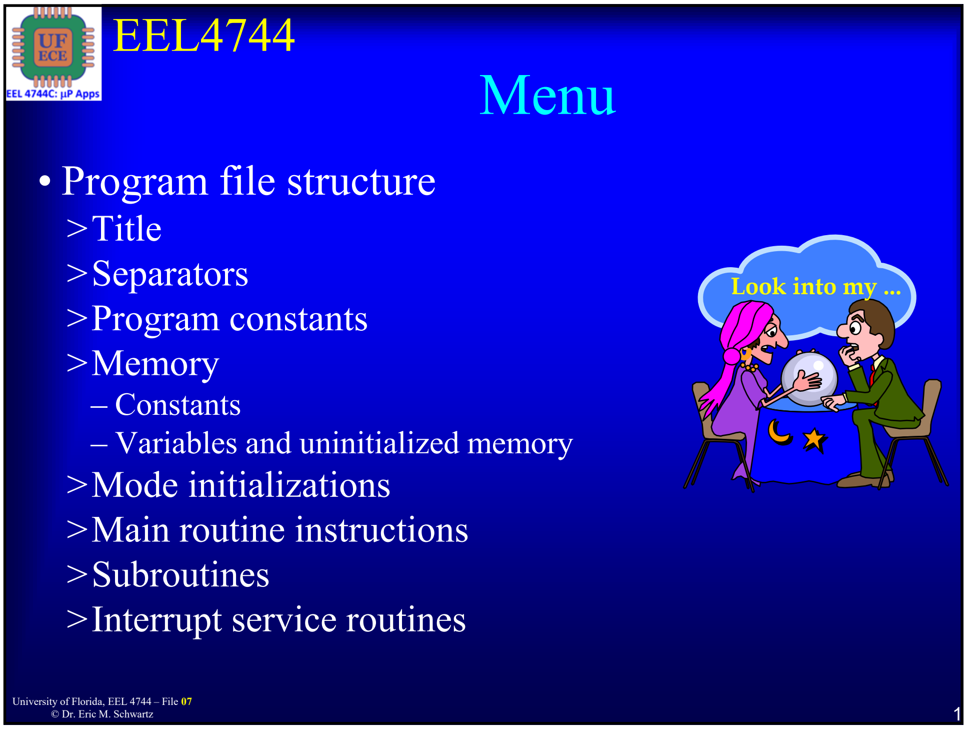


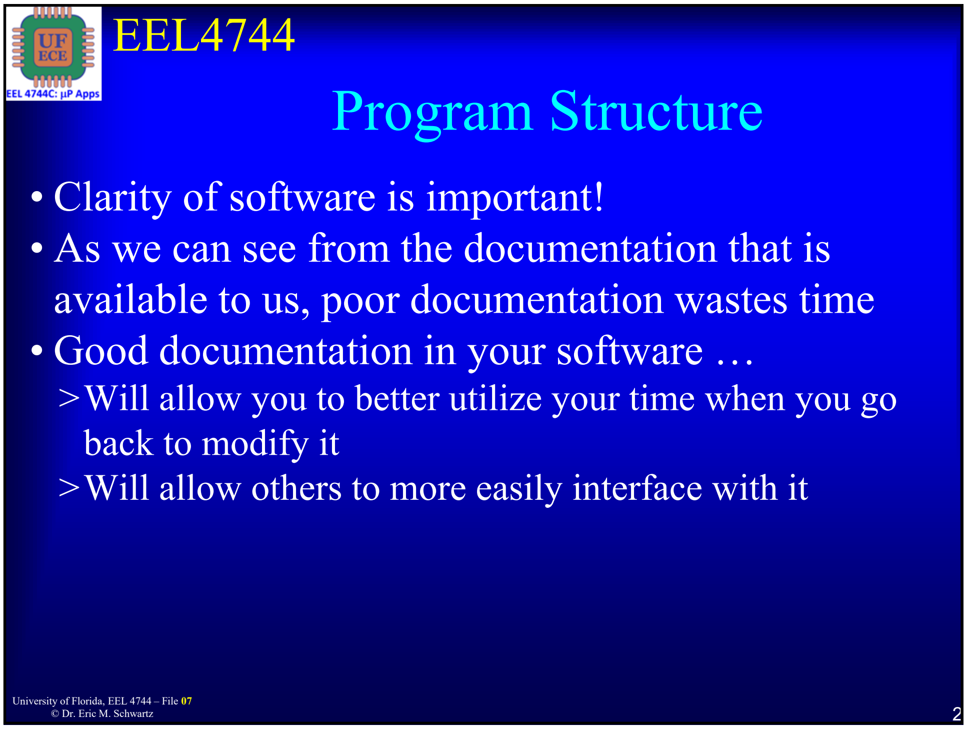
EEL4744

Menu

- Program file structure
  - >Title
  - >Separators
  - >Program constants
  - >Memory
    - – Constants
    - – Variables and uninitialized memory
  - >Mode initializations
  - >Main routine instructions
  - >Subroutines
  - >Interrupt service routines

Look into my ...

1

