"The customer is always right"

1. **Compaq is considering changing the command "Press Any Key" to "Press Return Key" because of the flood of calls asking where the "Any" key is.**
2. **AST technical support had a caller complaining that her mouse was hard to control with the dust cover on. The cover turned out to be the plastic bag the mouse was packaged in.**
3. **Another Dell customer called to say he couldn't get his computer to fax anything. After 40 minutes of troubleshooting, the technician discovered the man was trying to fax a piece of paper by holding it in front of the monitor screen and hitting the "Send" key.**
4. **Yet another, Dell customer called to complain that his keyboard no longer worked. He had cleaned it by filling up his tub with soap and water and soaking the keyboard for a day, then removing all the keys and washing them individually.**
5. **A Dell technician received a call from a customer who was enraged because his computer had told him he was "Bad and an invalid." The tech explained that the computer's "bad command" and "invalid" responses shouldn't be taken personally.**

Lecture 8 objectives

- Debugging using assembly language listings
- interface binary switch using pull-up resistor to an input port
- interface LED using a 7405 or PN2222
- draw flowchart of **`RTI`** project
- profiling with the scope showing just how small a percentage of time is spent in the background

Debugging from an assembly perspective

```
short X1;
static short X2;
const short X3=3000;
short add3(short z1, short z2, short z3){
  short y;
  y = z1+z2+z3;
  return(y);
}
void main(void){ short y;
  X1 = 1000;
  X2 = 2000;
  y = add3(X1,X2,X3);
  EnableInterrupts;
  for(;;) {} /* wait forever */
}
```

assembly listing edited from **`main.lst`**

```
Function: add3
  0000 3b           PSHD
  0001 ec86         LDD   6,SP
```

```
  0003 e384          ADDD  4,SP
  0005 e380          ADDD  0,SP
  0007 30            PULX
  0008 3d            RTS
Function: main
  0000 cc03e8        LDD   #1000
  0003 7c0000        STD   X1
  0006 59            ASLD
  0007 7c0000        STD   X2
  000a 49            LSRD
  000b 6cac          STD   4,-SP
  000d 59            ASLD
  000e 3b            PSHD
  000f cc0bb8        LDD   #3000
  0012 0700          BSR   add3
  0014 6ca3          STD   4,+SP
```

Action
1) New 9S12C32 project, select simulator
2) compile options, make S19, listing
3) copy paste example
4) Debug, single step

