CPE 323
Intro to Embedded Computer Systems
Assembly Language Programming

Aleksandar Milenkovic

milenka@uah.edu

Admin

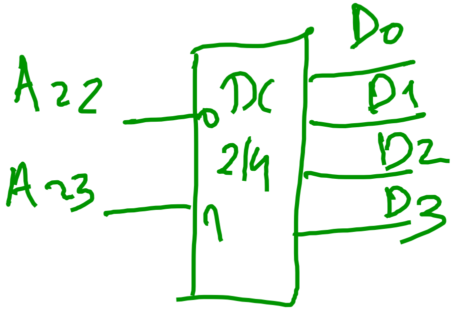
→ HW. 2 submit tonight

→ Office hours

→ 8-bit

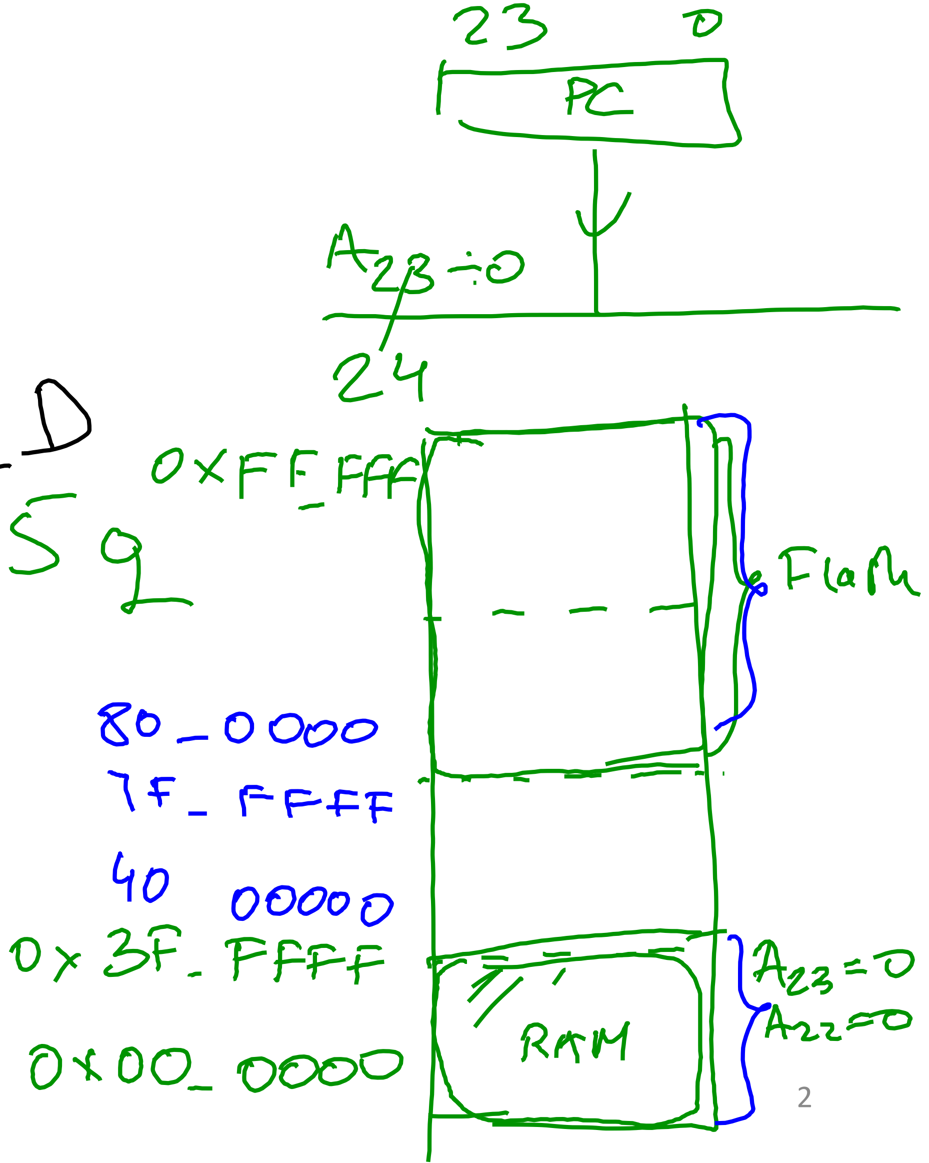
$0001\_1101_b = 0\times1D = 035_8$

2^{24}



00 11_1111_1111_1111_1111_1111

0000_0000_0000_0000_0000_0000



(a) 0x 1116 $\underbrace{\text{MOV}}_{\text{Move word}}$ $\underbrace{6(R5)}_{\text{Indexed}}$, R7 ← register direct ; $R7 \leftarrow \underline{M[6+R5]}$