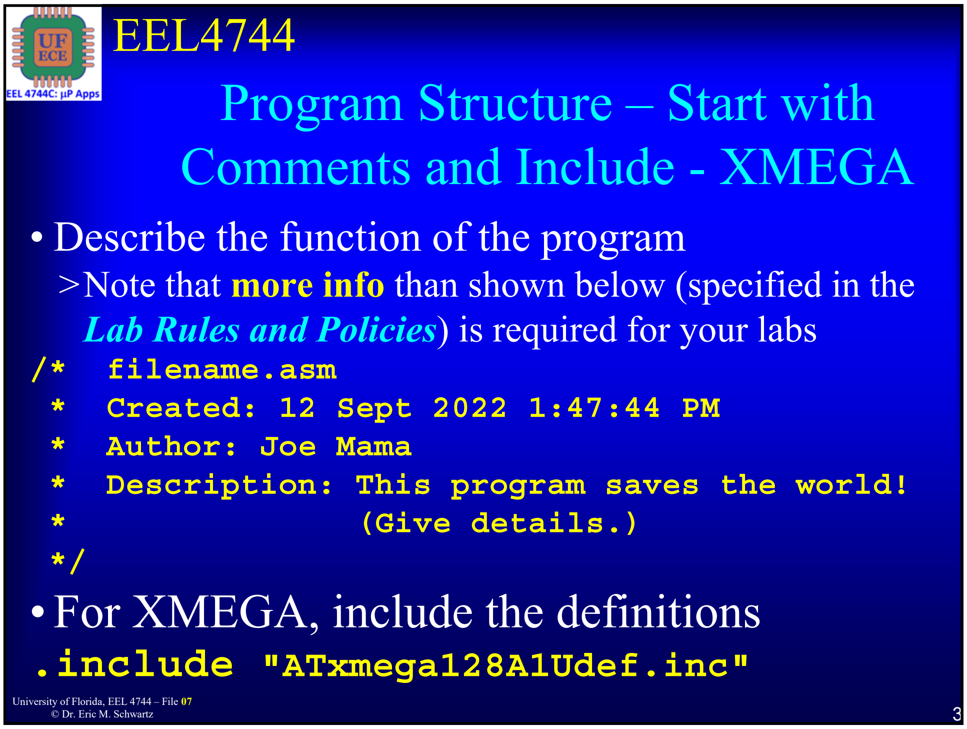


EEL4744

Program Structure

- Clarity of software is important!
- As we can see from the documentation that is available to us, poor documentation wastes time
- Good documentation in your software …
  - >Will allow you to better utilize your time when you go back to modify it
  - >Will allow others to more easily interface with it