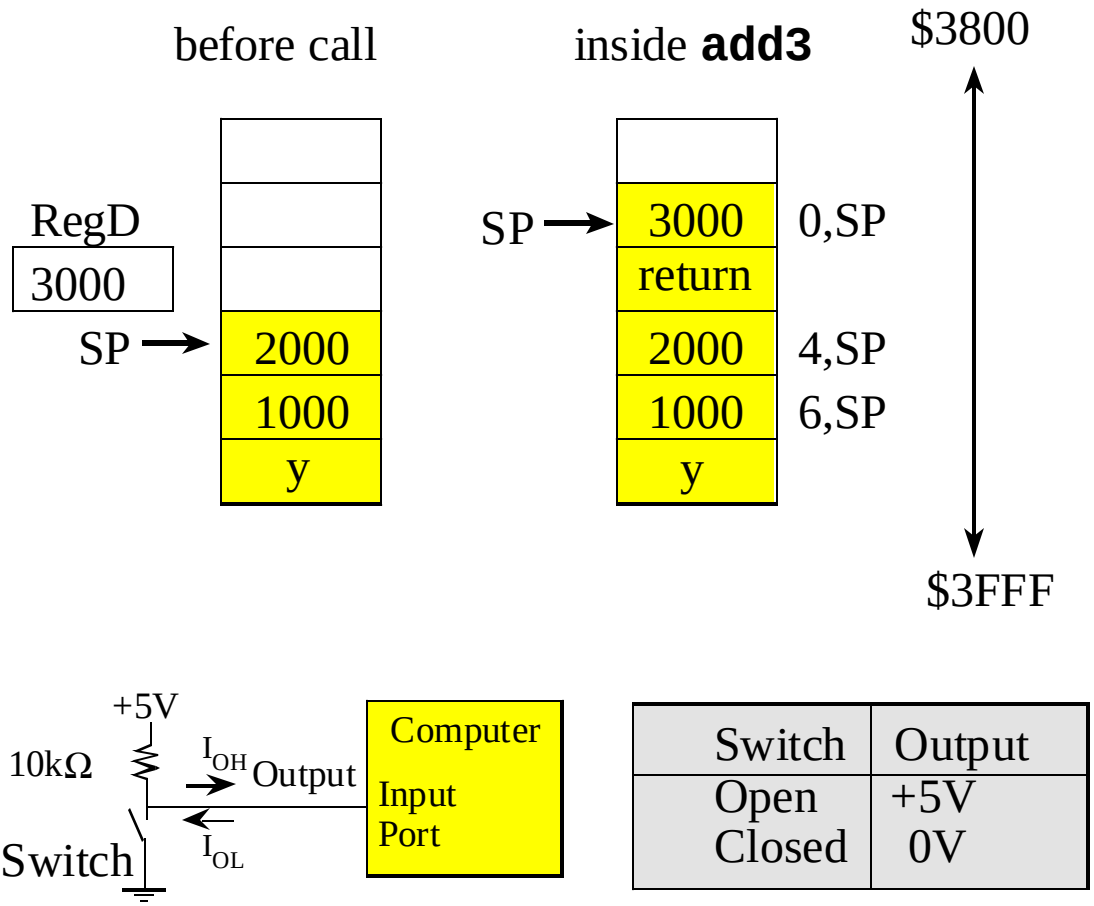
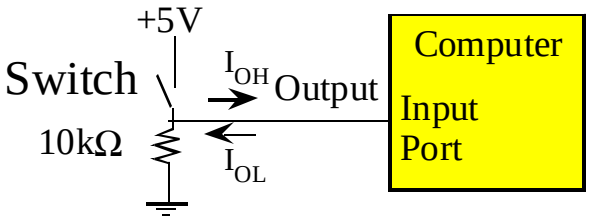
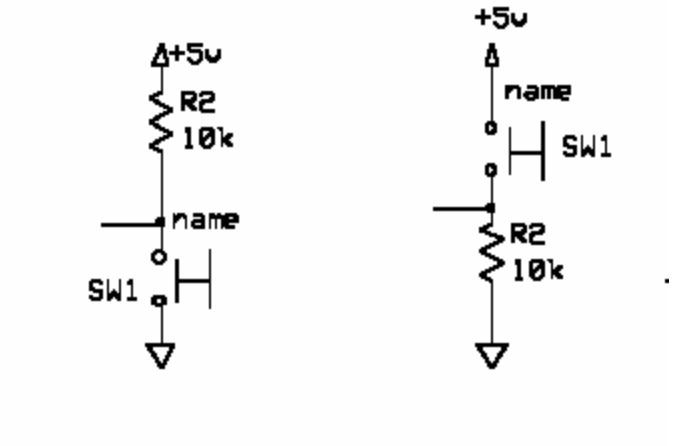


Figure 8.1. A simple switch interface.



Switch	Output
Open	0V
Closed	+5V

Figure 8.2. Another simple switch interface.



