 **EEL4744**

# Menu

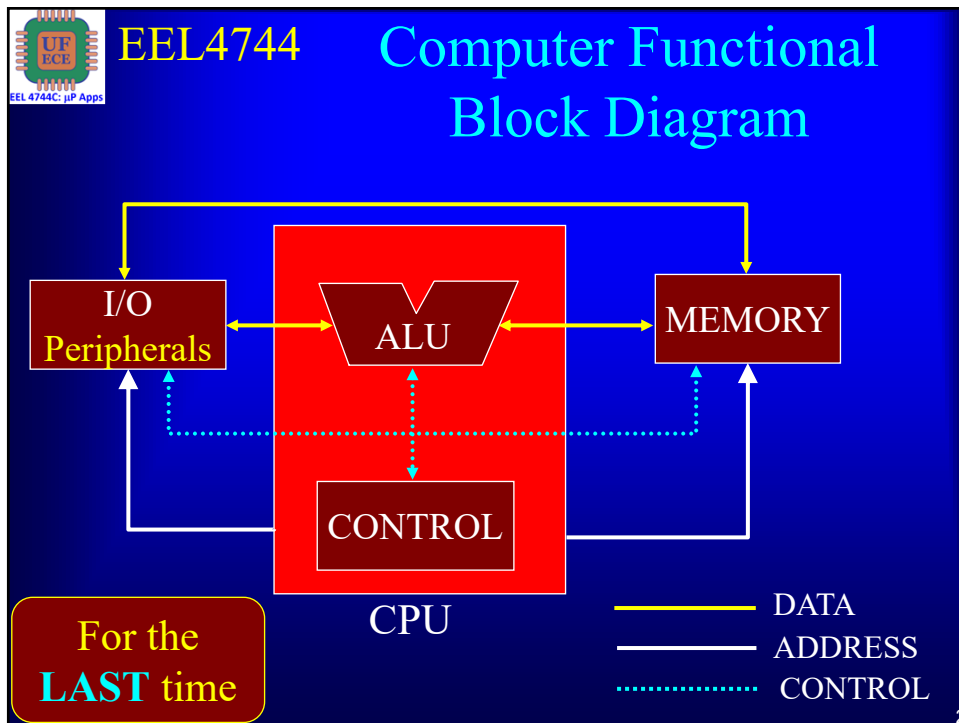
- Big Picture
- Assembler Directives
- Examples using:
  - > Debugging/Simulating in Microchip (Atmel) Studio
  - > Downloading and Debugging/Emulating with the UF-board

*Look into my ...*

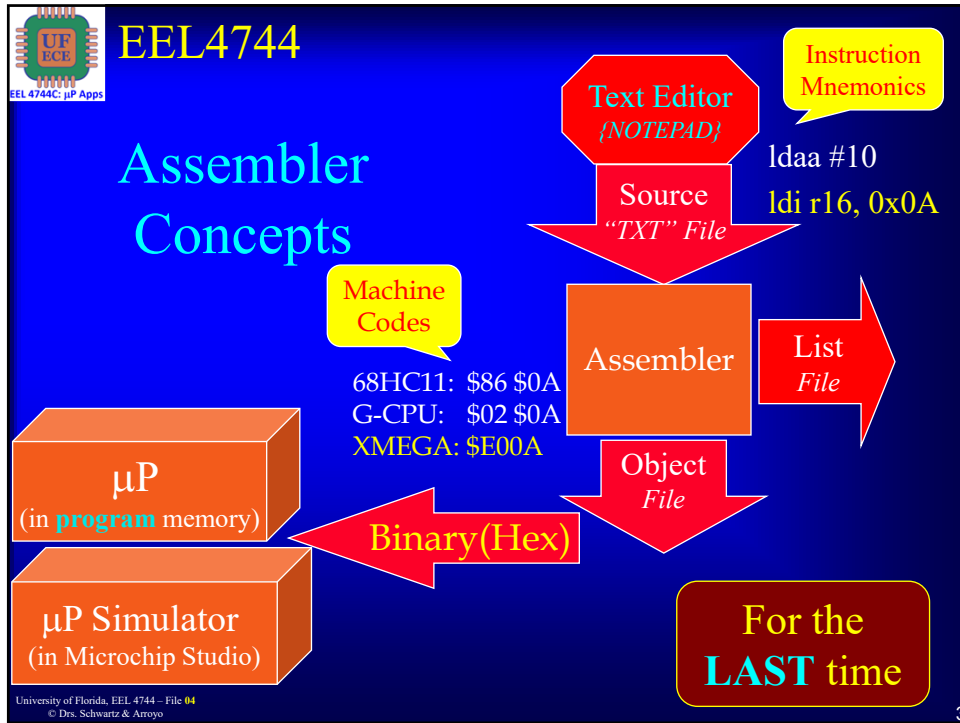
See on  
web-site: [GCPU\\_to\\_XMEGA.pdf](#),  
Examples: [Table\\_Load\\_Example.asm](#),  
[GPIO\\_Output.asm](#),  
[doc0856\\_AVR\\_Instruction\\_Set.pdf](#)