2



EEL4744

Program Structure – Start with Comments and Include - XMEGA

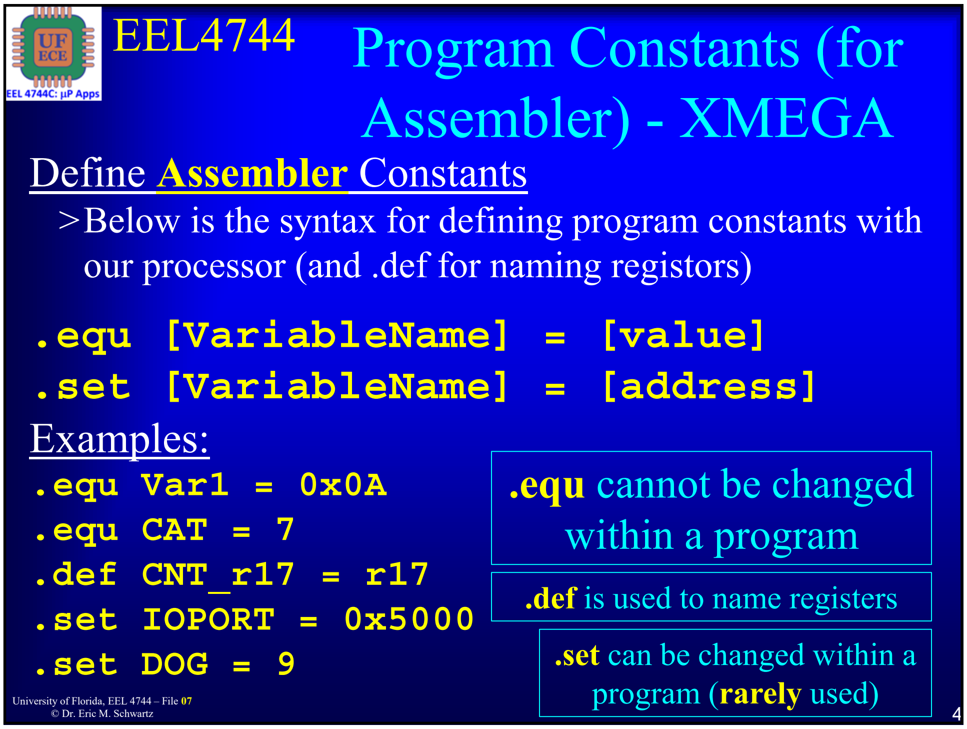
- Describe the function of the program
  - >Note that **more info** than shown below (specified in the ***Lab Rules and Policies***) is required for your labs

```
/*  filename.asm
 *  Created: 12 Sept 2022 1:47:44 PM
 *  Author: Joe Mama
 *  Description: This program saves the world!
 *               (Give details.)
 */
```

- For XMEGA, include the definitions

```
.include "ATxmega128A1Udef.inc"
```

3



EEL4744

Program Constants (for Assembler) - XMEGA

Define **Assembler** Constants

>Below is the syntax for defining program constants with our processor (and .def for naming registors)

```
.equ [VariableName] = [value]
.set [VariableName] = [address]
```

Examples:

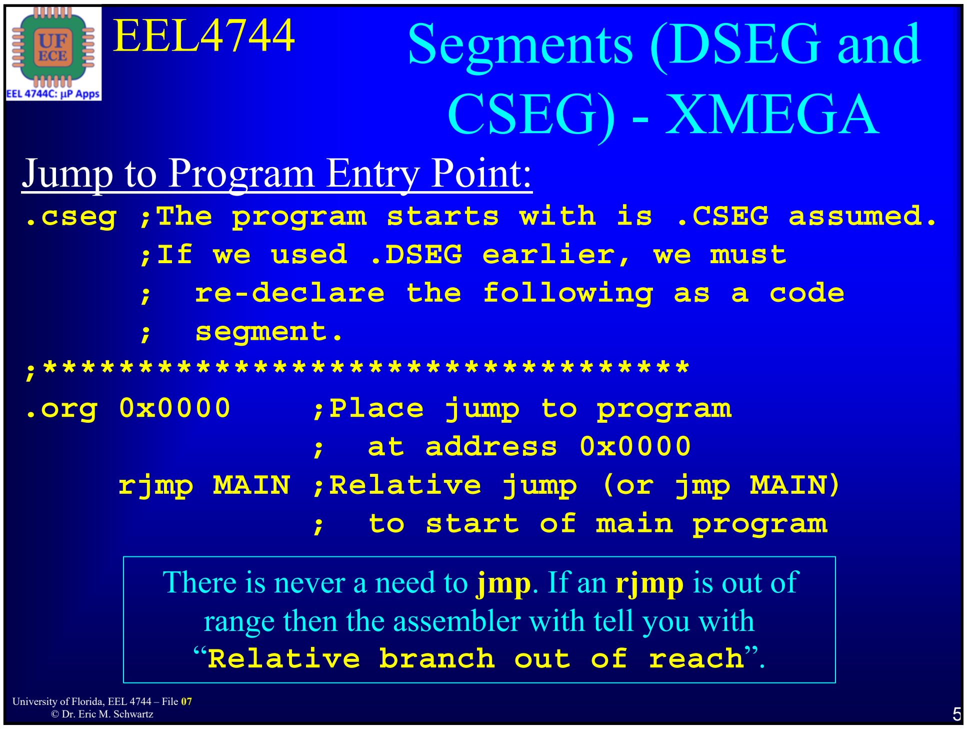
```
.equ Var1 = 0x0A
.equ CAT = 7
.def CNT_r17 = r17
.set IOPORT = 0x5000
.set DOG = 9
```

