registers:
            R12 - starting address of the array
            R13 - array length
            R14 - display id (0 for P2&P1, 1 for P4&P3)
* Output   : No output parameters
*------------------------------------------------------------------------------*/

#include "msp430.h"                     ; #define controlled include file

PUBLIC suma_rp

RSEG CODE

suma_rp:

; save the registers on the stack
    PUSH    R7                      ; temporal sum
    CLR     R7

lnext:  ADD     @R12+, R7
    DEC     R13
    JNZ     lnext
    BIT     #1, R14                 ; display on P1&P2
    JNZ     lp34                    ; it's P3&P4
    JNZ     lp34
    MOV.B   R7, P1OUT
    SWPB    R7
    MOV.B   R7, P2OUT
    JMP     lend

lp34:   MOV.B   R7, P3OUT
    SWPB   R7
    MOV.B   R7, P4OUT

lend:   POP    R7                      ; restore R7
    RET
END
Sum Up Two Integer Arrays (ver2)

/*------------------------------------------------------------------------------
* Program    : Find a sum of two integer arrays using a subroutine (suma_rp.s43)
* Input      : The input arrays are signed 16-bit integers in arr1 and arr2
* Output     : Display sum of arr1 on P1OUT&P2OUT and sum of arr2 on P3OUT&P4OUT
* Modified by: A. Milenkovic, milenkovic@computer.org
* Date       : September 14, 2008
* Description: MSP430 IAR EW; Demonstration of the MSP430 assembler
*-----------------------------------------------------------------------------*/

#include "msp430.h" ; #define controlled include file

NAME    main ; module name
PUBLIC  main ; make the main label visible
            ; outside this module
EXTERN  suma_rp

ORG     0FFFEh ; set reset vector to 'init' label
DC16    init
RSEG    CSTACK ; pre-declaration of segment
RSEG    CODE ; place program in 'CODE' segment

init:   MOV     #SFE(CSTACK), SP ; set up stack
Sum Up Two Integer Arrays (ver2)

main:   NOP ; main program
        MOV.W #WDTPW+WDTHOLD,&WDTCTL ; Stop watchdog timer
        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    #arr1, R12 ; put address into R12
        MOV    #8, R13 ; put array length into R13
        MOV    #0, R14 ; display #0 (P1&P2)
        CALL   #suma_rp

        MOV    #arr2, R12 ; put address into R12
        MOV    #7, R13 ; put array length into R13
        MOV    #1, R14 ; display #0 (P3&P4)
        CALL   #suma_rp

        JMP    $ ;

arr1   DC16  1, 2, 3, 4, 1, 2, 3, 4 ; the first array
arr2   DC16  1, 1, 1, 1, -1, -1, -1 ; the second array

END
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
Subroutine: SUMA_SP

/*---------------------------------------------------------------*/
/* Program : Subroutine for that sums up elements of an integer array*/
/* Input   : The input parameters are passed through the stack:     */
/*           starting address of the array                         */
/*           array length                                         */
/*           display id                                          */
/* Output   : No output parameters                              */
/*---------------------------------------------------------------*/
#include "msp430.h" ; #define controlled include file

PUBLIC suma_sp

RSEG CODE

suma_sp:
    ; save the registers on the stack
    PUSH    R7                      ; temporal sum
    PUSH    R6                      ; array length
    PUSH    R4                      ; pointer to array
    CLR     R7
    MOV     10(SP), R6               ; retrieve array length
    MOV     12(SP), R4

lnext:  ADD     @R4+, R7
        DEC     R6
        JNZ     lnext
        MOV     8(SP), R4               ; get id from the stack
        BIT     #1, R4                 ; display on P1&P2
        JNZ     lp34                    ; it's P3&P4
        MOV.B   R7, P1OUT
        SWPB    R7
        MOV.B   R7, P2OUT
        JMP     lend

lp34:   MOV.B   R7, P3OUT
        SWPB    R7
        MOV.B   R7, P4OUT

lend:   POP     R4                      ; restore R4
        POP     R6
        POP     R7
        RET
END
Sum Up Two Integer Arrays (ver3)

* Program    : Find a sum of two integer arrays
* Input      : The input arrays are signed 16-bit integers in arr1 and arr2
* Output     : Display sum of arr1 on P1OUT&P2OUT and sum of arr2 on P3OUT&P4OUT
* Modified by: A. Milenkovic, milenkovic@computer.org
* Date       : September 14, 2008
* Description: MSP430 IAR EW; Demonstration of the MSP430 assembler