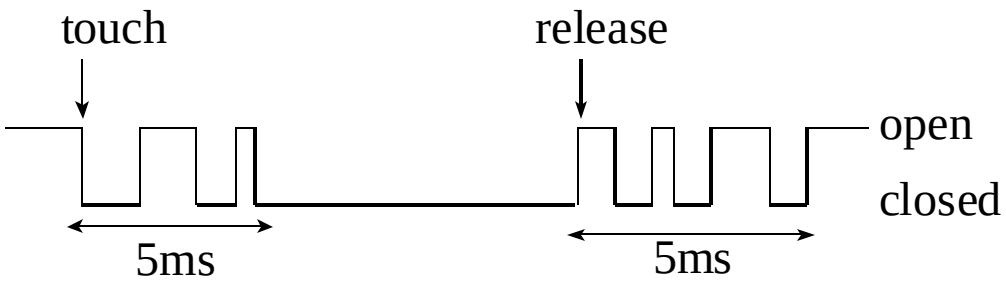


Figure 8.4. Switch timing showing bounce on touch and release.

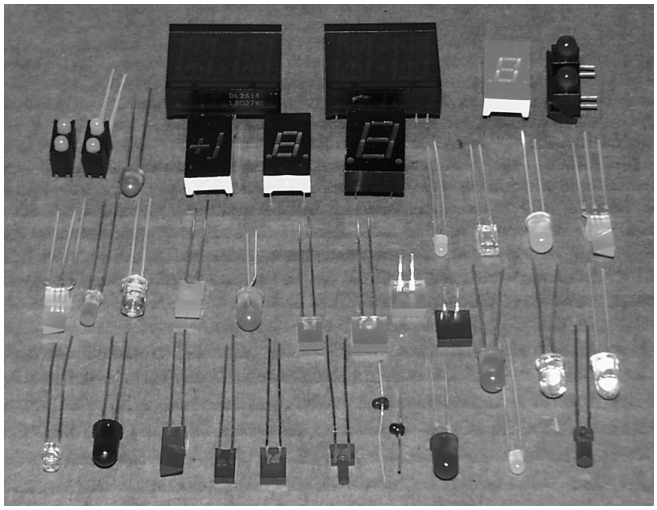


Figure 8.28. LED's come in a wide variety of shapes, sizes, colors.

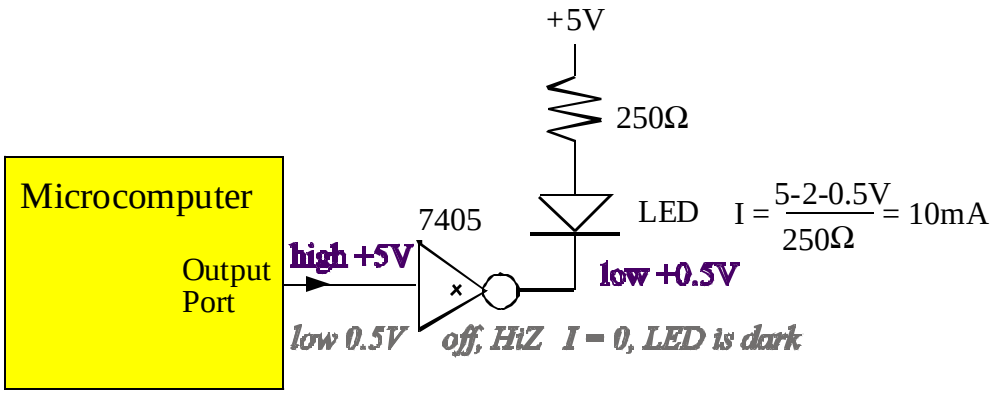
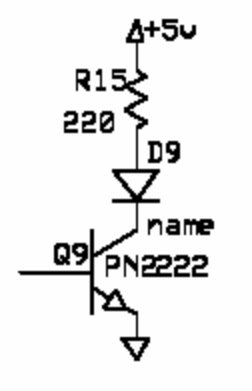


Figure 8.30. A single LED interface.



PN2222 V_{ce} = 0.3V
Red LED 1.5 min < V < 2.0 max
Max LED Current I = (5-1.5-0.3)/220=15mA
Min LED Current I = (5-2.0-0.3)/220=12mA
h_{fe}=50, so I_b = 0.3mA (which is less than I_{OH} of the 6811)

Software must respond to events within a prescribed time.
time between new keyboard inputs might be 10ms.
software latency
time from when new input is ready until the time software reads new data.
software latency must be less than 10ms.
interrupts guarantee an upper bound on the software response time.

