

EEL3744

Menu

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

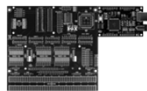


See on

web-site: GCPU_to_XMEGA.pdf,
 Examples: Table_Load_Example.asm,
 GPIO_Output.asm,
 doc0856_AVR_Instruction_Set.pdf

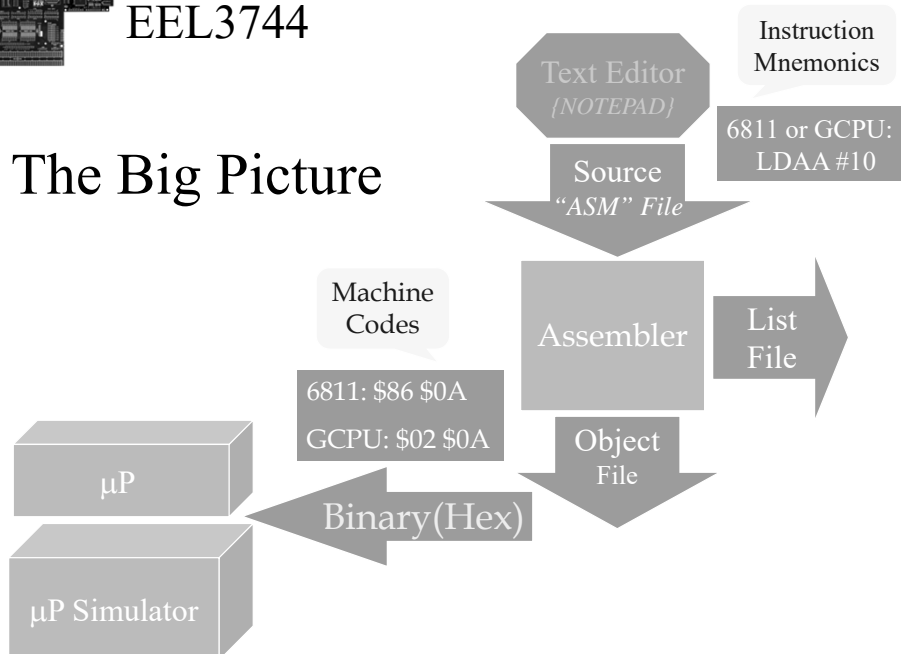
University of Florida, EEL 3744 – File 04
 © Drs. Schwartz & Arroyo

1



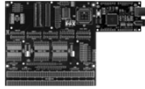
EEL3744

The Big Picture



University of Florida, EEL 3744 – File 04
 © Drs. Schwartz & Arroyo

2

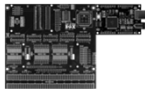


EEL3744 Common Assembler Directives (Pseudo-instruction, GCPU)

- 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)

University of Florida, EEL 3744 – File 04
© Drs. Schwartz & Arroyo

3



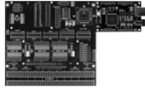
EEL3744 Atmel Assembler Directives

In Atmel Studio, see
Help, then *AVR
Assembler*, and then
Assembler directives

Also, see
http://www.avr-asm-tutorial.net/avr_en/beginner/DIREXP.html

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

University of Florida, EEL 3744 – File 04
© Drs. Schwartz & Arroyo



EEL3744

Atmel Assembler Directives

See doc1022 AVR
Assembler User
Guide

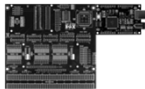
In Atmel Studio, see
Help, then *AVR
Assembler*, and then
Assembler directives

Also, see
http://www.avr-asm-tutorial.net/avr_en/beginner/DIREXP.html

Directive	Description
<u>MACRO</u>	<u>Begin Macro</u>
<u>NOLIST</u>	<u>Turn listfile generation off</u>
<u>ORG</u>	<u>Set program origin</u>
<u>SET</u>	<u>Set a symbol to an expression</u>
<u>ELSE,ELIF</u>	<u>Conditional assembly</u>
<u>ENDIF</u>	<u>Conditional assembly</u>
<u>ERROR</u>	<u>Outputs an error message</u>
<u>IF,IFDEF,IFNDEF</u>	<u>Conditional assembly</u>
<u>MESSAGE</u>	<u>Outputs a message string</u>
<u>DD</u>	<u>Define Doubleword</u>
<u>DQ</u>	<u>Define Quadword</u>
<u>UNDEF</u>	<u>Undefine register symbol</u>
<u>WARNING</u>	<u>Outputs a warning message</u>
<u>OVERLAP/NOOVERLAP</u>	<u>Set up overlapping section</u>