University of Florida, EEL 4744 – File 04  
© Drs. Schwartz & Arroyo

1




2



3

- 
- Assembly Control
    - >ORG Origin program counter
  - Symbol Definition
    - >EQU Assign permanent value
  - Data Definition/Storage Allocation
    - >DC.B Define constant byte
    - >DC.W Define constant word
    - >DS.B Define storage bytes
    - >DS.W Define storage word
    - >Old (alternate) form of above
      - FCB Form constant byte (similar to DC.B)
      - FCC Form constant character string (similar to DC.B)
      - FDB Form constant double byte (similar to DC.W)
      - RMB Reserve memory; single bytes (similar to DS.B)

4



# EEL4744

## Atmel Assembler Directives


AVR Assembler Manual (Section 5 has Assembler Directives)

Also, see [http://www.avr-asm-tutorial.net/avr\\_en/beginner/DIREXP.html](http://www.avr-asm-tutorial.net/avr_en/beginner/DIREXP.html)

| Directive             | Description                          |
|-----------------------|--------------------------------------|
| <b>BYTE</b>           | Reserve byte(s) to a variable.       |
| <b>CSEG</b>           | Code Segment                         |
| <b>CSEGSIZE</b>       | Program memory size                  |
| <b>DB</b>             | Define constant byte(s)              |
| <b>DEF</b>            | Define a symbolic name on a register |
| <b>DSEG</b>           | Data Segment                         |
| <b>DW</b>             | Define Constant word(s)              |
| <b>ENDM, ENDMACRO</b> | EndMacro                             |
| <b>EQU</b>            | Set a symbol equal to an expression  |
| <b>ESEG</b>           | EEPROM Segment                       |
| <b>EXIT</b>           | Exit from file                       |
| <b>INCLUDE</b>        | Read source from another file        |
| <b>LIST</b>           | Turn listfile generation on          |
| <b>LISTMAC</b>        | Turn Macro expansion in list file on |
| <b>MACRO</b>          | Begin Macro                          |
| <b>NOLIST</b>         | Turn listfile generation off         |
| <b>ORG</b>            | Set program origin                   |

University of Florida, EEL 4744 – File 04  
© Drs. Schwartz & Arroyo

5



# EEL4744

## More Atmel Assembler Directives

AVR Assembler Manual (Section 5 has Assembler Directives)

Also, see [http://www.avr-asm-tutorial.net/avr\\_en/beginner/DIREXP.html](http://www.avr-asm-tutorial.net/avr_en/beginner/DIREXP.html)

| Directive                | Description                   |
|--------------------------|-------------------------------|
| <b>SET</b>               | Set a symbol to an expression |
| <b>ELSE,ELIF</b>         | Conditional assembly          |
| <b>ENDIF</b>             | Conditional assembly          |
| <b>ERROR</b>             | Outputs an error message      |
| <b>IF,IFDEF,IFNDEF</b>   | Conditional assembly          |
| <b>MESSAGE</b>           | Outputs a message string      |
| <b>DD</b>                | Define Doubleword             |
| <b>DQ</b>                | Define Quadword               |
| <b>UNDEF</b>             | Undefine register symbol      |
| <b>WARNING</b>           | Outputs a warning message     |
| <b>OVERLAP/NOOVERLAP</b> | Set up overlapping section    |