Respond to infrequent but important events.
alarm conditions like low battery power and
error conditions can be handled with interrupts.

Periodic interrupts, generated by the timer at a regular rate
clocks and timers
computer-based data acquisition and
digital control systems.
Increase the overall bandwidth
buffer the data, spend less time waiting.
uses a ***first in first out queue***

4.1. What are interrupts?

An **interrupt** is the automatic transfer of software execution
in response to hardware that is asynchronous with the current software execution.
external I/O device (like a keyboard or printer) or
an internal event (like an op code fault, or a periodic timer.)
occurs the hardware needs service (busy to done state transition)

A **thread** is defined as the path of action of software as it executes.
a **background thread** interrupt service routine is called.
a new background thread is created for each interrupt request.
local variables and registers used in the interrupt service routine are unique
threads share globals

Each potential interrupt source has a separate **arm** bit. E.g., **`RTIE`**
set the arm bits for those devices from which it wishes to accept interrupts,
deactivate the arm bits within those devices from which interrupts are not to be allowed.

Each potential interrupt source has a separate **flag** bit. E.g., **`RTIF`**
hardware sets the flag when it wishes to request an interrupt
software clears the flag in ISR to signify it is processing the request

Interrupt **enable** bit, I, which is in the condition code register.
enable all armed interrupts by setting I=0, or **`cli`**
disable all interrupts by setting I=1. **`sei`**
I=1 does not dismiss the interrupt requests, rather it postpones

Shared open collector hardware request (polled)

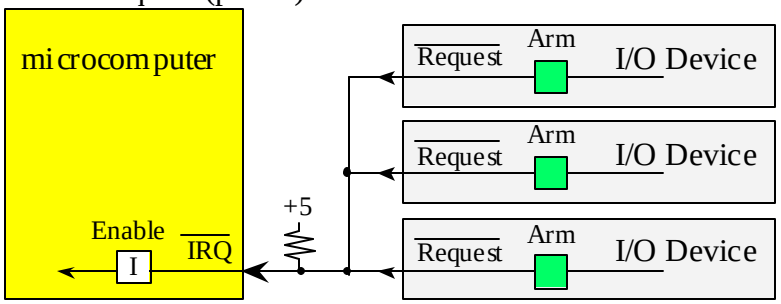


Figure 4.1. Wire-or negative-logic interrupt request line.

share the same interrupt vector
interrupt service routine must first determine which device requested the interrupt
TDRE and RDRF have a shared vector, ISR must poll

Separate edge-triggered hardware request (vectored)

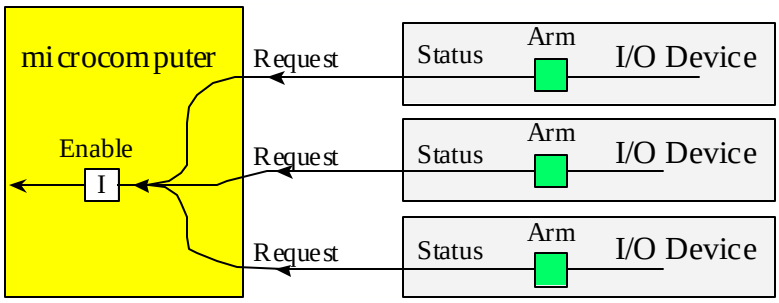


Figure 4.2. Dedicated edge-triggered interrupt request lines.

separate interrupt vectors
jumps automatically to specific interrupt service routine

1) the software is simpler, therefore it will be easier to debug and it will run faster,
2) there will be less coupling in between modules, making it easier to debug and to reuse code,
3) it will be easier to implement priority such that higher priority requests are handled quickly

Three conditions must be true simultaneously for an interrupt to occur:

1) Initialization software will set the arm bit
   individual control bit for each possible flag that can interrupt
2) When it is convenient, the software will enable, I=0
   allow all interrupts now
3) Hardware action (busy to done) sets a flag E.g., **`RTIF`**
   new input data ready,
   output device idle,
   periodic,
   alarm

What happens when an interrupt is processed?

