Real-Time Programming

Week 2:
Interrupt Driven Programs

Instructors – Tony Montiel & Ken Arnold
rtp@hte.com

Overview

- Re-entrant code
- Critical code segments
- Interrupts
- Communication Mechanisms

Re-entrant Code

- Code that can be interrupted, and called again one or more times, before completing execution.
  - Must use only local resources (i.e.variables)
  - Cannot modify global resources
  - Compiler libraries are usually *not* re-entrant
  - Most OS functions are *not* re-entrant
  - Code that uses dedicated hardware is not.

Re-entrant 8051 Example

```
int y;
void notReent( int *x )
{
  y = y + *x;
  *x = y;
}
```

Re-entrant 8051 Example

```
int y;
void reentSafe( int *x )reentrant
{
  DisableInterrupts();
  y = y + *x;
  *x = y;
  EnableInterrupts();
}
```


Critical Code Segments

- A Code Sequence that can't be Interrupted without potential for error
  - If an interrupt occurs inside one of these segments of code, an error can occur
  - Simplest fix is to disable interrupts before critical segment and enable afterward
  - Example: two processes use UART and signal use of the UART with a flag.

Critical Code Segments

- If Interrupted ***May*** Result in Errors
  - Actual Occurrence often VERY Low Probability
- Occurs When Resources are Shared
- Mutually Exclusive Access Required
- Requires Use of Semaphore (a.k.a. Mutex)

Shared-Data Problems

- Examples:
  - Generic example
  - Brake and Turn signal
  - Park/Out-of-gear, starter and brake select

Generic Shared Data Problem

```
static volatile int uartAvail = 1;
```

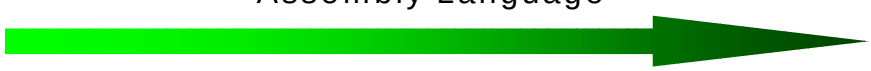
```
void Task1( void )
{
   while ( uartAvail == 0 ) ;
   uartAvail = 0;  // In Use
   writeUART( “Hello” );
   uartAvail = 1;  // Avail
}
```

```
void Task2( void )
{
   while ( uartAvail == 0 ) ;
   uartAvail = 0;  // In Use
   writeUART( “G-bye” );
   uartAvail = 1;  // Avail
}
```

PROBLEM: uartAvail is a FLAG NOT a SEMAPHORE.

Generic Shared Data:

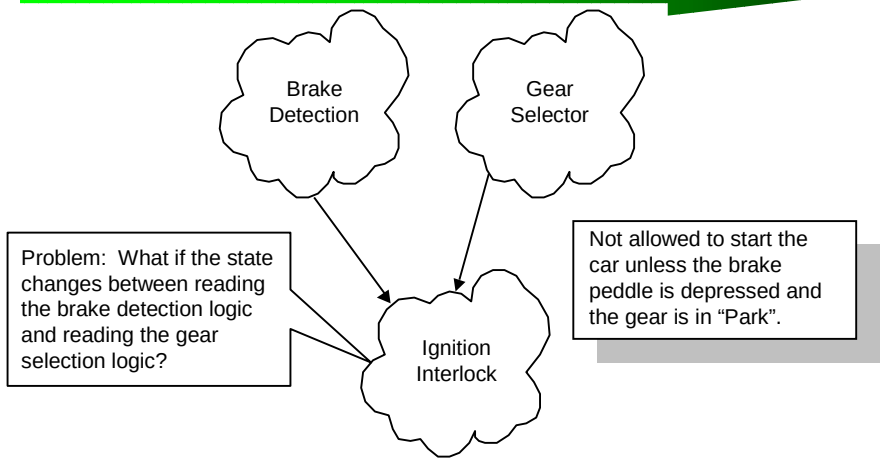
Assembly Language



```
task1:
   MOV      A,uartAvail+01H
   ORL      A,uartAvail
   JZ       task1

   MOV      uartAvail,#00H
   MOV      uartAvail+01H,#00H
   :
   :
```

Simple Automotive Example



“Atomic” Operation

- Atomic:
  - Cannot be interrupted.
  - An “indivisible” operation.
- Single assembly instruction is Atomic.
- Similar behavior by:
  - Disabling interrupts
  - Use “Control Object”

Ways to Protect Shared Data

- There are 3 ways to protect shared data:
  - Disabling Interrupts,
  - Using Semaphores,
  - Disabling Task Switches:
    - You can protect shared data from an inopportune task switch by disabling task switches while you are reading or writing the shared data.
- More about this later...