			Ad B/W As	
1116	4	5	0001	7
1118	0	0	0	6

$EA_S = 6 + R5$

$= 6 + F002$

$= 0xF008$

$R5 = 0x\,F002$

$R7 \leftarrow M[0xF008]$

$= 0xF0\underline{14}$

MOV.B 6(R5), R7

(i) MOV #41 R7 ; R7 ← 41 (0x0029)

s-reg

4	∅	0011	7
0	0	2	9

$$\underbrace{\text{ADD.B}}_{\text{Addition}}\ \underbrace{4(R5)}_{\text{Indexed}},\ \underbrace{6(R6)}_{\text{Indexed}}\ ;\ M[\underbrace{6+R6}_{EA_d}] \leftarrow M[6+R6] + M[\underbrace{4+R5}_{EA_s}]$$

$R6 = 0x0401$

$R5 = 0xC006$

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

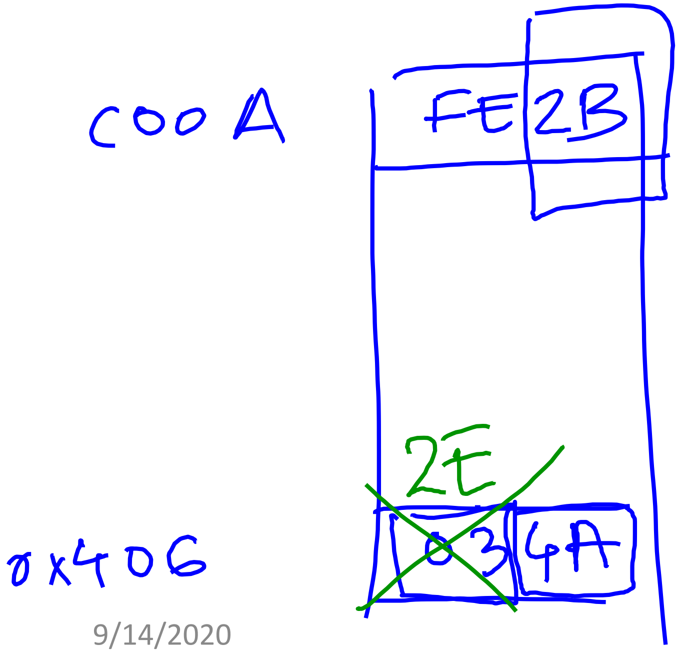
$$EA_d = 6 + R6 = 6 + 0x0401 = 0x0407$$

source operand: $M[EA_s] = M[C00A] = 0x2B$

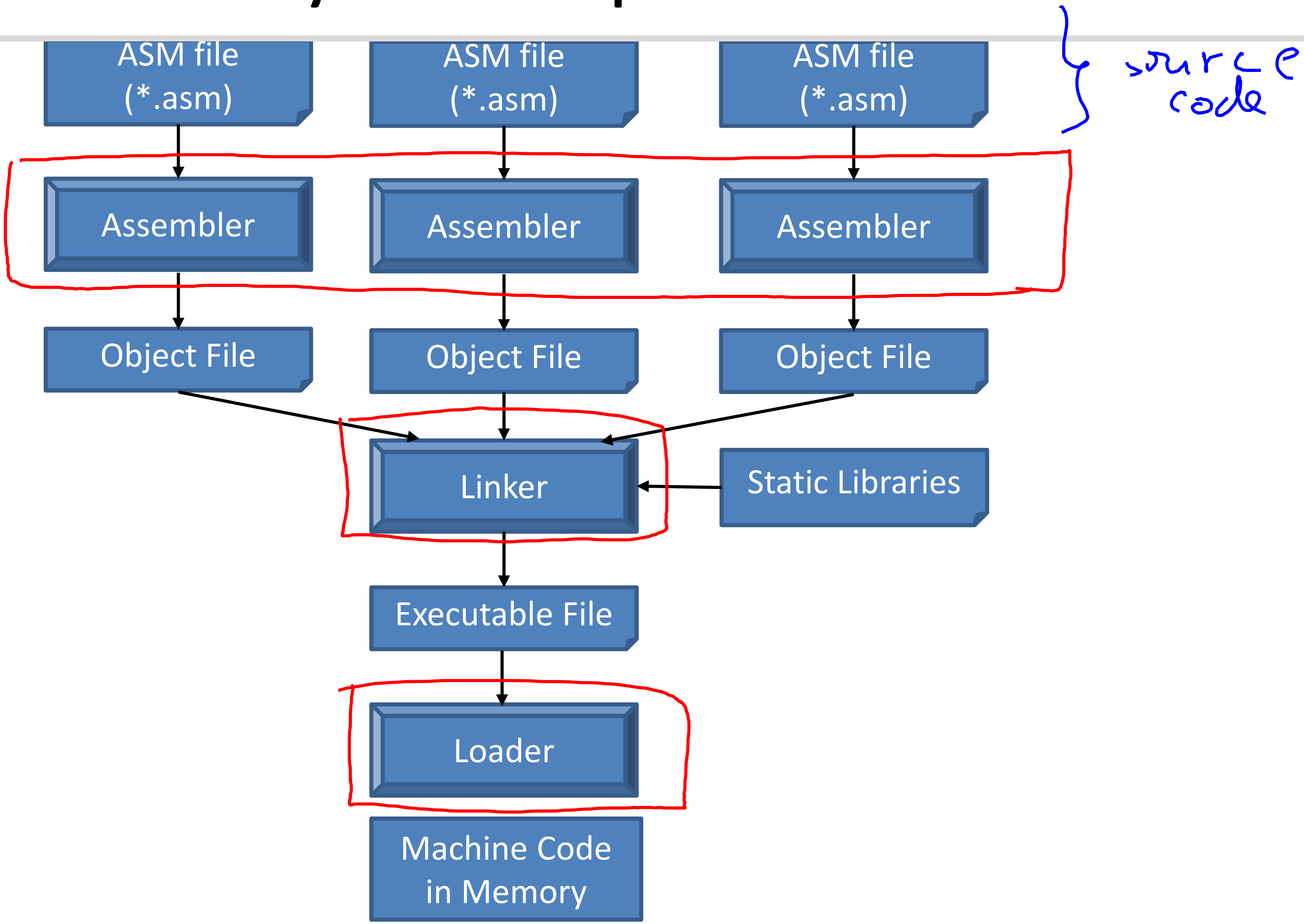
source / dest : $M[EA_d] = M[0407] = 0x03$