.equ cannot be changed within a program

.def is used to name registers

.set can be changed within a program (**rarely** used)

4



EEL4744

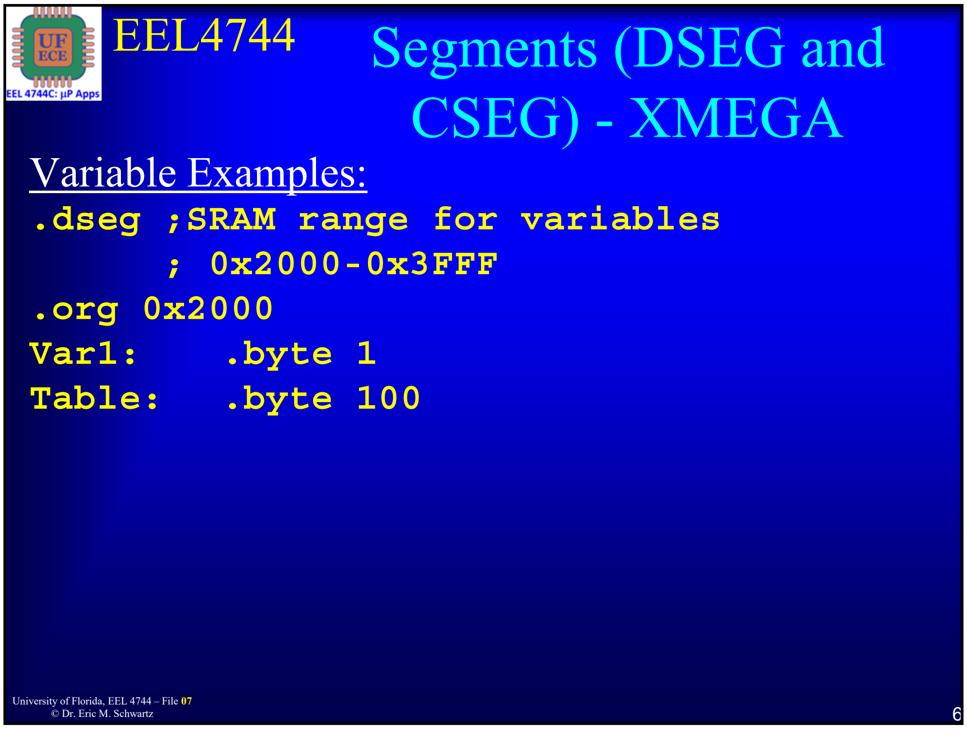
Segments (DSEG and CSEG) - XMEGA

Jump to Program Entry Point:

```
.cseg ;The program starts with is .CSEG assumed.
      ;If we used .DSEG earlier, we must
      ;  re-declare the following as a code
      ;  segment.
;**********************************
.org 0x0000    ;Place jump to program
               ;  at address 0x0000
     rjmp MAIN ;Relative jump (or jmp MAIN)
               ;  to start of main program
```

There is never a need to **jmp**. If an **rjmp** is out of range then the assembler with tell you with "`Relative branch out of reach`".