Types of Interrupts

- Single vs. Multiple (nested)
  - Can one ISR interrupt another?
- Multiple Prioritized
  - Can a higher priority ISR interrupt a lower?
- Maskable vs. Non-Maskable
  - Can interrupts be disabled in code?
- Level vs. Edge Triggered
  - Sampled or Transition Induced

Single vs. Multiple Interrupts

- Some simple uCs only allow one interrupt
- Most uCs allow nested interrupts
  - 8051 family allows nested interrupts
    - Interrupts are disabled by default in the ISR.
  - ISR must re-enable interrupt(s)
  - Potential for calling ISR more than once
- In order to make good use of multiple interrupts, most processors also include some form of prioritized interrupts.

Prioritized Interrupts

- Higher priority events supersede low
- Fixed and variable priorities
- 8051 Interrupts:
  - Fixed priority within a level (hardware)
  - Two levels (low/high) under program control

Maskable vs. Non-maskable

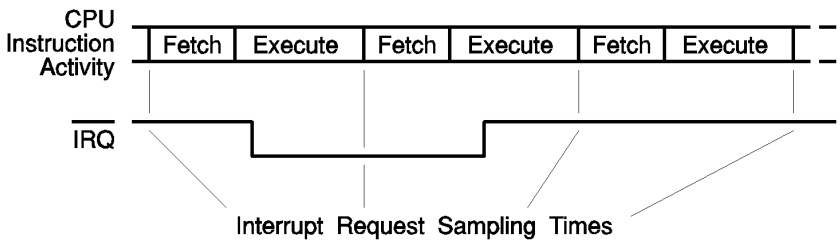
- Maskable Ints can be disabled by code
- Global disable
  - e.g.: The “Enable All” bit on the 8051
- Specific, individual interrupt enables
  - e.g.: Enable the Timer 1 interrupt
- Non-Maskable Interrrupts *can't* be turned off by the program.

Level vs. Edge

- Level:
  - CPU samples the interrupt request (IRQ) input at the end of every instruction, and calls the appropriate ISR if the IRQ is active
- Edge:
  - Signal transition (edge) on IRQ input is held in a latch until the CPU processes the interrupt.

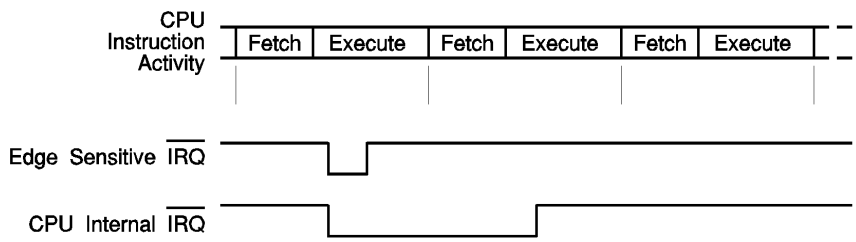
Level Sensitive Interrupt

- Interrupt input sampled by CPU at the end of every instruction, when interrupt is enabled
- Request must be active when sampled



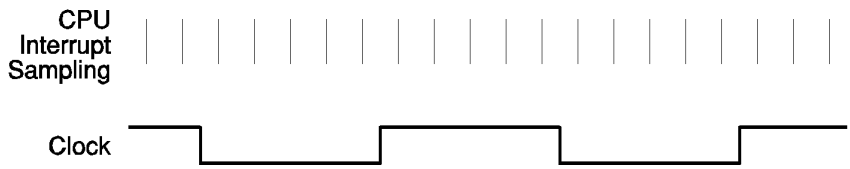
Edge Sensitive Interrupt

- Transition on input (edge) is latched internally by a flip flop in the CPU
- Internally latched version is then sampled just like a level sensitive input



When use Edge Sensitive?