1) the execution of the main program is suspended (the current instruction is finished),
   pushes registers on the stack
   sets the I bit
   gets the vector address from high memory
2) the interrupt service routine, or background thread is executed,
   clears the flag that requested the interrupt
   performs necessary operations
   communicates using global variables
3) the main program is resumed when the interrupt service routine executes **`rti`**.
   pulls the registers from the stack

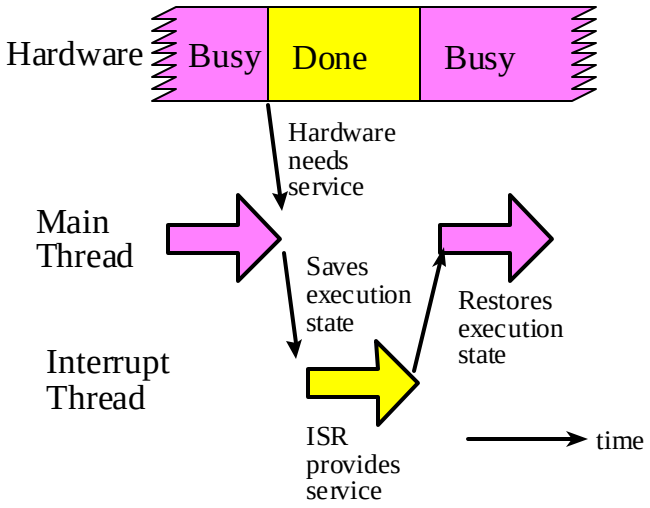


Figure 4.3. An interrupt causes the main thread to be suspended, and the interrupt thread is run.

What type of situations lend themselves to interrupt solutions?

Gadfly	**Interrupts**	**DMA**
Predicable	Variable arrival times	low latency
Simple I/O	Complex I/O, different speeds	high bandwidth
Fixed load	Variable load	
Dedicated, single thread	Other functions to do	
Single process	Multithread or multiprocess	
Nothing else to do	Infrequent but important alarms	
	Program errors	
	Overflow, invalid op code	
	Illegal stack or memory access	
	Machine errors	
	Power failure, memory fault	
	Breakpoints for debugging	
	Real time clocks	
	Data acquisition and control	

Table 4.1. Each synchronization method has motivations for its use.

9S12C32 interrupts we will be using

0xFFD6	interrupt 20 SCI
0xFFDE	interrupt 16 timer overflow
0xFFE0	interrupt 15 timer channel 7
0xFFE2	interrupt 14 timer channel 6
0xFFE4	interrupt 13 timer channel 5
0xFFE6	interrupt 12 timer channel 4
0xFFE8	interrupt 11 timer channel 3
0xFFEA	interrupt 10 timer channel 2
0xFFEC	interrupt 9 timer channel 1
0xFFEE	interrupt 8 timer channel 0

0xFFF0 interrupt 7 RTI real time interrupt
0xFFF6 interrupt 4 SWI software interrupt
0xFFF8 interrupt 3 trap software interrupt
0xFFFE interrupt 0 reset