#include "msp430.h" ; #define controlled include file

NAME    main                    ; module name
PUBLIC  main                    ; make the main label visible
                             ; outside this module
EXTERN  suma_spORG     0FFFEh
DC16    init                    ; set reset vector to 'init' label
                             ; pre-declaration of segment
                             ; place program in 'CODE' segment
init:   MOV     #SFE(CSTACK), SP  ; set up stack
Sum Up Two Integer Arrays (ver3)

main:    NOP                             ; main program
        MOV.W #WDTPW+WDTHOLD,&WDTCTL  ; Stop watchdog timer
        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               ; collapse the stack
        ADD     #6,SP                   ; ; push display id
        CALL    #suma_sp               ; collapse the stack
        JMP     $                        ;

arr1    DC16     1, 2, 3, 4, 1, 2, 3, 4     ; the first array
arr2    DC16     1, 1, 1, 1, -1, -1, -1     ; the second array

END
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

/*------------------------------------------------------------------------------
* Program  : Subroutine for that sums up elements of an integer array
* Subroutine variables are all allocated on the stack frame
*   counter  (SFP+2)
*   sum     (SFP+4)
* Input    : The input parameters are passed through the stack:
*            starting address of the array
*            array length
*            display id
* Output   : No output parameters
*------------------------------------------------------------------------------*/
#include "msp430.h"                     ; #define controlled include file
PUBLIC suma_sp
RSEG CODE
suma_sp:
; save the registers on the stack
PUSH    R12                     ; save R12 - R12 is stack frame pointer
MOV     SP, R12                 ; R12 points on the bottom of the stack frame
SUB     #4, SP                  ; allocate 4 bytes for local variables
PUSH    R4                      ; pointer register
CLR     -4(R12)                 ; clear sum, sum=0
MOV     6(R12), -2(R12)         ; init count
MOV     8(R12), R4              ; R4 points to the array starting address
INext:  ADD     @R4+, -4(R12)           ; add next element
DEC     -2(R12)                 ; decrement counter
JNZ     INext
BIT     #1, 4(R12)               ; get id from the stack
JNZ     lp34                    ; it's P3&P4
MOV.B   -4(R12), P1OUT          ; add next element
MOV.B   -3(R12), P2OUT          ; decrement counter
JMP     lend
lp34:   MOV.B   -4(R12), P3OUT
MOV.B   -3(R12), P4OUT
lend:   POP     R4                      ; restore R4
ADD     #4, SP                  ; collapse the stack frame
POP     R12                     ; restore stack frame pointer
RET
END

CPE 323 Intro2EmbeddedSystems 29
## Sum Up Two Integer Arrays (ver3)

* Program : Find a sum of two integer arrays
* Input  : The input arrays are signed 16-bit integers in arr1 and arr2
* Output : Display sum of arr1 on P1OUT&P2OUT and sum of arr2 on P3OUT&P4OUT
* Modified by: A. Milenkovic, milenkovic@computer.org
* Date    : September 14, 2008
* Description: MSP430 IAR EW; Demonstration of the MSP430 assembler

```c
#include "msp430.h" ; #define controlled include file

NAME    main                    ; module name
PUBLIC  main                    ; make the main label visible
; outside this module
EXTERN  suma_spORG     0FFFEh
DC16    init                    ; set reset vector to 'init' label
RSEG    CSTACK                  ; pre-declaration of segment
RSEG    CODE                    ; place program in 'CODE' segment

init:   MOV     #SFE(CSTACK), SP ; set up stack
```
Sum Up Two Integer Arrays (ver3)

main:   NOP                             ; main program
        MOV.W  #WDTPW+WDTHOLD,&WDTCTL  ; Stop watchdog timer
        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                ; collapse the stack
        ADD    #6,SP                   ; collapse the stack
        PUSH   #arr2                   ; push the address of arr2
        PUSH   #7                      ; push the number of elements
        PUSH   #1                      ; push display id
        CALL   #suma_sp                ; collapse the stack
        JMP    $                        ; the first array
        DC16   1, 2, 3, 4, 1, 2, 3, 4     ; the first array
        DC16   1, 1, 1, 1, -1, -1, -1     ; the second array

