CPE 323
Intro to Embedded Computer Systems
Assembly Language Programming

Aleksandar Milenkovic

milenka@uah.edu

Admin

Addition Indexed Indexed

$\text{Add.B} \quad 4(R5), \quad 6(R6) \; ; \quad \overset{\text{Ead}}{M[6+R6]} \leftarrow M[R6+6] + \overset{\text{Eas}}{M[4+R5]}$

$R6 = 0x0401$

$R5 = 0xC006$

$Eas = 4 + R5 = 4 + 0xC006 = 0xC00A$

$Ead = R6 + 6 = 6 + 0x0401 = 0x0407$

Source operand: $M[EAS] = M[C00A] = 2B$

Source/dest: $M[EAd] = M[0x0407] = 0x0003$

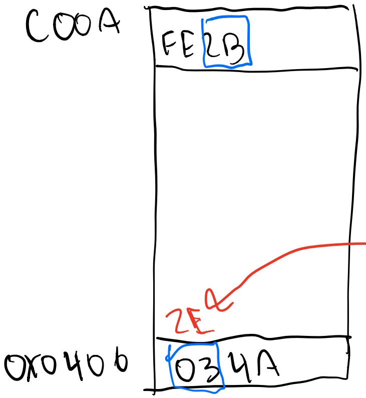
$$\begin{array}{r} \scriptstyle 0\;\;0 \\ 0x\,2B \\ +\;0x\,03 \\ \hline 0x\,2E \end{array}$$

$C = 0$

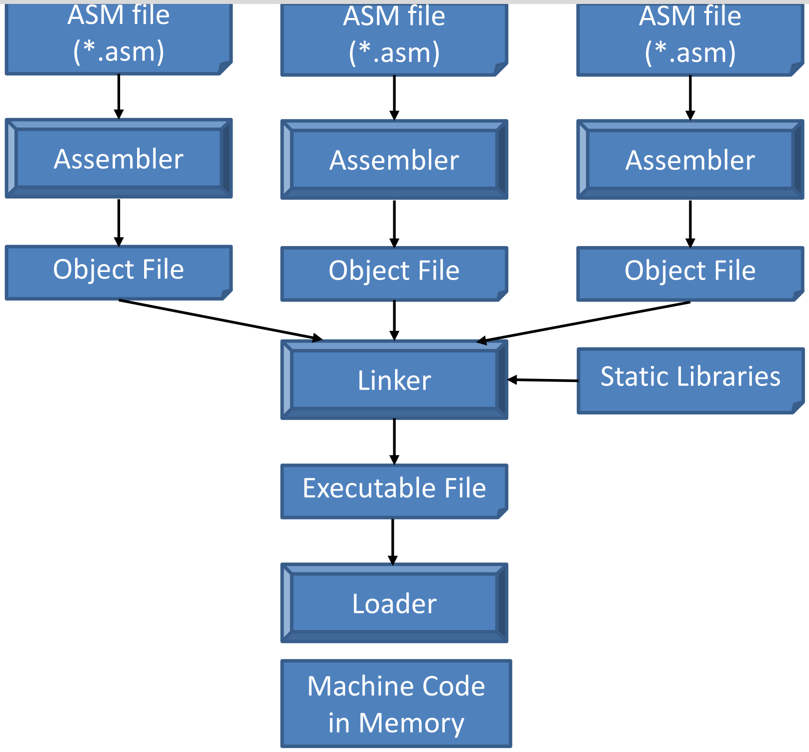
$V = 0$

$N = 0$

$Z = 0$



Assembly Development Flow



Assembly Language 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

ASM Section Control Directives

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

Constant Initialization Directives

- .byte
- .float
- .word
- .long
- .string

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
```

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
```