University of Florida, EEL 4744 – File 04  
© Drs. Schwartz & Arroyo

6




**EEL4744**      ORG: Assembler Directive  
(Pseudo-instruction, **Atmel**)

- **ORG (Origin)**: It can be used to alter the location counter by setting it to any memory location in memory.
  - > Note that the **SRAM and EEPROM location counters count bytes.**
  - > Note that the **Program Memory location counter counts words.**

University of Florida, EEL 4744 – File 04  
© Drs. Schwartz & Arroyo

7



**EEL4744**      ORG: Assembler Directive  
(Pseudo-instruction, **Atmel**)

- **ORG (Origin)**: It can be used to alter the location counter by setting it to any location in memory.

Syntax:

```
.ORG expression      ; where operand is a 16-bit address or an
                    ; expression that evaluates to a 16-bit address
```


Example:

```
.DSEG ; Start data segment
.ORG 0x2000      ; Set SRAM address to 0x2000
Total: .BYTE 1   ; Reserve a byte at SRAM address 0x2000

.CSEG
.ORG 0x0200     ; Set Program Memory address to 0x200
MAIN: ldi r16, 0xF ; Do something
```

University of Florida, EEL 4744 – File 04  
© Drs. Schwartz & Arroyo

8



**EEL4744**      **EQU: Assembler Directive  
(Pseudo-instruction, **Atmel**)**

- **EQU (Equate)**: Set a symbol equal to an expression
  - > The EQU directive assigns a value to a label.
  - > This label can then be used in later expressions.

Syntax:

```
.EQU label = expression ; where operand is a value or an expression
                        ; that evaluates to a value
```

Example:

```
.EQU BestNo = $37 ; BestNo will be replaced by $37
.EQU Table_Size = 10*BestNo ; Set the table size here
```

University of Florida, EEL 4744 – File 04  
© Drs. Schwartz & Arroyo

9




**EEL4744**      **DB: Assembler Directive  
(Pseudo-instruction, **Atmel**)**


- **DB (Define Constant Byte in program memory and EEPROM)**: It allocates space in memory and also initializes memory locations to specified values at the time of assembly
  - > The DB directive reserves memory resources in the program memory or the EEPROM memory
    - **NOT for data memory and SRAM**
  - > The DB directive must be placed in a **Code Segment** or an **EEPROM Segment**

University of Florida, EEL 4744 – File 04  
© Drs. Schwartz & Arroyo

10



## EEL4744




### DB: Assembler Directive (Pseudo-instruction, **Atmel**)

- If the DB directive is given in a Code Segment and the expression list contains more than one expression, the expressions are packed so that **TWO BYTES are placed in each PROGRAM MEMORY WORD**
  - >If the expression list contains an odd number of expressions, the last expression will be placed in a program memory word of its own, even if the next line in the assembly code contains a DB directive
    - The unused half of the program word is set to zero
    - A warning is given, in order to notify the user that an extra zero byte is added to the .DB statement

University of Florida, EEL 4744 – File 04  
© Drs. Schwartz & Arroyo

11



## EEL4744

### DB: Assembler Directive (Pseudo-instruction, **Atmel**)

**Syntax:**  
 (label:) .DB operand ; where operand is an 8-bit value, a list of  
 ; bytes, or an expression that evaluates to an 8-bit value

**Examples (see .lss file):**

```


.ORG 0x100 ; This is in the Code Segment (Program Memory)
.DB 37, 0x73, 0xF1, 0 ; (0x200) = 37 = 0x25, (0x201) = 0x73
; (0x202) = 0xF1, (0x203) = 0
GSmrt: .DB 0x99 ; (GSmrt) = (0x204) = 0x99
; Note that since there is only one byte above, (0x205) = 0
Tab2: .DB 3, 9, 44, 0x2E, 244, 0xCD ; Tab2 = 0x206
EOT: .DB 0xFF ; EOT = End of Table
Mesg: .DB "3744 is the 'best class' ever!" ; Text strings ok too
    
```