END
Outline

- Assembly Language Programming
  - Adding two 32-bit numbers (decimal, integers)
  - Counting characters ‘E’
- Subroutines
  - CALL&RETURN
  - Subroutine Nesting
  - Passing parameters
  - Stack and Local Variables
- C and the MSP430
Assembly Language and C

We are interested in:
- How a high-level language uses low-level language features?
- C: System programming, device drivers, ...
- Use of addressing modes by compilers
- Parameter passing in assembly language
- Local storage
C and the MSP430

- Compiler and the MSP430 instruction set
- C data types and implementation
- Storage classes
- Functions and parameters
- Pointers
Compiling a C Program: Example #1

```
#include "io430.h"
int main ( void ) {
    int i1, i2;
    unsigned int ui1;
    short int sint1;
    long int lint2;
    int a[4];
    // Stop watchdog timer to prevent time out reset
    WDTCTL = WDTPW + WDTHOLD;
    i1 = 2; i2 = -2;
    ui1=65535;
    sint1=127;
    lint2=128243;
    a[0]=20; a[1]=9;
    return 0;
}
```
Example #1 Compiler Generated List File (no optimization)
Example #1 Compiler Generated List File (no optimization)

1 #include "io430.h"

\ 
\ union <unnamed> volatile __data16 __A_WDTCTL
\ __A_WDTCTL:
\ 000000 DS8 2
\ In segment CODE, align 2

2 int main( void ) {
 \ main:
 \ 000000 0A12 PUSH.W R10
 \ 000002 0812 PUSH.W R8
 \ 000004 0912 PUSH.W R9
 \ 000006 3182 SUB.W #0x8, SP

3 int i1, i2;

Warning[Pe550]: variable "i1" was set but never used

int i1, i2;

Warning[Pe550]: variable "i2" was set but never used

"C:\Documents and Settings\Aleksandar\My Documents\Work\teaching\cpe323-08F\tutorial\test_dtypes.c",3 Warning[Pe550]:
 variable "i2" was set but never used

4 unsigned int ui1;

Warning[Pe550]: variable "ui1" was set but never used

5 short int sint1;

Warning[Pe550]: variable "sint1" was set but never used

6 long int lint2;

Warning[Pe550]: variable "lint2" was set but never used

7 int a[4];

Warning[Pe550]: variable "a" was set but never used
Example #1 Compiler Generated List File (no optimization)

Maximum stack usage in bytes:

<table>
<thead>
<tr>
<th>Function</th>
<th>CSTACK</th>
</tr>
</thead>
<tbody>
<tr>
<td>main</td>
<td>16</td>
</tr>
</tbody>
</table>

Segment part sizes:

<table>
<thead>
<tr>
<th>Function/Label</th>
<th>Bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>_A_WDTCTL</td>
<td>2</td>
</tr>
<tr>
<td>main</td>
<td>56</td>
</tr>
</tbody>
</table>

56 bytes in segment CODE
2 bytes in segment DATA16_AN
56 bytes of CODE memory
0 bytes of DATA memory (+ 2 bytes shared)

Errors: none
Warnings: 6
## C Data Types

<table>
<thead>
<tr>
<th>Data type</th>
<th>Size</th>
<th>Range</th>
<th>Alignment</th>
</tr>
</thead>
<tbody>
<tr>
<td>bool</td>
<td>8 bits</td>
<td>0 to 1</td>
<td>1</td>
</tr>
<tr>
<td>char</td>
<td>8 bits</td>
<td>0 to 255</td>
<td>1</td>
</tr>
<tr>
<td>signed char</td>
<td>8 bits</td>
<td>-128 to 127</td>
<td>1</td>
</tr>
<tr>
<td>unsigned char</td>
<td>8 bits</td>
<td>0 to 255</td>
<td>1</td>
</tr>
<tr>
<td>signed short</td>
<td>16 bits</td>
<td>-32768 to 32767</td>
<td>2</td>
</tr>
<tr>
<td>unsigned short</td>
<td>16 bits</td>
<td>0 to 65535</td>
<td>2</td>
</tr>
<tr>
<td>signed int</td>
<td>16 bits</td>
<td>-32768 to 32767</td>
<td>2</td>
</tr>
<tr>
<td>unsigned int</td>
<td>16 bits</td>
<td>0 to 65535</td>
<td>2</td>
</tr>
<tr>
<td>signed long</td>
<td>32 bits</td>
<td>-2(^{31}) to 2(^{31}-1)</td>
<td>2</td>
</tr>
<tr>
<td>unsigned long</td>
<td>32 bits</td>
<td>0 to 2(^{32}-1)</td>
<td>2</td>
</tr>
<tr>
<td>signed long long</td>
<td>64 bits</td>
<td>-2(^{63}) to 2(^{63}-1)</td>
<td>2</td>
</tr>
<tr>
<td>unsigned long long</td>
<td>64 bits</td>
<td>0 to 2(^{64}-1)</td>
<td>2</td>
</tr>
<tr>
<td>float</td>
<td>32 bits</td>
<td></td>
<td>2 (*)</td>
</tr>
<tr>
<td>double</td>
<td>32 bits</td>
<td></td>
<td>2 (*)</td>
</tr>
<tr>
<td>double</td>
<td>64 bits</td>
<td></td>
<td>2 (*)</td>
</tr>
</tbody>
</table>
C Data Types, cont’d