University of Florida, EEL 3744 – File 04
© Drs. Schwartz & Arroyo

5



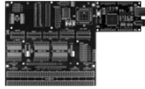
EEL3744

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.
 - > The ORG directive sets the location counter to an absolute value.
 - The value to set is given as a parameter
 - If an ORG directive is given within a **Data Segment**, then it is the **SRAM** location counter which is set
 - If the directive is given within a **Code Segment**, then it is the **Program Memory** counter which is set and if the directive is given within an EEPROM Segment, it is the **EEPROM** location counter which is set
 - > The default values of the Code and the EEPROM location counters are zero, and the default value of the SRAM location counter is the address immediately following the end of I/O address space (0x60 for devices without extended I/O, 0x100 or more for devices with extended I/O) when the assembling is started.
 - > Note that the **SRAM and EEPROM location counters count bytes**.
 - > Note that the **Program Memory location counter counts words**.

University of Florida, EEL 3744 – File 04
© Drs. Schwartz & Arroyo

6



EEL3744

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

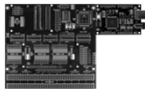
Example:

```
.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 Counter to 0x200
MAIN: ldi r16, 0xF ; Do something
```

University of Florida, EEL 3744 – File 04
© Drs. Schwartz & Arroyo

7



EEL3744

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. A label assigned to a value by the EQU directive is a constant and can not be changed or redefined.
 - > EQU informs the assembler to equate the specified symbol name to the value of the operand.
 - > When the symbolic name is subsequently encountered in the assembly process, the assembler replaces it with the binary value of the corresponding operand.
 - > The operand can be either a value or an expression that can be evaluated. It should be used to improve the clarity and readability of the assembly program.

Syntax:

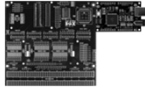
.EQU label = expression ; where operand is a value or an expression
; that evaluates to a value

Example:

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

University of Florida, EEL 3744 – File 04
© Drs. Schwartz & Arroyo

8



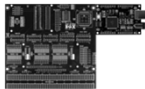
EEL3744

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**
 - > To refer to the reserved locations, the DB directive must be preceded by a label
 - > The DB directive takes a list of expressions, and must contain at least one expression
 - > The DB directive must be placed in a **Code Segment** or an **EEPROM Segment**
- The expression list is a sequence of expressions, delimited by commas
- Each expression must evaluate to a number between -128 and 255

University of Florida, EEL 3744 – File 04
© Drs. Schwartz & Arroyo

9



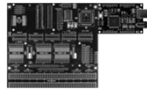
EEL3744

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 3744 – File 04
© Drs. Schwartz & Arroyo

10



EEL3744

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

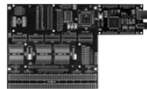
EOT: .DB 0xFF ; EOT = End of Table

Mesg: .DB "3744 is the 'best class' ever!" ; Text strings ok too

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

University of Florida, EEL 3744 – File 04
© Drs. Schwartz & Arroyo

11



EEL3744

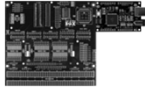
doc0856_AVR_Instruction_Set.pdf

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”

University of Florida, EEL 3744 – File 04
© Drs. Schwartz & Arroyo

12



EEL3744

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)

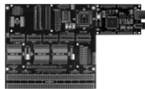
lpm              ; Load constant from Program memory pointed
                  ; to by Z (notice with no operand, the default is r0),
                  ; r0 ← (Z)

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 3744 – File 04
© Drs. Schwartz & Arroyo

13



EEL3744

doc0856_AVR_Instruction_Set.pdf

LD – Load Indirect from Data Space to Register

Examples:

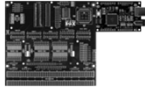
```
clr r29          ; Clear high byte of Y (Y = r29 | r28), r29 ← 0
ldi r28,$37      ; Set low byte of Y to $37 (0x and $ are hex prefixes), r28 ← $37
ld r0,Y+         ; Load r0 with data at address $37 (Y post increment),
                  ; r0 ← (Y), Y++

ld r1,Y          ; Load r1 with data at address $38, r1 ← (Y)
ldi r28,$42      ; Set low byte of Y to $42, r28 ← $42
ld r2,Y          ; Load r2 with data at address $42 (since r29 = 0), r2 ← (Y)
ld r3,-Y         ; Load r3 with data at address $41 (Y pre decrement)
                  ; Y--, r3 ← (Y) [Y-- means Y ← Y-1]
ldd r4,Y+2       ; Load r4 with data at address $43, r4 ← (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 3744 – File 04
© Drs. Schwartz & Arroyo

14



EEL3744

doc0856_AVR_Instruction_Set.pdf

LPM – Load Program Memory

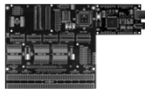
Examples:

```
.org 0x100
Table_1: .dw 0x4744      ; 0x44 is addresses when ZLSB = 0
                        ; 0x47 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 ← (Z); Z++ (r17 ← 0x44)
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) (r18 ← 0x47)
```

University of Florida, EEL 3744 – File 04
© Drs. Schwartz & Arroyo

15



EEL3744

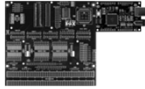
Common Assembler Directives (Pseudo-instruction, DSC)

See spru513,
Table 4-1,
Table 4-2

- Define Sections .bss = better save space ;)
 - > **.bss** Reserves size words in the .bss (uninitialized data) section
 - > **.data** Assembles into the .data (initialized data) section
 - > **.sect** Assembles into a named (initialized) section
 - > **.text** Assembles into the .text (executable code) section
 - > **symbol .usect** “section name”, size in words
- Define Constants (Data and Memory)
 - > **.byte** Initializes 1 or more successive bytes in current section
 - > **.char** Initializes 1 or more successive bytes in current section
 - > **.word** Initializes one or more 16-bit integers
 - > **.int** Initializes one or more 16-bit integers
 - > **.long** Initializes one or more 32-bit integers

University of Florida, EEL 3744 – File 04
© Drs. Schwartz & Arroyo

16



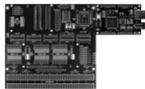
EEL3744

Machine Language and Program Assembly

- A consequence of the stored program concept is that the assembly language instructions must be transformed into a form that can be stored in memory locations, and also that can be processed by the microprocessor. In other words, an **assembly language program**, also called **source code**, 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., 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 3744 – File 04
© Drs. Schwartz & Arroyo

17



EEL3744

XMEGA Assembler Prefix, Suffix, Operators

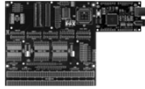
- XMEGA assembler
 - > By default, numbers are assumed to be **decimal**

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

University of Florida, EEL 3744 – File 04
© Drs. Schwartz & Arroyo

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)

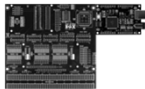
18



EEL3744

XMEGA Assembler Functions

- LOW(expression) returns the low byte of an expression
- HIGH(expression) returns the second byte of an expression
- BYTE2(expression) is the same function as HIGH
- BYTE3(expression) returns the third byte of an expression
- BYTE4(expression) returns the 4th 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^{expression}
- LOG2(expression) returns the integer part of log2(expression)



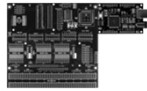
EEL3744



Table_Load_Example.asm



Using the ATMEL Studio Simulator

- 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)



EEL3744

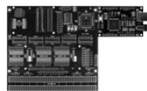
Using uPAD Board and ATMEL Studio Emulator

- Find and select the Simulator menu on top right
 - > Select debugger/programmer | AVR ICE
- 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?  Table_Load_Example.asm
 - > Show memory and registers
 - > Now run **GPIO_Output** (but do **NOT** look at too closely)
 - Will see this file again in Lecture 6: GPIO 

University of Florida, EEL 3744 – File 04
© Drs. Schwartz & Arroyo

GPIO_Output.asm

21



EEL3744

The End!

University of Florida, EEL 3744 – File 04
© Drs. Schwartz & Arroyo

22