- When Interrupt signal is too long
- e.g.: 60 Hz square wave for clock
- Level sensitive would respond more than once

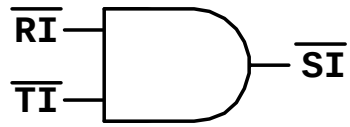


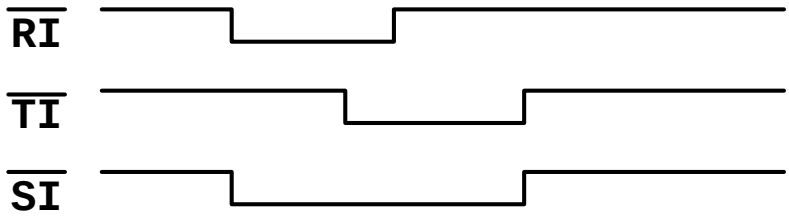
When use Edge Sensitive?

- When Interrupt signal is too short
- e.g.: very short pulse from a sensor
- Level Sensitive would miss the event

When use Level Sensitive?

- Common IRQ line
- More than one event
- Only one edge on output!





Vectored vs. Non-vectored

- Non-vectored:
  - All interrupts go to the same ISR address
  - ISR must poll hardware to identify source
- Vectored:
  - Each interrupt input calls a separate ISR address
  - ISR addresses may be fixed (8051)
  - Or may be stored in a table (x86)

8051 Interrupt Structure

- Multiple, Nested, Two Priorities
- Fixed Interrupt Vectors
- Interrupt Sources:
  - External Interrupt /INT0, /INT1
    - Level or Edge Sensitive Programmable
  - Timer/Counters 0, 1, 2
  - Serial Input and Output

Interrupt Sources

- Internal Interrupt Sources
  - Timers 0, 1 and 2
  - Serial Port Transmit or Receive
- External Interrupt Sources
  - /INT0 and /INT1
- Reset (sort of…)
  - Non-maskable, forces jump to addr Zero

8xx2 Interrupt Vectors

- Address, Source:
- 0003h, IE0 - External Interrupt 0
- 000Bh, TF0 - Timer 0
- 0013h, IE1 - External Interrupt 1
- 001Bh, TF1 - Timer 1
- 0023h, TI or RI - Serial Port Tx or Rx
- 002Bh, TF2 - Timer 2

Interrupt Enable IE 0A8h

7	6	5	4	3	2	1	0
EA	-	ET2	ES	ET1	EX1	ET0	EX0

- EA - Enable All Interrupts (Global enable)
- ES - Enable Serial Port
- ETn - Enable Timer n
- EXn - Enable External Interrupt n

Interrupt Priority IP 0B8h

7	6	5	4	3	2	1	0
-	-	PT2	PS	PT1	PX1	PT0	PX0

- 0==Low Priority, 1==High Priority
- PS - Serial Port Priority
- PTn - Timer n Priority
- PXn - External Interrupt n Priority

Priority (cont'd)

- Within a level, simultaneous interrupts are processed according to priority shown:
  - 1: IE0 External Interrupt 0
  - 2: TF0 Timer Interrupt Flag 0
  - 3: IE1 External Interrupt 1
  - 4: TF1 Timer Interrupt Flag 1
  - 5: RI or TI Transmit/Receive Interrupt
  - 6: TF2 Timer Interrupt Flag 2

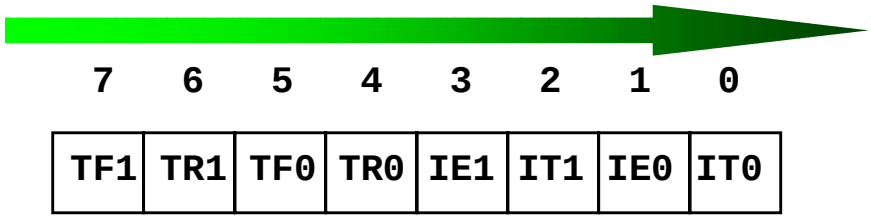
Clearing Interrupt Requests

- These are cleared when CPU calls ISR:
  - IE0 and IE1 External IRQ Flag
  - TF0, TF1, TF2 Timer IRQ Flags
- Must be explicitly cleared by program:
  - TI Serial Port Transmit Buffer Empty
  - RI Serial Port Receive Buffer Full
  - ex: CLR TI or CLR RI inside the serial ISR