5



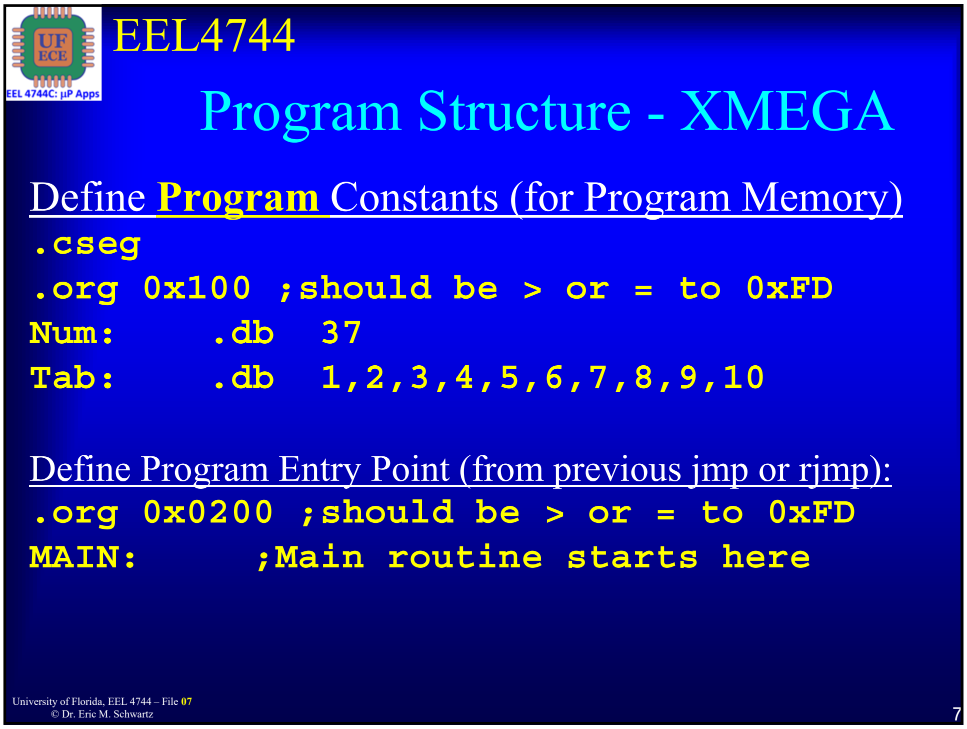
EEL4744

Segments (DSEG and CSEG) - XMEGA

Variable Examples:

```
.dseg ;SRAM range for variables
      ; 0x2000-0x3FFF
.org 0x2000
Var1:    .byte 1
Table:   .byte 100
```

6



EEL4744

Program Structure - XMEGA

Define **Program** Constants (for Program Memory)

```
.cseg
.org 0x100 ;should be > or = to 0xFD
Num:     .db  37
Tab:     .db  1,2,3,4,5,6,7,8,9,10
```

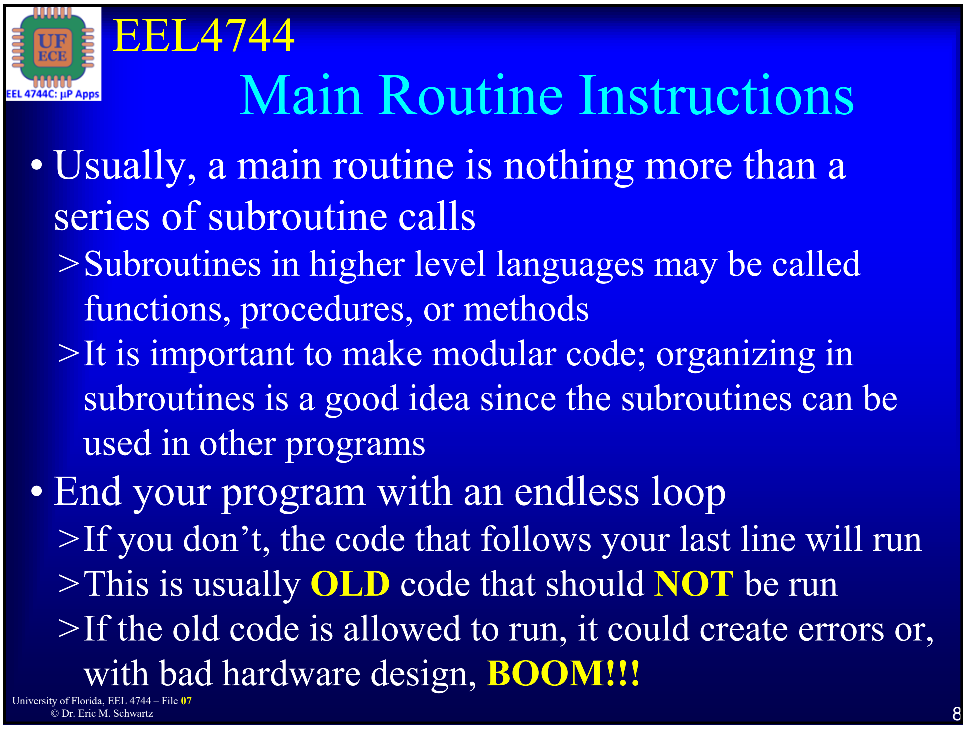
Define Program Entry Point (from previous jmp or rjmp):

```
.org 0x0200 ;should be > or = to 0xFD
MAIN:      ;Main routine starts here
```