- Local variables
  - Defined inside a function
  - Cannot be accessed from outside the function
  - Normally lost when a return from the function is made

- Global variables
  - Defined outside a function
  - Can be accessed both from inside and outside the function

- Variables defined in a block exist only within that block

```c
int i; /*global variable, visible to everything from this point*/
void function_1(void) /*A function with no parameters*/
{
    int k; /*Integer k is local to function_1*/
    {
        int q; /*Integer q exists only in this block*/
        int j; /*Integer j is local and not the same as j in main*/
    }
}
void main(void)
{
    int j; /*Integer j is local to this block within function main*/
} /*This is the point at which integer j ceases to exist*/
```
Storage Class Specifiers

- **auto**
  - Variable is no longer required once a block has been left; Default

- **register**
  - Ask compiler to allocate the variable to a register
  - Also is automatic
  - Cannot be accessed by means of pointers

- **static**
  - Allows local variable to retain its value when a block is reentered
  - Initialized only once, by the compiler!

- **extern**
  - Indicates that the variable is defined outside the block
  - The same global variable can be defined in more than one module
Storage Class Modifiers

- **volatile**
  - To define variables that can be changed externally
  - Compiler will not put them in registers
  - Think about Status Registers!

- **const**
  - Variable may not be changed during the execution of a program
  - Cannot be changed unintentionally, but CAN be changed externally (as a result of an I/O, or OS operations external to the C program)

- Type conversion
  - In C, done either automatically or explicitly (casting)
#include "io430.h"
int main( void ) {
    volatile int i1, i2;
    volatile unsigned int ui1;
    volatile short int sint1;
    volatile long int lint2;
    volatile int a[4];
    // Stop watchdog timer to prevent time out reset
    WDTCTL = WDTPW + WDTHOLD;
    i1 = 2; i2 = -2;
    ui1=65535;
    sint1=127;
    lint2=128243;
    a[0]=20; a[1]=9;
    return 0;
}
Example #2 Compiler Generated List File (no optimization)

C:\Documents and Settings\Aleksandar\My Documents\Work\teaching\cpe323-08F\tutorial\test_dtypes.c
1#include "io430.h"

\ In segment DATA16_AN, at 0x120
\ union <unnamed> volatile __data16 __A_WDTCTL
| __A_WDTCTL:
\ 000000 DS8 2

\ In segment CODE, align 2
2 int main( void ) {
  
  \ main:
2    000000 31801400 SUB.W #0x14, SP
3    volatile int i1, i2;
4    volatile unsigned int ui1;
5    volatile short int sint1;
6    volatile long int lint2;
7    volatile int a[4];
8    // Stop watchdog timer to prevent time out reset
9    WDTCTL = WDTPW + WDTHOLD;
\ 000004 B240805A2001 MOV.W #0x5a80, &0x120
10   i1 = 2; i2 = -2;
\ 00000A A1430000 MOV.W #0x2, 0(SP)
\ 00000E B140FEFF0200 MOV.W #0xfffe, 0x2(SP)
11   ui1=65535;
\ 000014 B1430400 MOV.W #0xffff, 0x4(SP)
12   sint1=127;
\ 000018 B1407F000600 MOV.W #0x7f, 0x6(SP)
13   lint2=128243;
\ 00001E B140F3F40800 MOV.W #0xf4f3, 0x8(SP)
\ 000024 91430A00 MOV.W #0x1, 0xa(SP)
Example #2 Compiler Generated List File (no optimization)

