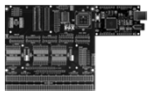
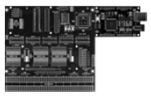


EEL 3744

Menu

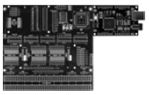
- Program file structure
  - >Title
  - >Separators
  - >Program constants
  - >Memory
    - – Constants
    - – Variables and uninitialized memory
  - >Mode initializations
  - >Main routine instructions
  - >Subroutines
  - >Interrupt service routines



EEL 3744

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



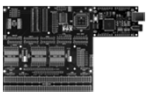
EEL 3744 Program Structure – Start with Comments and Include - XMEGA

- 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: 1/27/2014 5:37:42 PM
 *  Author: Joe Mama
 *  Description: This program saves the world!
 *               (Give details.)
 */
```

- For XMEGA, include the definitions

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

EEL 3744 Program Constants (for Assembler) - XMEGA

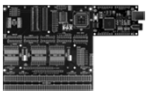
Define **Assembler** Constants

- >Below is the syntax for defining program constants with our processor

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

Examples:

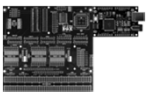
```
.set IOPORT = 0x5000
.set DOG = 9
.equ Var1 = 0x0A
.equ CAT = 7
.def Counter = R17
```



EEL 3744 Segments (DSEG and CSEG) - XMEGA

Variable Examples:

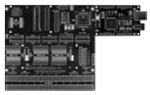
```
.dseg ;SRAM range for variables
      ; 0x2000-0x3FFF
.org 0x2000
Var1:    .byte 1
Table:   .byte 100
```

EEL 3744 Program Structure - XMEGA

Jump to Program Entry Point:

```
.cseg ;Everything is .CSEG by default,
      ;since we used .DSEG earlier, we must
      ;redeclare 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
```



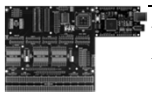
EEL 3744
Program Structure - XMEGA

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

```
.org 0x100
Num:    .db  37
Tab:    .db  1,2,3,4,5,6,7,8,9,10
```

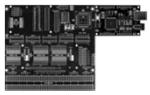
Define Program Entry Point (from previous jmp or rjmp):

```
.org 0x0200
MAIN:     ;Main routine starts here
```

EEL 3744 Mode Initializations:
Watchdog Timer - XMEGA

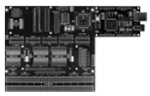
- The Watchdog Timer (WDT) is clocked from the 1kHz output from the 32kHz ultra low power (ULP) internal oscillator.
- Bit 1 of the CTRL register is the WDT ENABLE
  - >Writing a 0 to this bit will disable the watchdog timer
- Bit 0 of the CTRL register is the CEN (Change Enable)
  - >You must write a 1 to this bit at the same time in order to make changes to WDT ENABLE



EEL 3744

Main Routine Instructions

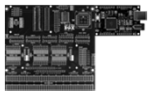
- Usually, a main routine is nothing more than a series of subroutine calls
  - >Subroutines in higher level languages may be called functions, procedures, or methods
  - >It is important to make modular code; organizing in subroutines is a good idea since the subroutines can be used in other programs
- 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!!!**



EEL 3744

Subroutines

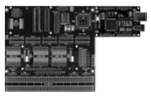
- A subroutine should perform **ONE** specific task
  - >In general, subroutines should **NOT** perform multiple functions
  - >Each subroutine needs a header that does the following
    - – Describe its purpose
    - – Describe the set of inputs and outputs
    - – Describe the effected registers (i.e., 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



EEL 3744

Interrupt Service Routines

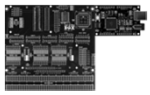
- An interrupt service routine (**ISR**) is a lot like a subroutine in that it performs a single function, but an ISR occurs **automatically**
  - >ISR will run when a **event** occurs
  - >Events are generated by peripheral systems inside the µP or by external (interrupt) pins
- ISRs need header sections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



EEL 3744

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 in 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


EEL 3744

The End!