This is a  
memory  
snapshot

|              |    |    |    |    |    |           |    |    |                 |
|--------------|----|----|----|----|----|-----------|----|----|-----------------|
| <b>0x200</b> | 25 | 73 | f1 | 00 | 99 | <b>00</b> | 03 | 09 | %sñ.™...        |
| 0x208        | 2c | 2e | f4 | cd | ff | 00        | 33 | 37 | ,,δİy.37        |
| 0x210        | 34 | 34 | 20 | 69 | 73 | 20        | 74 | 68 | <b>44 is th</b> |
| 0x218        | 65 | 20 | 27 | 62 | 65 | 73        | 74 | 20 | <b>e 'best</b>  |
| 0x220        | 63 | 6c | 61 | 73 | 73 | 27        | 20 | 65 | <b>class' e</b> |
| 0x228        | 76 | 65 | 72 | 21 | ff | ff        | ff | ff | <b>ver!ÿÿÿÿ</b> |

University of Florida, EEL 4744 – File 04  
© Drs. Schwartz & Arroyo

12



EEL4744

doc0856\_AVR\_Instruction\_Set.pdf

## Atmel Instructions


- Instruction Set Nomenclature
- I/O Registers (later)
- The Program and Data Addressing Modes
- Conditional Branch Summary
- Complete Instruction Set Summary
  - > Explore instructions
- See LD, for example, in instruction manual:
  - > Find “LD – Load Indirect from Data Space to Register”

**NOTE:** Our device, the ATxmega128A1U is an AVRxm in the doc0856, Table 4-2: *Instruction Set Summary*

University of Florida, EEL 4744 – File 04  
© Drs. Schwartz & Arroyo

13

13



EEL4744

doc0856\_AVR\_Instruction\_Set.pdf

## LDI – Load Immediate

### Examples:

```
ldi r16, 27 ; Load 27 (0x1B) into register r16, r16 ← $1B
ldi r17, 0x34 ; Load 0x34 into register r17, r17 ← $34

clr r31 ; Clear Z high byte, r31 ← 0
ldi r30, $F0 ; Set Z low byte to $F0 (0x and $ are hex prefixes)
; r30 ← $F0

lpm r17, Z ; Load constant from Program memory pointed
; to by Z, r17 ← (Z)

lpm ; Load constant from Program memory pointed
; to by Z (notice with no operand, the default is r0),
; r0 ← (Z)


lpm r18, Z+ ; r18 ← (Z), Z++ [Z++ means Z ← Z+1]

ldi ZL, low(Table<<1) ; Load ZL with low address of Table
ldi ZH, high(Table<<1) ; Load ZH with high address of Table
```

University of Florida, EEL 4744 – File 04  
© Drs. Schwartz & Arroyo

14

14



EEL4744 doc0856\_AVR\_Instruction\_Set.pdf

## LD – Load Indirect from Data Space to Register

**Examples:**

```


clr r29 ; Clear high byte of Y (Y = r29 | r28),  $r29 \leftarrow 0$ 
ldi r28,$37 ; Set low byte of Y to $37 (0x and $ are hex prefixes),  $r28 \leftarrow \$37$ 
ld r0,Y+ ; Load r0 with data at address $37 (Y post increment),
;  $r0 \leftarrow (Y), Y++$ 
ld r1,Y ; Load r1 with data at address $38,  $r1 \leftarrow (Y)$ 
ldi r28,$42 ; Set low byte of Y to $42,  $r28 \leftarrow \$42$ 
ld r2,Y ; Load r2 with data at address $42 (since r29 = 0),  $r2 \leftarrow (Y)$ 
ld r3,-Y ; Load r3 with data at address $41 (Y pre decrement)
;  $Y--, r3 \leftarrow (Y)$  [Y-- means  $Y \leftarrow Y-1$ ]
ldd r4,Y+2 ; Load r4 with data at address $43,  $r4 \leftarrow (Y+2)$ 

```

- Note that for loading with Y and Z, the **ldd** instruction lets you add up to 63 to Y or Z (%11 1111 = 63, i.e., a 6-bit post increment)
- There is **NO ldd** with X