14    a[0]=20; a[1]=9;
\    000028  B14014000C00  MOV.W  #0x14, 0xc(SP)
\    00002E  B14009000E00  MOV.W  #0x9, 0xe(SP)
15    return 0;
\    000034  0C43         MOV.W  #0x0, R12
\    000036  31501400     ADD.W  #0x14, SP
\    00003A  3041         RET
\    00003C  REQUIRE _A_WDTCTL
16    }  
Maximum stack usage in bytes:

    Function  CSTACK
              ------- ------
               main    22

Segment part sizes:

    Function/Label  Bytes
                --------- ----
               _A_WDTCTL     2
               main         60

60 bytes in segment CODE
2 bytes in segment DATA16_AN

60 bytes of CODE memory
0 bytes of DATA memory (+ 2 bytes shared)

Errors: none
Warnings: none
#include "stdio.h"
#include "io430.h"

int fact(int n);

int main(void) {
    int n = 5;
    int nf;
    nf = fact(n);
    printf("n=%d, nf=%d\n", n, nf);
    return 0;
}

int fact(int n) {
    if(n>1) return n*fact(n-1);
    else return 1;
}
```c
#include "stdio.h"
#include "io430.h"

int fact(int n);

int main(void) {
    int n = 5;
    int nf;
    nf = fact(n);
    printf("n=%d, nf=%d\n", n, nf);
    return 0;
}
```
Factorial: List File

19 int fact(int n) {
   \ fact:
   000000 0A12 PUSH.W R10
   000002 0A4C MOV.W R12, R10
   20
   21   if(n>1) return n*fact(n-1);
   000004 2A93 CMP.W #0x2, R10
   000006 0E38 JL ??fact_0
   000008 0C4A MOV.W R10, R12
   00000A 3C53 ADD.W #0xffff, R12
   \ ??fact_0:
   00000C B012.... CALL #fact
   000010 0212 PUSH.W SR
   000012 32C2 DINT
   000014 824A3001 MOV.W R10, &0x130
   000018 824C3801 MOV.W R12, &0x138
   00001C 1C423A01 MOV.W &0x13a, R12
   000020 3241 POP.W SR
   \ ??fact_1:
   000022 013C JMP ??fact_1
   22   else return 1;
   \ ??fact_0:
   000024 1C43 MOV.W #0x1, R12
   \ ??fact_1:
   000026 3A41 POP.W R10
   000028 3041 RET
   23 }
   \ In segment DATA16_C, align 1, align-
   \ sorted
   `?<Constant "n=%d, nf=%d\n">`
   000000 6E3D25642C20 DC8 "n=%d, nf=%d\012"
   \ 6E663D25640A
   \ 00
Functions and Parameters

#include "io430.h"
void swapbyv(int a, int b);
void swapbyr(int *a, int *b);
int main( void )
{
    // Stop watchdog timer to prevent time out reset
    WDTCTL = WDTPW + WDTHOLD;
    int x = 5;
    int y = 6;
    // pass parameters by value
    swapbyv(x, y);
    // pass parameters by reference
    swapbyr(&x, &y);

    return 0;
}

void swapbyv(int a, int b) {
    int temp;
    temp = a;
    a = b;
    b = temp;
}

void swapbyr(int *a, int *b) {
    int temp;
    temp = *a;
    *a = *b;
    *b = temp;
}
Functions and Parameters

```c
int main( void )
{
    // Stop watchdog timer to prevent time out reset
    WDTCTL = WDTPW + WDTHOLD;

    int x = 5;
    int y = 6;

    swapbyv(x, y);
    swapbyr(&x, &y);

    return 0;
}
```

 REQUIRE _A_WDTCTL

CPE 323 Intro2EmbeddedSystems
Functions and Parameters

In segment CODE,
align 2
32    void swapbyv(int a, int b) {
   
   swapbyv:
   33        int temp;
   34
   35        temp = a;
   36
   000000   0F4C         MOV.W   R12, R15
   37        a = b;
   38
   000002   0C4D         MOV.W   R13, R12
   39        b = temp;
   40
   000004   0D4F         MOV.W   R15, R13
   41    }
   000006   3041         RET

   In segment CODE,
align 2
40    void swapbyr(int *a, int *b) {
   
   swapbyr:
   41        int temp;
   42
   43        temp = *a;
   44
   000000   2F4C         MOV.W   @R12, R15
   45        *a = *b;
   46
   000002   AC4D0000     MOV.W   @R13, 0(R12)
   47        *b = temp;
   48
   000004   8D4F0000     MOV.W   R15, 0(R13)
   49    }
   000006   3041         RET