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

$C=0$
$V=0$
$N=0$
$Z=0$



Assembly Development Flow

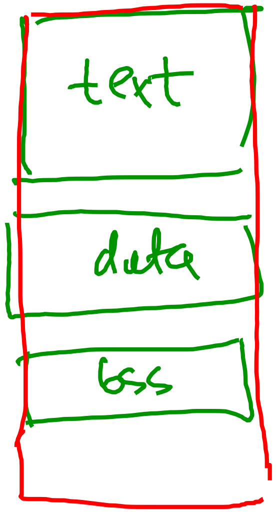


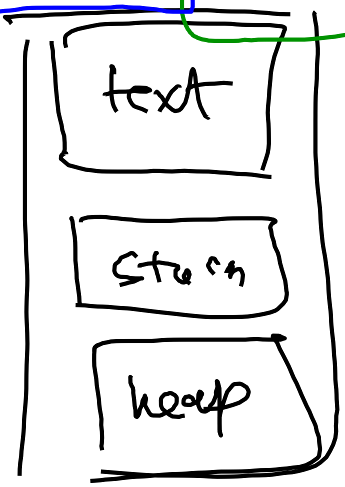
Assembly Language Directives

- 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

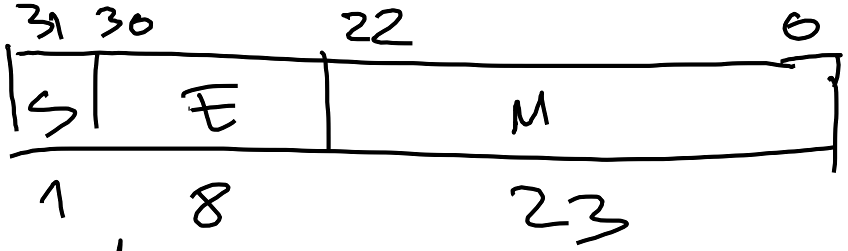
Description	ASM430 (CCS)	A430 (IAR)
Reserve size bytes in the uninitialized sect.	.bss	-
Assemble into the initialized data section	.data	RSEG const
Assemble into a named initialized data sect.	.sect	RSEG
Assemble into the executable code	.text	RSEG code
Reserve space in a named (uninitialized) section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





