

Last Name: \_\_\_\_\_ First Name: \_\_\_\_\_

Jonathan Valvano

February 13, 2006, 1:00pm-1:05pm

```

static char aa=0; // static restricts scope to file
char addIt(const char bb){ // const makes bb read-only
char cc;
    aa++;
// at this point during execution
    cc = aa+bb;
    return(cc);
}
volatile char dd; // volatile disables optimizer
void main(void){static char ee; // static makes is permanent
    ee = PTT;
    dd = addIt(ee+5);
}

```

**Question 1.** Immediately before the `cc=aa+bb;` line is executed, which of the variables `aa bb cc dd ee` will be temporarily allocated (stack or register)? I am not asking which variables are legally within the scope of the function, but where they are allocated (note: proper variable naming style has been violated).

bb, cc
--------

**Question 2.** Which three events will trigger a Real Time Interrupt (RTI)? I.e., which events if true will cause an RTI interrupt to happen? *Give three letters where the order of events does not matter.*

A, F, I
---------

- A) Software sets **RTIE** to 1, **arm**
- F) Software sets the **I** bit to 0, **enable**
- I) Timer hardware sets **RTIF** to 1, **flag**

Why these are not true:

- B) Software sets **RTIE** to 0, **disarm**
- C) Software sets **RTIF** to 1, software can not set flag
- D) Software sets **RTIF** to 0, acknowledge, software does this in ISR
- E) Software sets the **I** bit to 1, **disable**
- G) Timer hardware sets **RTIE** to 1, hardware can not change this bit
- H) Timer hardware sets **RTIE** to 0, hardware can not change this bit
- J) Timer hardware sets **RTIF** to 0, hardware can only set the flag

**Question 3.** Which three events occur automatically in hardware as an interrupt is processed, making the thread switch from foreground to background? *This order matters.*

1) D
2) A or F
3) F or A

- D) The registers are pushed on the stack, notice it pushes CCR with I=0
  - A) The **I** bit is set to 1, automatically disables before running ISR
  - F) The interrupt vector is loaded into the PC register, vector contains the address of ISR
- Why these are not true:
- G) The **RTIE** bit is cleared to 0, the arm bit is under software control
  - B) The **I** bit is cleared to 0, this will be cleared at the end of the ISR by the `rti` instruction
  - C) The **RTIF** bit is cleared to 0, acknowledge occurs as a software action in the ISR
  - E) The interrupt vector is loaded into the SP register, SP is the stack pointer