Maximum stack usage in bytes:

<table>
<thead>
<tr>
<th>Function</th>
<th>CSTACK</th>
</tr>
</thead>
<tbody>
<tr>
<td>main</td>
<td>6</td>
</tr>
<tr>
<td>-&gt; swapbyv</td>
<td>6</td>
</tr>
<tr>
<td>-&gt; swapbyr</td>
<td>6</td>
</tr>
<tr>
<td>swapbyr</td>
<td>2</td>
</tr>
<tr>
<td>swapbyv</td>
<td>2</td>
</tr>
</tbody>
</table>

Segment part sizes:

<table>
<thead>
<tr>
<th>Function/Label</th>
<th>Bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>_A_WDTCTL</td>
<td>2</td>
</tr>
<tr>
<td>main</td>
<td>46</td>
</tr>
<tr>
<td>swapbyv</td>
<td>8</td>
</tr>
<tr>
<td>swapbyr</td>
<td>12</td>
</tr>
</tbody>
</table>

66 bytes in segment CODE
2 bytes in segment DATA16_AN

66 bytes of CODE memory
0 bytes of DATA memory (+ 2 bytes shared)
#include "io430.h"
#include "stdio.h"

int main( void ) {
    // Stop watchdog timer to prevent time out reset
    WDTCTL = WDTPW + WDTHOLD;
    int x = 5;  // an integer x
    int *p_x;   // a pointer to int
    int y1;     // an integer y1 (uninitialized)
    long int y2, y3; // long integers y2, y3
    long int *p_y2; // a pointer to long integer
    char mya[20] = "hello world, cpe323!"; // character array
    char *p_mya; // pointer to character

    p_x = &x;        // p_x points to x
    y1 = 10 + x;     // new value to y1
    y2 = -1;
    p_y2 = &y2;      // pointer p_y2 points to y2
    y3 = 10 + *p_y2;
    p_mya = mya;     // p_mya points to array mya
    p_mya = p_mya + 3;

    // display addresses and variables in terminal i/o
    printf("a.x=%x, x=%x\n", &x, x);
    printf("a.p_x=%x, p_x=%x\n", &p_x, p_x);
    printf("a.y1=%x, y1=%x\n", &y1, y1);
    printf("a.y2=%x, y2=%lx\n", &y2, y2);
    printf("a.y3=%x, y3=%lx\n", &y3, y3);
    printf("a.p_y2=%x, p_y2=%x\n", &p_y2, p_y2);
    printf("a.mya=%x, mya=%s\n", &mya, mya);
    printf("a.p_mya=%x, p_mya=%x\n", &p_mya, p_mya);
    return 0;
}
Pointers and C, cont’d

```c
#include "io430.h"

#include "stdio.h"

int main(void) {
    // Stop watchdog timer to prevent time out reset
    WDTCTL = WDTPW + WDTHOLD;
    int x = 5;  // an integer x
    int *p_x;   // a pointer to int
    int y1;     // an integer y1 (uninitialized)
    long int y2, y3; // long integers y2, y3
    long int *p_y2; // a pointer to long integer
    char mya[20] = "hello world, cpe323!";    // character array
    char *p_mya;     // pointer to character
    p_x = &x;        // p_x points to x
```

```
#include "io430.h"

#include "stdio.h"

int main(void) {
    // Stop watchdog timer to prevent time out reset
    WDTCTL = WDTPW + WDTHOLD;
    int x = 5;  // an integer x
    int *p_x;   // a pointer to int
    int y1;     // an integer y1 (uninitialized)
    long int y2, y3; // long integers y2, y3
    long int *p_y2; // a pointer to long integer
    char mya[20] = "hello world, cpe323!";    // character array
    char *p_mya;     // pointer to character
    p_x = &x;        // p_x points to x

    In segment CODE, align 2
```

```c
    int main(void) {
        // Stop watchdog timer to prevent time out reset
        WDTCTL = WDTPW + WDTHOLD;
        int x = 5;  // an integer x
        int *p_x;   // a pointer to int
        int y1;     // an integer y1 (uninitialized)
        long int y2, y3; // long integers y2, y3
        long int *p_y2; // a pointer to long integer
        char mya[20] = "hello world, cpe323!";    // character array
        char *p_mya;     // pointer to character
        p_x = &x;        // p_x points to x
```
Pointers and C, cont’d