Constant Initialization Directives

- .byte
- .float
- .word ~ 16-bit
- .long ~ 32-bit
- .string — strings



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
```

synonim

octal

01010011

5 3

symbolic

mov.b b1, r7; r7←M[b1]

r7=0x0005

mov.b 0x3100, r7;

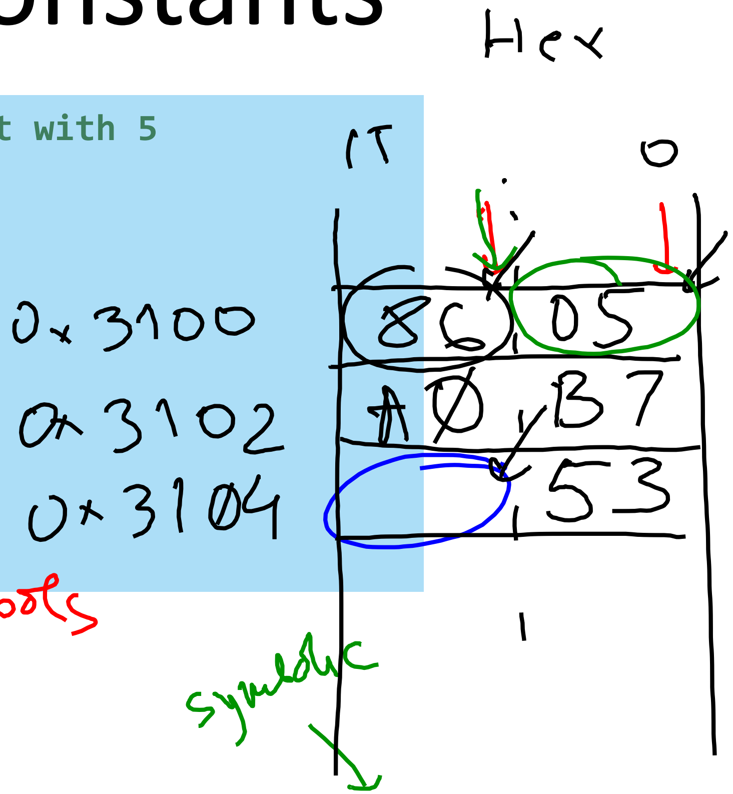


Table of symbols

b1	0x3100
b2	0x3101
b3	0x3102
b4	0x3103
b5	0x3104
tf	0x0019

mov.w b1, r8; r8←8605

mov.w #b1, r9;

r9←0x3100

mov.b b1+3, r10; r10← M[3103]

r10=0x00A0

Directives: Dealing with Constants

.align 2

```
...
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
```

100,000 → 0001 86A0

upper | lower

$21 / 16 = 1 \quad 15$

$1 / 16 = 0 \quad 1$

$21_{10} = 15_{16}$

0015

1st compl: FFEA

+1

FFEB

Table of symbols

...	
w1	0x3106
w2	0x3108
w3	0x310A
dw1	0x310C
dw2	0x3110

w1	3106	0015
w2	3108	FFEB
w3	310A	0019
dw1	310C	86A0
	310E	0001
dw2	3110	FFEA
	3112	FFFF
	→ 3114	

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
```