University of Florida, EEL 4744 – File 04  
© Drs. Schwartz & Arroyo

15



EEL4744 doc0856\_AVR\_Instruction\_Set.pdf

## LPM – Load Program Memory

**Examples:**

```

.org 0x100
Table_1: .dw 0x3744 ; 0x44 is addresses when ZLSB = 0
; 0x37 is addresses when ZLSB = 1
...
ldi ZH, high(Table_1<<1) ; Initialize Z-pointer
ldi ZL, low(Table_1<<1)

lpm r17, Z+ ; Load constant from Program memory pointed
; to by Z (r31:r30),  $r17 \leftarrow (Z); Z++ (r17 \leftarrow 0x44)$ 
lpm ; Load constant from Program memory pointed
; to by Z (notice with no operand, the default is r0),
;  $r0 \leftarrow (Z)$ 
lpm r18, Z ;  $r18 \leftarrow (Z) (r18 \leftarrow 0x37)$ 

```

University of Florida, EEL 4744 – File 04  
© Drs. Schwartz & Arroyo

16






## EEL4744 Machine Language and Program Assembly

- An *assembly language program* is also called *source code*
- An *assembly language program* has to be transformed into a *machine language program*, also called *object code*.
- This transformation process, called *program assembly*, can be done automatically by a computer program called an *assembler*
  - > Usually built into the IDE (Integrated Development Environment), e.g., Microchip (Atmel) Studio
- The machine language program is simply a *coded version* of the assembly language program, with each machine language instruction corresponding to an assembly language instruction.

University of Florida, EEL 4744 – File 04  
© Drs. Schwartz & Arroyo

17



## EEL4744 XMEGA Assembler Prefix, Suffix, Operators


- XMEGA assembler
  - > By default, numbers are assumed to be **decimal**

| Base    | Prefix                |
|---------|-----------------------|
| Hex     | 0x (or \$)            |
| Octal   | 0 (leading zero only) |
| Binary  | 0b                    |
| Decimal | Nothing               |

| Operator Symbol | Operation                  |
|-----------------|----------------------------|
| !               | Logical Not                |
| ~               | Bitwise Not                |
| -               | Unary Minus                |
| *               | Multiplication             |
| /               | Divide                     |
| %               | Remainder after division   |
| +               | Addition                   |
| -               | Subtraction                |
| <<              | Shift Left                 |
| >>              | Shift right                |
| <, <=           | Less than (or equal to)    |
| >, >=           | Greater than (or equal to) |
| ==, !=          | Equal, not equal           |
| &,              | Bitwise And (Bitwise OR)   |
| ^               | Bitwise XOR                |
| &&,             | Logical AND (Logical OR)   |

University of Florida, EEL 4744 – File 04  
© Drs. Schwartz & Arroyo

18




## EEL4744

# XMEGA Assembler Functions

- **LOW**(expression) returns the low byte of an expression
- **HIGH**(expression) returns the second byte of an expression
- **BYTE1**(expression) is the same function as LOW
- **BYTE2**(expression) is the same function as HIGH
- **BYTE3**(expression) returns the third byte of an expression
- **BYTE4**(expression) returns the 4<sup>th</sup> byte of an expression
- **ABS**() Returns the absolute value of a constant expression
- **STRLEN**(string) returns the length of a string constant, in bytes
- **LWRD**(expression) returns bits 0-15 of an expression
- **HWRD**(expression) returns bits 16-31 of an expression
- **EXP2**(expression) returns  $2^{\text{expression}}$
- **LOG2**(expression) returns the integer part of  $\log_2(\text{expression})$

University of Florida, EEL 4744 – File 04  
© Drs. Schwartz & Arroyo

19



## EEL4744


# Using the Microchip (Atmel) Studio Simulator

`Table_Load_Example.asm`

- Demo the simulator with **Table\_Load\_Example.asm**
  - > Assemble (F7)
  - > Single step (F11), Continue [Run] (F5), Run to Cursor (Ctrl-F10)
  - > Run with breakpoints
    - Add breakpoint by clicking on line number (in asm file) [creates a red dot]
      - Select **Debug | Windows | Breakpoints** to be able to see and remove breakpoints
  - > To see the address for the instructions (while simulating/emulating),
    - Debug | Windows | Disassembly** (or use Alt-8)
    - Go to address 0x100 (for program memory) and 0x200 for program
  - > See **Solution Explorer** window
    - Look at **Dependencies | ATxmega128A1Udef.inc** (from .include in source)
    - Look at **File | Open | File** (or in **Solution Explorer** see **Output Files**)
      - List file: **Debug/Table\_Load\_Example.lst**
      - Object file: **Debug/Table\_Load\_Example.hex** (human readable; compare to **.obj**)
      - Map file: **Debug/Table\_Load\_Example.map** (see end of file)