16          y1 = 10 + x;     // new value to y1
  y1 = 10 + x;     // new value to y1
\  000028   2F41         MOV.W   @SP, R15
\  00002A   3F500A00     ADD.W   #0xa, R15
\  00002E   814F0600     MOV.W   R15, 0x6(SP)
  y2 = -1;
\  000032   B1430A00     MOV.W   #0xffff, 0xa(SP)
\  000036   B1430C00     MOV.W   #0xffff, 0xc(SP)
  p_y2 = &y2;      // pointer p_y2 points to y2
\  00003A   0F41         MOV.W   SP, R15
\  00003C   3F500A00     ADD.W   #0xa, R15
\  000040   814F0400     MOV.W   R15, 0x4(SP)
  y3 = 10 + *p_y2;
\  000044   1F410400     MOV.W   0x4(SP), R15
\  000048   2E4F         MOV.W   @R15, R14
\  00004A   1F4F0200     MOV.W   0x2(R15), R15
\  00004E   3E500A00     ADD.W   #0xa, R14
\  000052   0F63         ADDC.W  #0x0, R15
\  000054   814E0E00     MOV.W   R14, 0xe(SP)
\  000058   814F1000     MOV.W   R15, 0x10(SP)
  p_mya = mya;     // p_mya points to array mya
\  00005C   0F41         MOV.W   SP, R15
\  00005E   3F501200     ADD.W   #0x12, R15
\  000062   814F0200     MOV.W   R15, 0x2(SP)
  p_mya = p_mya + 3;
\  000066   B15003000200 ADD.W   #0x3, 0x2(SP)
Speed and Performance of Microprocessors

- Why is difficult to compare the speed of two microprocessors?
  - Performance
  - Execution time
  - MIPS: Million of Instructions Per Second
- Carefully interpret benchmarks!
- Clock Cycles/Bus Cycles
Speed and Performance of Microprocessors, cont’d

```assembly
#include "msp430.h" ; #define controlled include file
NAME main ; module name
PUBLIC main ; make the main label visible
; outside this module

ORG 0FFFEh DC16 init ; set reset vector to 'init' label
RSEG CSTACK ; pre-declaration of segment
RSEG CODE ; place program in 'CODE' segment

init: MOV #SFE(CSTACK), SP ; set up stack
main: NOP ; main program
MOV.W #WDTPW+WDTHOLD,&WDTCTL ; Stop watchdog timer
PUSH R14 ; R14 will serve as a frame pointer
MOV SP, R14 ; R14 points to the top of the stack
MOV #aend, R6
MOV R6, R5
SUB #arr1, R5 ; how many bytes is in the array
SUB R5, SP ; allocate storage for array on the stack

lnext: DEC R6 ; decrement pointer to arr1
DEC R14 ; decrement pointer on the stack
MOV.B @R6, 0(R14)
DEC R5
JNZ lnext
JMP $ NOP

arr1 DC8 1, 2, 3, 4, 5, 6, 7, 8, 9
aend
END
```
#include "msp430.h" ; #define controlled include file
NAME main ; module name
PUBLIC main ; make the main label visible ; outside this module

ORG 0FFFEh ; set reset vector to 'init' label
DC16 init ; pre-declaration of segment
RSEG CSTACK ; place program in 'CODE' segment
RSEG CODE

init: MOV #SFE(CSTACK), SP ; 4 cc
main: NOP ; 1 cc
MOV.W #WDTPW+WDTHOLD,&WDTCTL ; 5 cc
PUSH R14 ; 3 cc (table 3.15)
MOV SP, R14 ; 1 cc
MOV #aend, R6 ; 2 cc
MOV R6, R5 ; 1 cc
SUB #arr1, R5 ; 2 cc
SUB R5, SP ; 1 cc

lnext: DEC R6 ; 1 cc x 9
DEC R14 ; 1 cc x 9
MOV.B @R6, 0(R14) ; 4 cc x 9
DEC R5 ; 1 cc x 9
JNZ lnext ; 2 cc x 9
JMP $ ;

arr1 DC8 1, 2, 3, 4, 5, 6, 7, 8, 9
aend

TOTAL NUMBER OF CLOCK CYLES: 4+1+5+3+1+2+1+2+1+9x(1+1+4+1+2) = 20+9x9 = 101 cc
TOTAL NUMBER OF INSTRUCTIONS 9+9x5 = 54 instructions
CPI 101/54 = 1.87 cc/instruction