NULL

- .string "abcd" → 4 bytes
- .cstring "abcd" → 5 bytes

	'B'	'A'
s1 0x3114	42	41
0x3116	44	43
s2 0x3118	42	41
0x311A	44	43
		00

NULL

TOS

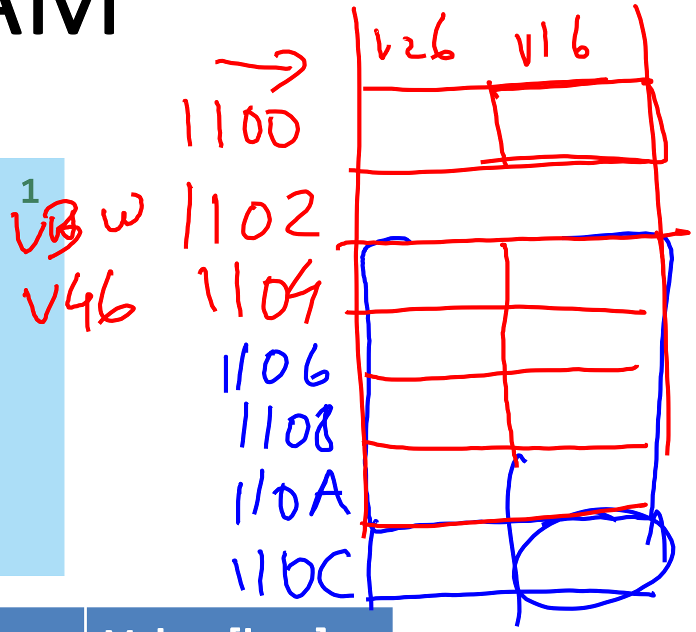
- - —	
s1	0x3114
s2	0x3118

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
```



v26 v16

Label	Address	Memory[15:8]	Memory[7:0]
v1b	0x 1100	--	--
v3w	0x 1102	--	--
v4b		--	--
		--	--
		--	--
		--	--
vx			

Symbol	Value [hex]
v1b	0x 1100
v2b	0x 1101
v3w	0x 1102
v4b	0x 1104
vx	0x 110C

mov.b V16, r7 ;

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`

0 0 0 0 0 1 0 0

Binary ADD

68AC F0AC

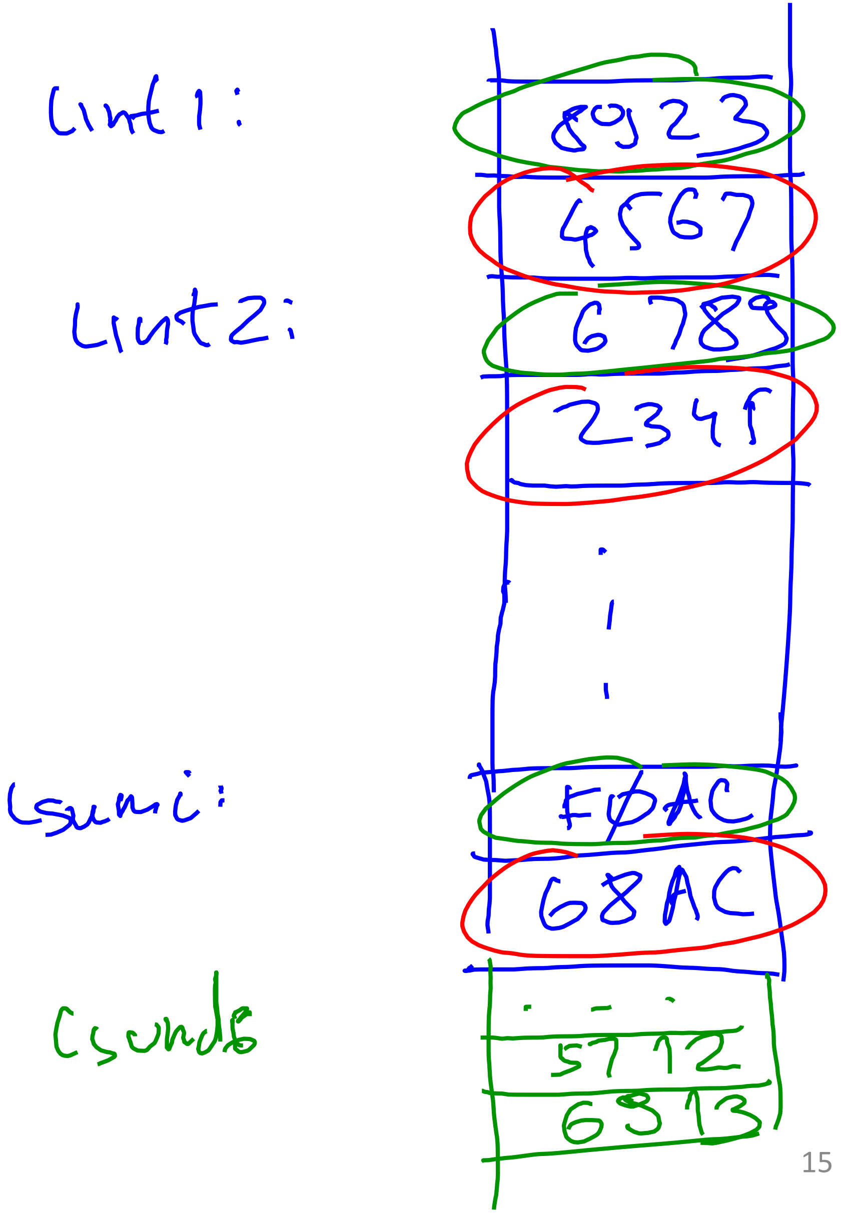
1 1 1 1 1 1

0x45678923

+ 0x23456789

Decimal addition

69135712



Allocate Space & Start Program

```
                                  bic #1, R2
clc
mov.w    lint1, R8              ; R8 ← M[lint1]
dadd.w   lint2, R8              ; R8 ← R8 + M[lint2] + C
                                          decimal
mov.w    R8, lsumd              ; M[lsumd] ← R8

mov.w    lint1+2, R8            ; R8 ← M[lint1+2]
dadd.w   lint2+2, R8            ; R8 ← R8 + M[lint2+2] + C
mov.w    R8, lsumd+2            ; M[lsumd+2] ← R8
```

Main Code (Ver. 1)

```
mov.w    &int1, R8
add.w    &int2, R8      ;   R8 ← R8 + M[&int2]
mov.w    R8, &sumi
mov.w    &int1+2, R8
addc.w   &int2+2, R8
mov.w    R8, &sumi+2
```

Main Code (Ver. 2)