MC9S12C32 Real Time Interrupt RTI project

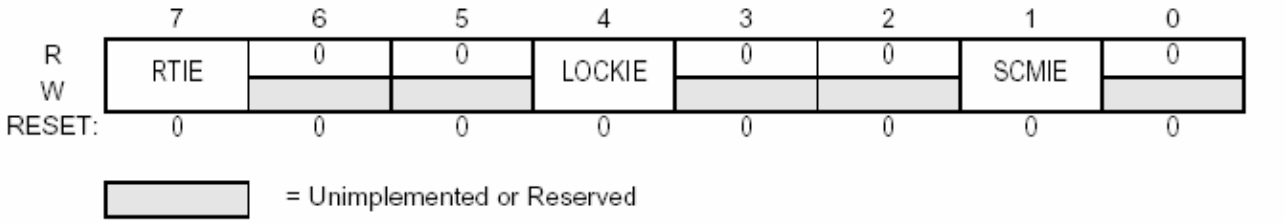


Figure 3-5 CRG Interrupt Enable Register (CRGINT)

Data sheets in file "S12CRGV4.pdf"

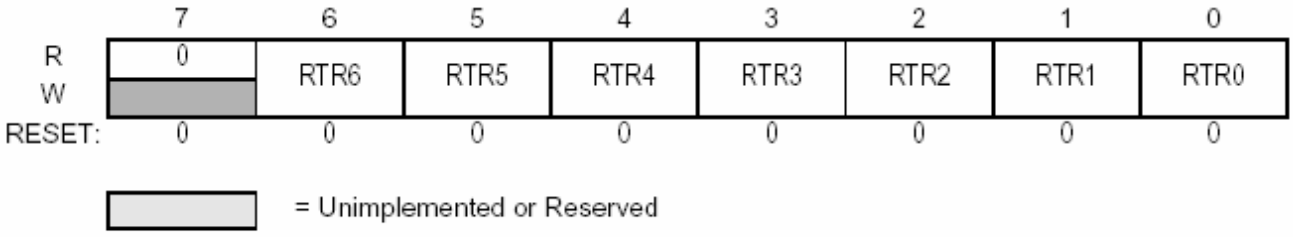


Figure 3-8 CRG RTI Control Register (RTICTL)

Data sheets in file "S12CRGV4.pdf"

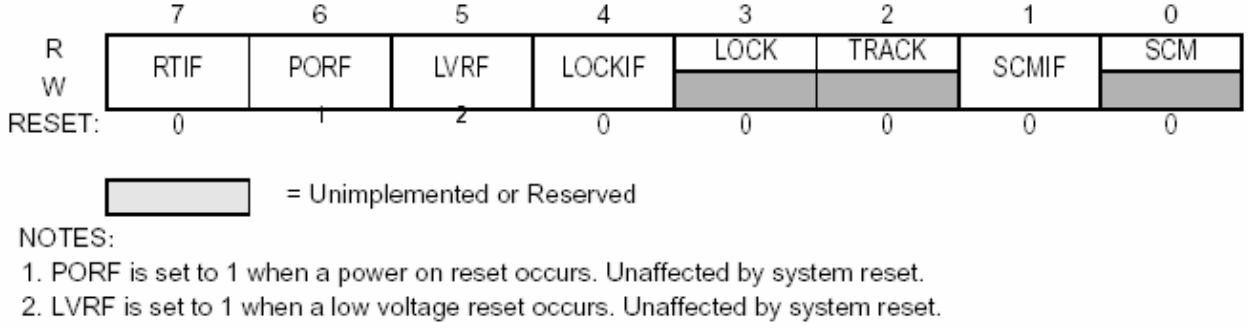


Figure 3-4 CRG Flags Register (CRGFLG)