University of Florida, EEL 4744 – File 07
© Dr. Eric M. Schwartz

7

7



EEL4744

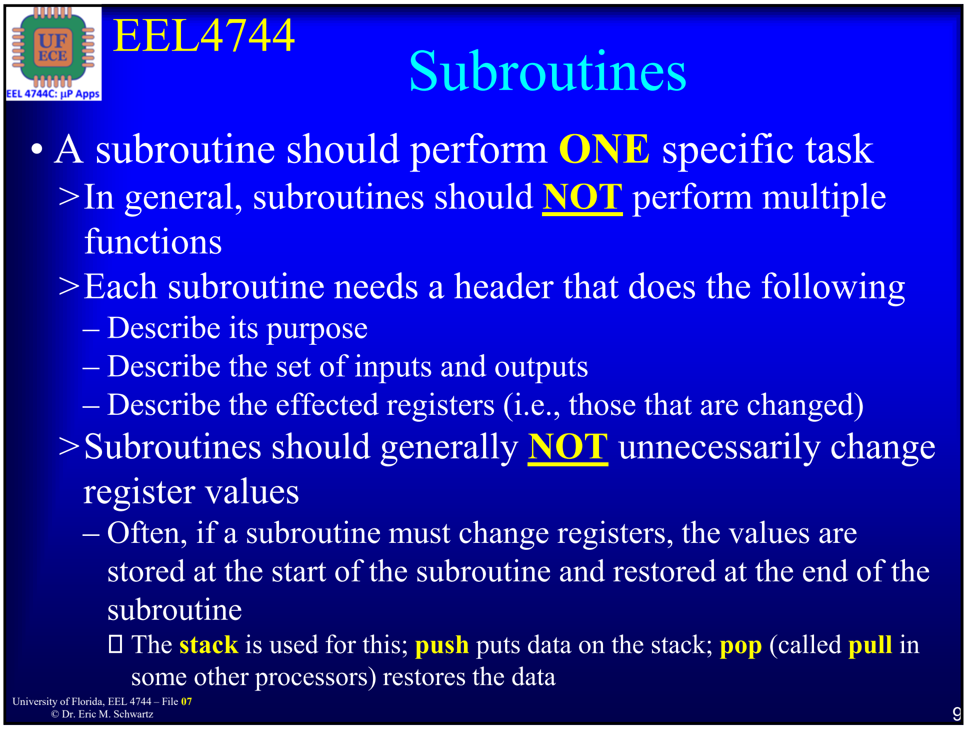
Main Routine Instructions

- Usually, a main routine is nothing more than a series of subroutine calls
  - >Subroutines in higher level languages may be called functions, procedures, or methods
  - >It is important to make modular code; organizing in subroutines is a good idea since the subroutines can be used in other programs
- End your program with an endless loop
  - >If you don't, the code that follows your last line will run
  - >This is usually **OLD** code that should **NOT** be run
  - >If the old code is allowed to run, it could create errors or, with bad hardware design, **BOOM!!!**