```
      mov.w   #Lint1, R4
      mov.w   #Lsumd, R8
      mov.w   #2, R5       ; R5 is step counter
      clr     R10          ; R10 = 0
lda:  mov.w   4(R4), R7    ; R7 = 0x6789
      mov.w   R10, R2      ; ?
                                6789 + 0x8923
->    dadd.w  @R4+, R7     ; R7 ← R7 + M[R4]; R4 ← R4 + 2
      mov.w   R2, R10      ;
      mov.w   R7, 0(R8)
->    dec.w   R5           ; R5 ← R5 - 1
      jnz     lda
```

R4 → Lint1 (+4 → Lint2)

R8 → Lsumd

Lint1: F000 0x8923

F002 0x4567

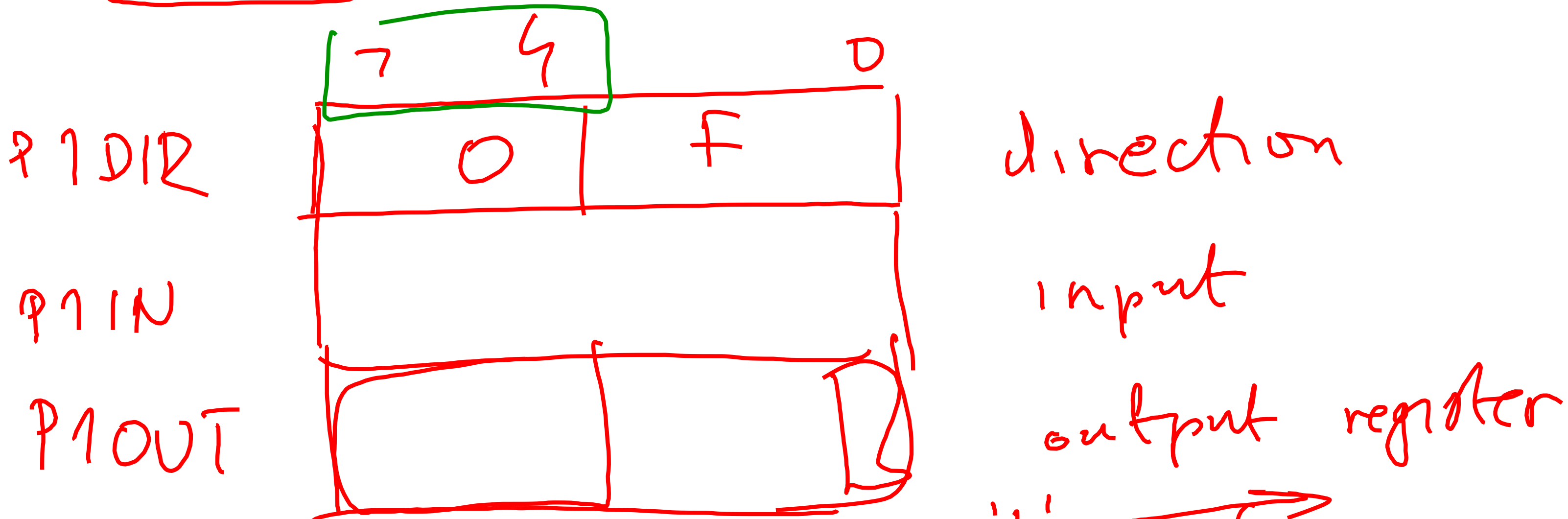
Lint2: F004 0x6789

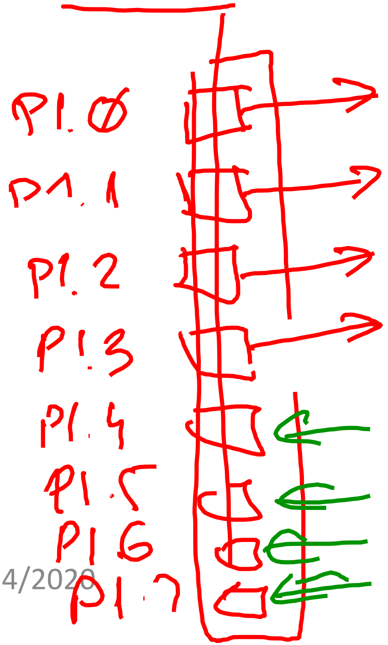
F006 0x2345

Lsumd:

Count Characters 'E' in a String

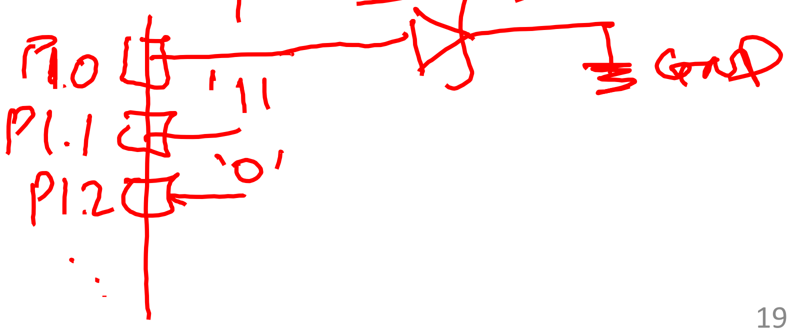
- Write an assembly program that processes an input string to find the number of characters 'E' in the string
