EEL 3744

Menu

• Debugging
  > Why do we need to debug?
  > When do we need to debug?
  > How do we debug?
    – Before/as we code
    – After we code
  > Common bugs

The Cost of Debugging

• Increases exponentially in time

Example:
```c
int i,j;
for (;i++;i<50)
```

Cost:
80 hrs * $18/hr = $1.4k
with overhead, $2.8k
When should we start debugging?

• Before we code (easiest)
• As we code
• After we code (hardest)

How to debug before and as we code

• Make code modular
  > Should be small, independent, self-contained
  > Inter-modular bugs are the worst kind
• Well defined functions and parameters
• Start simple; work toward more complex
• Evaluate all degenerate cases
  > Code defensively
• Run, debug each small module as we write it
Which of the principles does this code violate?

```assembly
.ORG 0x0000
rjmp MAIN
.ORG 0x200
MAIN:
; Load stack pointer
ldi R16, 0x37

ISR:
ADD R16, R17
RETI
```

- **Modularity**
  > ISR depends on the main routine to initialize R16
  
  **Warning: Dangerous bug!**

- **Well defined parameters**
  > R16 is probably not documented as a parameter to ISR

- **Degenerate cases**
  > What if Main or some other previously executed subroutine ever changes R16?

How to debug after we code (1)

- Be creative
- **Cardinal rule of debugging: Isolate the Error**
  > Where does the problem occur?
- What do the hardware and software do?
- Stare at code
  > Look at list file
- Memory dumps and stack traces
How to debug after we code (2)

- Use breakpoints and single stepping
  > Make sure memory and registers are as expected
- Write debug code
  > Print
  > Tags
  > Outputs, LEDs
- Keep it simple

Example of using tags

```
.ORG 0x0000
rjmp MAIN
.ORG 0x200
MAIN:
; load stack pr
    CLR R25
    ldi R25, 1
    ldi R25, 2
    ldi R25, 3
```

- The goal is to locate an error
- Set up (and clear) a register e.g., R25, (or a variable in memory)
- Change the value in the register (or memory) at certain strategic locations
- You could use LEDs or print statements, if available
Remember common error types

- Off by 1
- Zero case, -1 case, +1 case, MaxInt, MinInt
- No termination condition for a loop
- Variables not initialized
- Unexpected side effects
- Inter-modular bugs

Example of Common Errors

```plaintext
.equ Size = 256
.org 0x2000
Table: .byte Size
...
MAIN:
  ldi XL, low(Table)
  ldi XH, high(Table)
  ldi R18, Size
Loop:
  dec R18
  ld R17, X+
  add R16, R17
  brne Done
  beq Loop
Done:
```

- This program sums the contents of Table, in R16
- Can not put 256 into 8-bit register R18 (too big!)
- The first time through the loop, we add the contents of @Table+0 to R16
  > R16 is not initialized to zero!
- DEC should be last thing before branches (since ADD could change flags)
- First branch should be BEQ or BRSH, not BRNE
- Degenerate case: R16 is probably too small to hold the sum of so many numbers
Additional Comments

• Do something; divide and conquer
• Errors in ISRs are especially dangerous so take extra precautions
• Other assemblers may be available (for $$$) that give better warning messages when things are “funny”
• When writing code (and designing hardware) keep in mind how you can test it
  > Design in extra things that can help you test modules (hardware OR software)

The End!