University of Florida, EEL 4744 – File 07
© Dr. Eric M. Schwartz

8

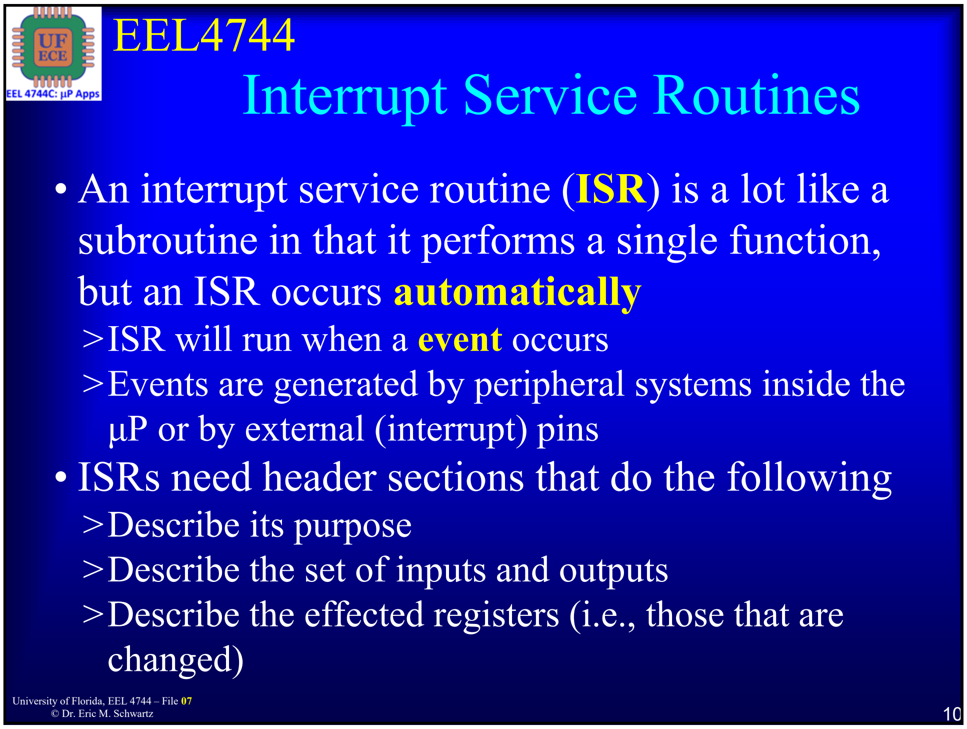
8



EEL4744

Subroutines

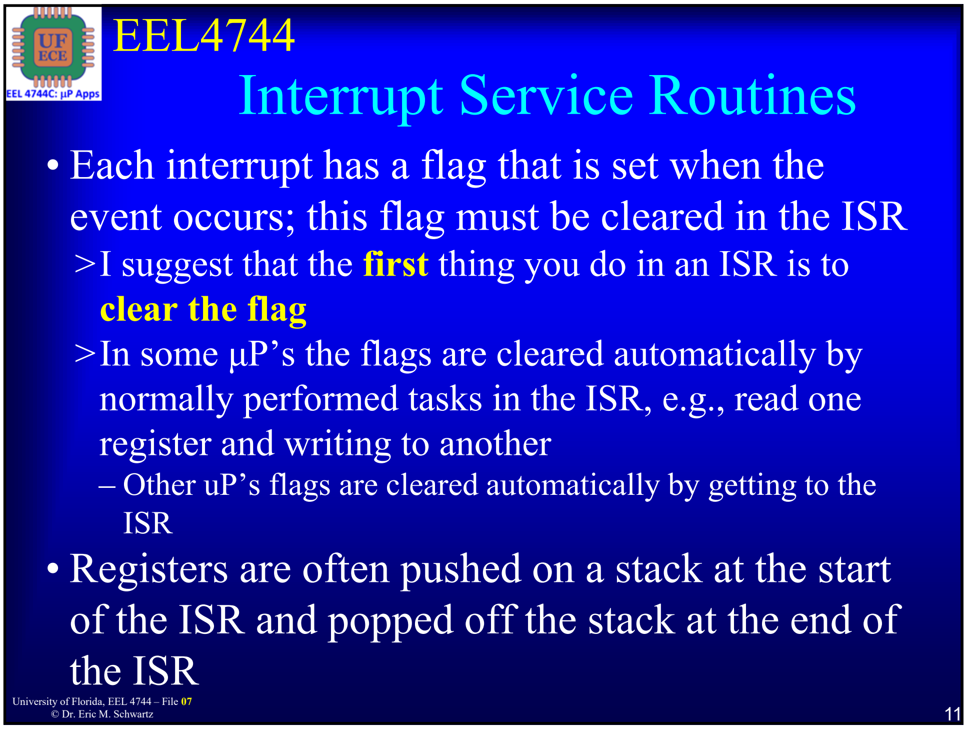
- A subroutine should perform **ONE** specific task
  - >In general, subroutines should **NOT** perform multiple functions
  - >Each subroutine needs a header that does the following
    - – Describe its purpose
    - – Describe the set of inputs and outputs
    - – Describe the effected registers (i.e., those that are changed)
  - >Subroutines should generally **NOT** unnecessarily change register values
    - – Often, if a subroutine must change registers, the values are stored at the start of the subroutine and restored at the end of the subroutine
      - The **stack** is used for this; **push** puts data on the stack; **pop** (called **pull** in some other processors) restores the data