University of Florida, EEL 4744 – File 04  
© Drs. Schwartz & Arroyo

20




## EEL4744 Using uPAD Board and Microchip (Atmel) Studio Emulator

- Find and select the Simulator menu on top right
  - > Select **E**mulator **D**e**Bu**Gger/programmer | **E**DBG...
- All controls for Emulator are the same as the Simulator Assemble (F7)
  - > Single step (F11), Continue [Run] (F5), Run to Cursor (Ctrl-F10)
  - > Run with breakpoints
- Just do it
  - > Demo single step, run, breakpoint?
  - > Show memory and registers
- Now run **GPIO\_Output** (but do **NOT** look at too closely)
  - > Will see this file again in Lecture 6: GPIO

Table\_Load\_Example.asm  
GPIO\_Output.asm

University of Florida, EEL 4744 – File 04  
© Drs. Schwartz & Arroyo

21

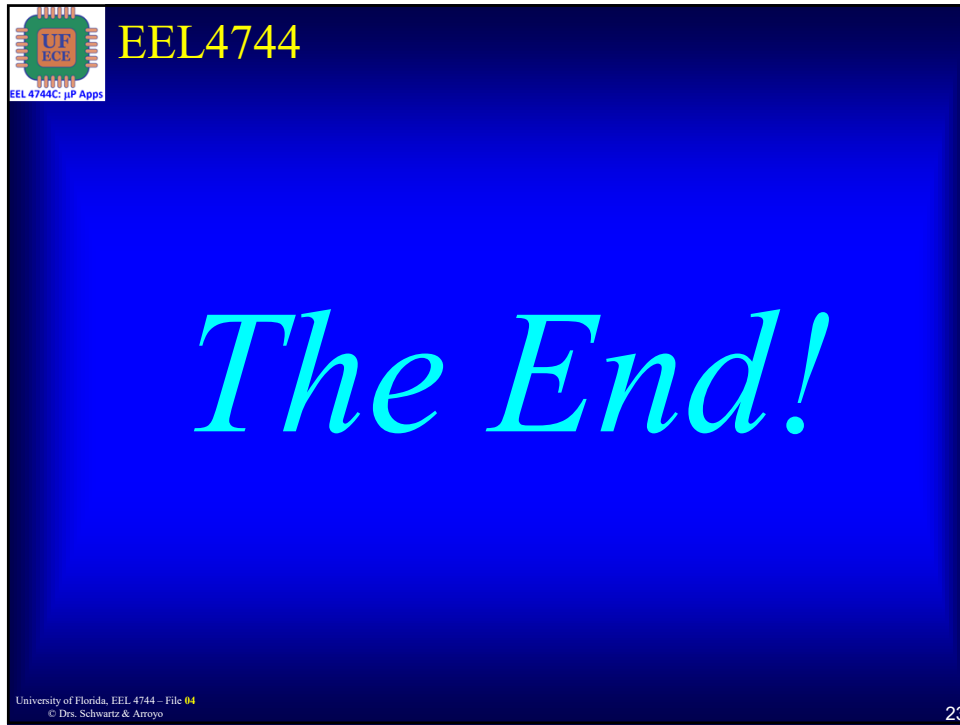


## EEL4744 Assembler Macros

- A macro (macroinstruction) is a rule that converts a short stream of characters into a set of instructions.
- Macros are used to make a sequence of computing instructions available to the programmer as a single program statement
  - > Programming with macros may be less tedious
  - > Programming with macros may be less error-prone
- See the following for info about creating macros:
  - > <http://www.rjhcoding.com/avr-asm-macros.php>
  - > <http://www.avrbeginners.net/asm/asm/macros.html>
  - > <https://tinyurl.com/y3w8bbhv>

University of Florida, EEL 4744 – File 04  
© Drs. Schwartz & Arroyo

22



23