- The number of characters is "displayed" on the port 1 of the MSP430





P1DIR = 0x FF

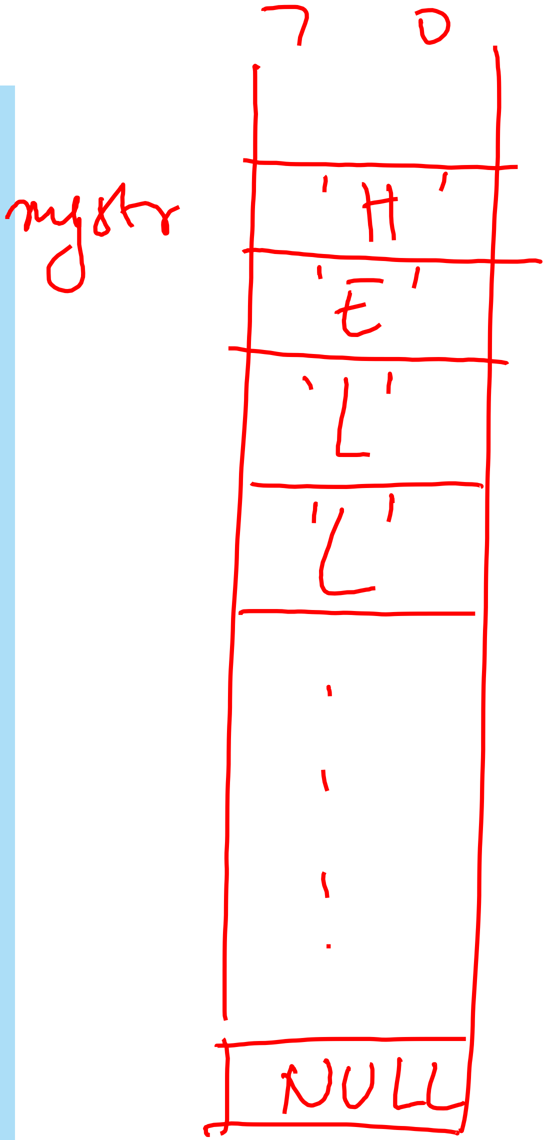
P1OUT = 0x 03



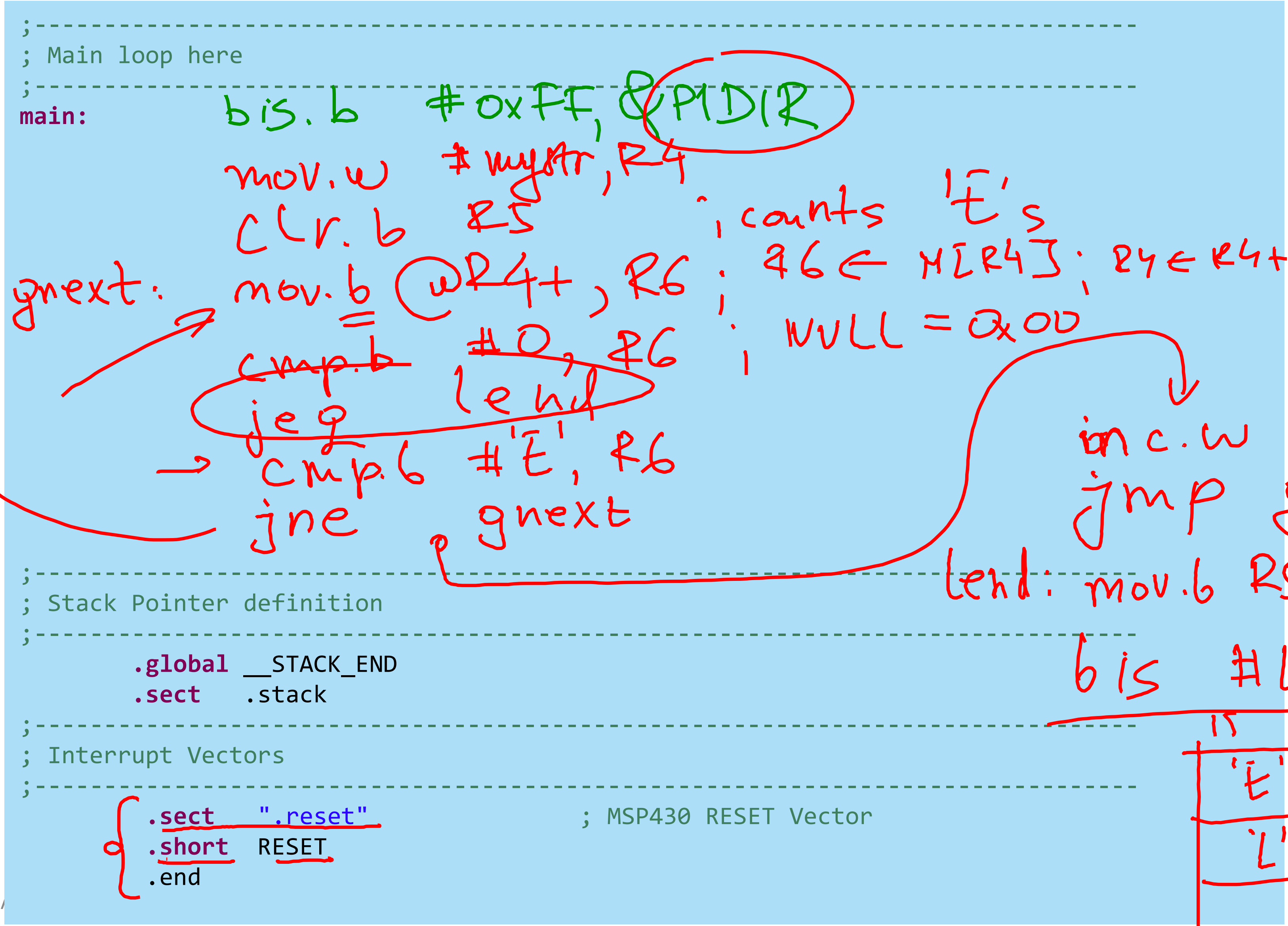
Count Characters ‘E’ in a String

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

.cstring "HELLO - - - !"

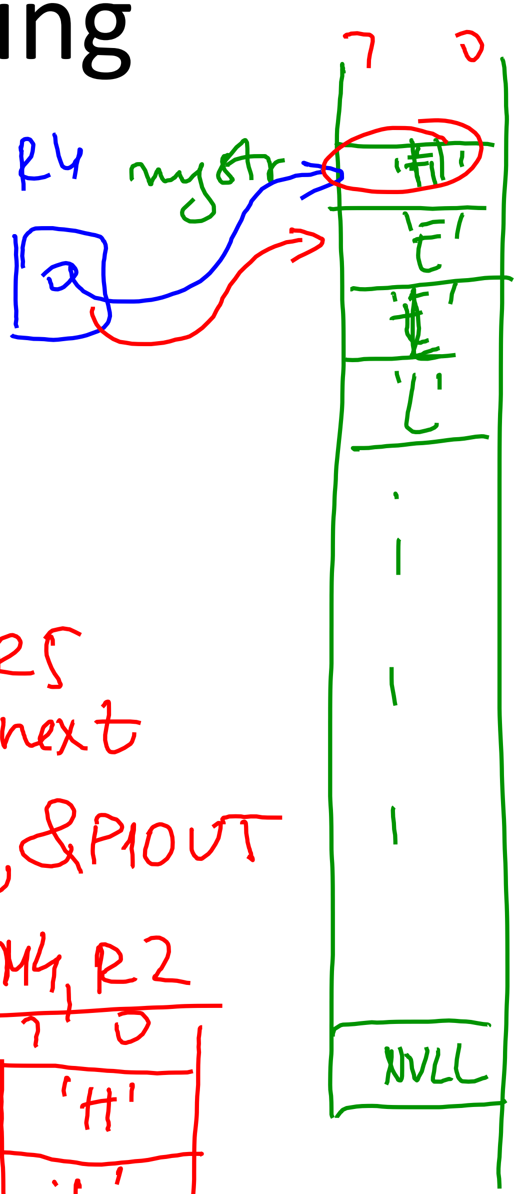


Count Characters ‘E’ in a String



```
;-------------------------------------------------------------------
; Main loop here
;-------------------------------------------------------------------
main:     bis.b   #0xFF, &P1DIR
          mov.w   #myStr, R4
          clr.b   R5              ; counts 'E's
gnext:    mov.b   @R4+, R6        ; R6 ← M[R4]; R4 ← R4+1
          ~~cmp.b #0, R6~~        ; NULL = 0x00
          jeq     lend
          cmp.b   #'E', R6
          jne     gnext
          inc.w   R5
          jmp     gnext
lend:     mov.b   R5, &P1OUT
          bis     #LPM4, R2
;-------------------------------------------------------------------
; Stack Pointer definition
;-------------------------------------------------------------------
          .global __STACK_END
          .sect   .stack
;-------------------------------------------------------------------
; Interrupt Vectors
;-------------------------------------------------------------------
          .sect   ".reset"                 ; MSP430 RESET Vector
          .short  RESET
          .end
```

15 8	7 0
'E'	'H'
'L'	'L'



7
0
R4
myStr
a
'H'
'E'
'E'
'L'
NULL