EEL4744

Interrupt Service Routines

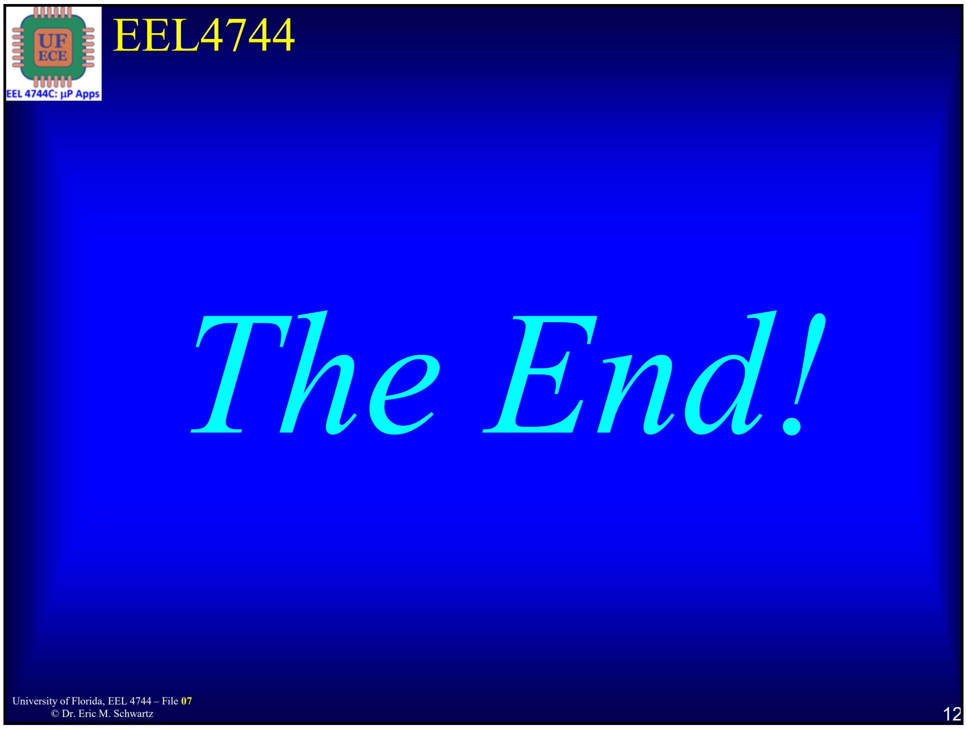
- An interrupt service routine (**ISR**) is a lot like a subroutine in that it performs a single function, but an ISR occurs **automatically**
  - >ISR will run when a **event** occurs
  - >Events are generated by peripheral systems inside the µP or by external (interrupt) pins
- ISRs need header sections that do the following
  - >Describe its purpose
  - >Describe the set of inputs and outputs
  - >Describe the effected registers (i.e., those that are changed)



EEL4744

Interrupt Service Routines

- Each interrupt has a flag that is set when the event occurs; this flag must be cleared in the ISR
  - >I suggest that the **first** thing you do in an ISR is to **clear the flag**
  - >In some µP's the flags are cleared automatically by normally performed tasks in the ISR, e.g., read one register and writing to another
    - – Other uP's flags are cleared automatically by getting to the ISR
- Registers are often pushed on a stack at the start of the ISR and popped off the stack at the end of the ISR



EEL4744

The End!