RTIF — Real Time Interrupt Flag

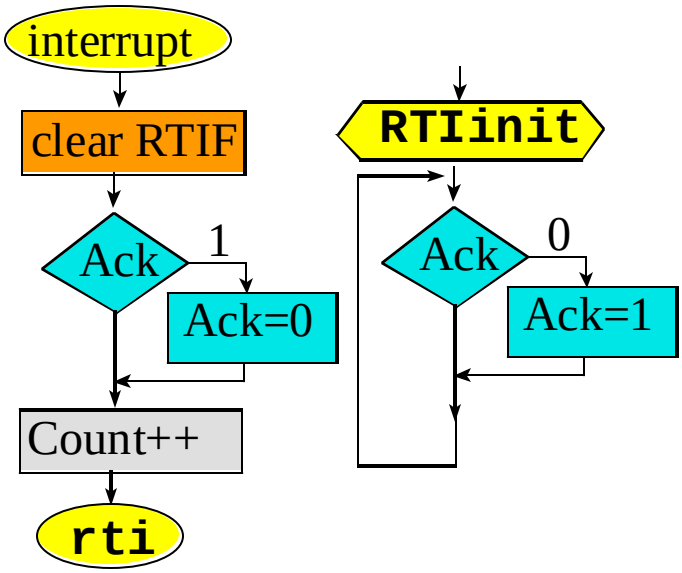
RTIF is set to 1 at the end of the RTI period. This flag can only be cleared by writing a 1. Writing a 0 has no effect. If enabled (RTIE=1), RTIF causes an interrupt request.
1 = RTI time-out has occurred.
0 = RTI time-out has not yet occurred.

Data sheets in file "S12CRGV4.pdf"

The rate is independent of the PLL

```
void RTIinit(void){
  asm sei         // Make ritual atomic
  DDRT |= 0x03;  // PortT bit 1,0 to LEDs
  PTT = 0x00;
  CRGINT = 0x80; // RTIE=1 enable rti
  RTICTL = 0x33; // period=250ns*4096*4
  Count = 0;     // interrupt counter
/*  base clock is OSCCLK=250nsec
RTR[6:4] Real Time Interrupt Prescale Rate
  000 off
  001 divide by 1024
  010 divide by 2048
  011 divide by 4096
  100 divide by 8192
  101 divide by 16384
  110 divide by 32768
  111 divide by 65536
RTR[3:0] Real Time Interrupt Modulus Counter
 0000 divide by 1
 0001 divide by 2
 0010 divide by 3
 0011 divide by 4
 0100 divide by 5
 0101 divide by 6
 0110 divide by 7
 0111 divide by 8
*/
  Ack = 1;       // means foreground is ready
  asm cli
}
void interrupt 7 handler(){
  PTT |= 0x01;       // debugging
  CRGFLG = 0x80;     // ack, clear RTIF
  if(Ack==1){        // software handshake
    Ack = 0;         // means RTI happened
  }
  Count++;           // number of interrupts
  PTT &= ~0x01;      // debugging
}
void main(void) {
  RTIinit();
  for(;;){
    if(Ack==0){    // software handshake
      Ack = 1;     // main has happened
      PTT ^= 0x02; // toggle bit 1
```

```
    }
  }
}
```



run RTI project, see scope signals

"The customer is always right" (continued)

6. A confused caller to IBM was having trouble printing documents. He told the technician that the computer had said it couldn't find the printer. The user had also tried turning the computer screen to face the printer, but that his computer still couldn't "see" the printer.

7. An exasperated caller to Dell Computer Tech Support couldn't get her new Dell Computer to turn on. After ensuring the computer was plugged in, the technician asked her what happened when she pushed the power button. Her response, "I pushed and pushed on this foot pedal and nothing happens." The foot pedal" turned out to be the computer's mouse.

8. Another customer called Compaq tech support to say her brand new computer wouldn't work. She said she unpacked the unit, plugged it in and sat there for 20 minutes waiting for something to happen. When asked what happened when she pressed the power button, she asked, "What power button?"

9. Another IBM customer had trouble installing software and rang for support "I put in the first disk, and that was OK. It said to put in the second disk, and had some problems with the disk. When it said to put in the third disk, I couldn't even fit it in..." The user hadn't realized that "Insert Disk 2" implied to remove Disk 1 first.

10. A woman called the Canon help desk with a problem with her printer. The tech asked her if she was "running it under windows." The woman responded, No, my desk is next to the door. But that is a good point. The girl sitting in the cubicle next to me is under a window and her printer is working fine.