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
```

Table of Symbols

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

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		--	--
v3w		--	--
v4b		--	--
		--	--
		--	--
		--	--
vx			

Symbol	Value [hex]
v1b	
v2b	
v3w	
v4b	
vx	

Decimal/Integer Addition of 32-bit Numbers

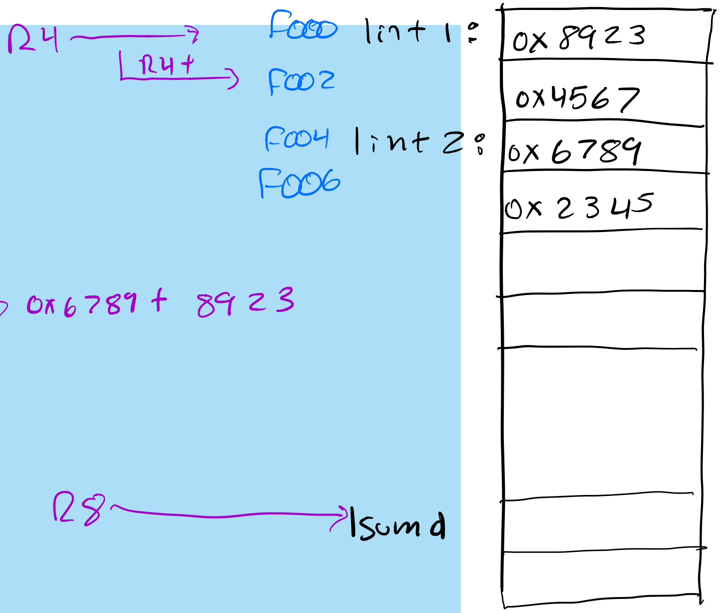
- Write an assembly program that finds a sum of two 32-bit numbers
  - Input numbers are decimal numbers (8-digit in length)
  - Input numbers are signed integers in two's complement
- E.g.:
- `lint1: .long 0x45678923`
- `lint2: .long 0x23456789`

Allocate Space & Start Program

Main Code (Ver. 1)

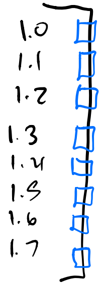
Main Code (Ver. 2)

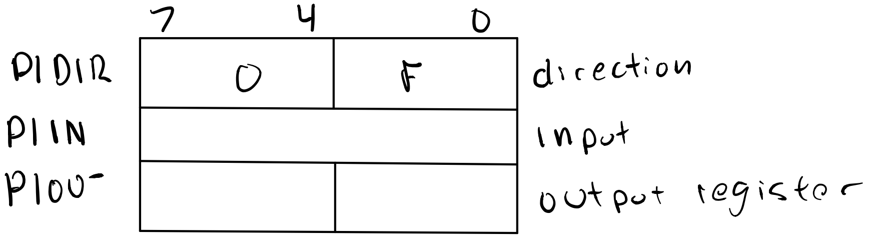
```
mov.w     #lint1, R4
mov.w     #lsumd, R8
mov.w     #21, R5       ; R5 step counter
clr       R10           ; R10 = 0
mov.w     4(R4), R7     ; R7 = 0x6789
mov.w     R10, R2       ;
dadd.w    @R4+, R7      ; R7 ← R7 + M[R4] -> 0x6789 + 8923
mov.w     R2, R10       ;
mov.w     R7, 0(R8)     ;
dec.w     R5
```



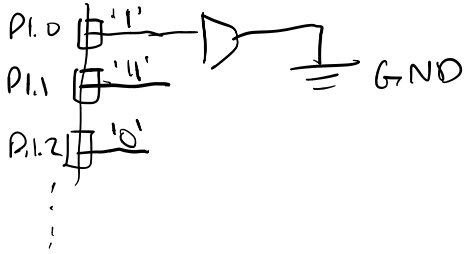
Count Characters ‘E’ in a String

- 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





P1 DIR = 0xFF
P1 OUT = 0x03



Count Characters ‘E’ in a String

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

Count Characters ‘E’ in a String

```
;-------------------------------------------------------------------------------
; Main loop here
;-------------------------------------------------------------------------------
main:  bis.b   #0xFF, &P1DIR
       mov.w   #mystr, R4
       clr.b   R5                ; counts 'E's
Qnext: mov.b   @R4+, R6          ; R6 ← M[R4]; R4←R4+1
       cmp.b   #0, R6            ; NULL = 0x00
       jeq     lend
                                 lend: mov.b R5, &P1OUT
       cmp.b   #'E', R6
       jne     qnext
       inc.w   R5
       jmp     qnext
;-------------------------------------------------------------------------------
; Stack Pointer definition
;-------------------------------------------------------------------------------
          .global __STACK_END
          .sect   .stack
;-------------------------------------------------------------------------------
; Interrupt Vectors
;-------------------------------------------------------------------------------
            .sect   ".reset"              ; MSP430 RESET Vector
            .short  RESET
            .end
```