External Interrupt Requests

- IE0 External Interrupt 0 Flag
- IE1 External Interrupt 1 Flag
  - IE0 in TCON.1, IE1 in TCON.3
  - Detects when INTx input pin goes low
  - IEx flags set when the INTx pin goes low
  - IEx Flags are cleared when ISR is called

Timer Control TCON 088h

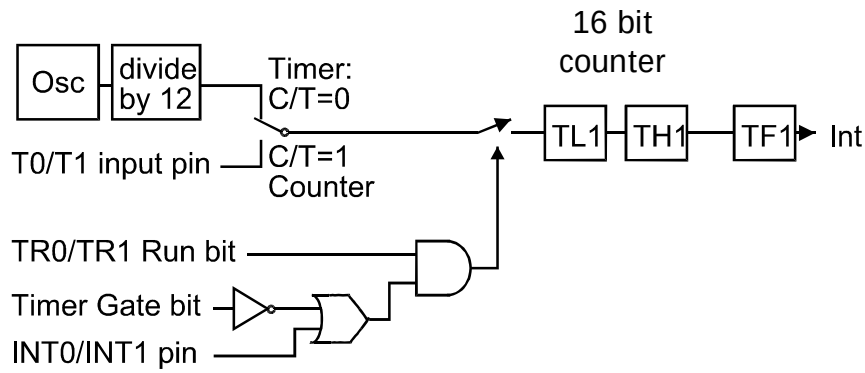


- TFx Timer x Overflow Flag (input)
- TRx Timer x Run Enable (control)
- IEx Ext Int Detect Flag (input)
- ITx Ext Int Edge Sensitive 1=Edge 0=Level

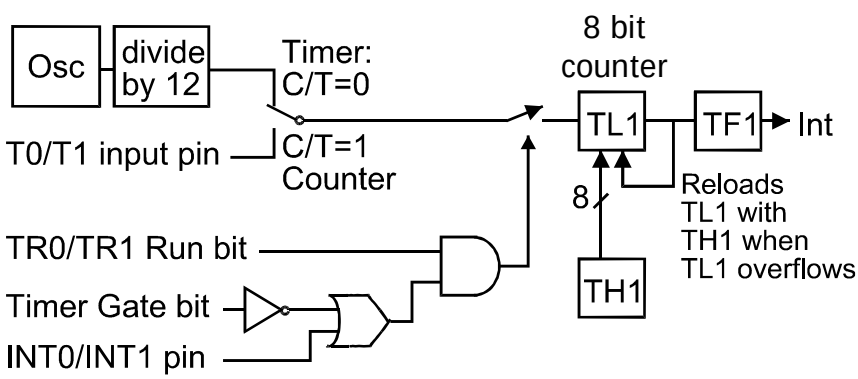
Timer Interrupts

- Timers interrupt on overflow if enabled
  - Count increments every 12 clock cycles
- Mode 1 used for a single 16 bit delay
  - One-shot delay, then it must be reloaded
- Mode 2 reloads automatically, 8 bit delay
  - TLx reloaded with THx when TLx overflows
  - Results in constant interrupt frequency
- Load timer with 2's complement of count

8051 Timer Mode 1



8051 Timer Mode 2



Summary

- Use only the elements needed.
- For simplest situations, shared memory may be “best” approach.
- Variety of IPC mechanisms exist:
  - Queues,
  - Mailboxes,
  - Signals/Events/Semaphores

Questions

- When is re-entrant code necessary?
- What happens if a critical code segment is interrupted?
- How does one avoid the problem?
- What is the difference between a flag and a semaphore?
- When must an edge sensitive interrupt be used?
- When must a level sensitive interrupt be used?

Homework #2

- Fix corrupt_uart program.
- Read Pont Ch. 7 and 8.
- Review Pont Ch. 9 and 10.
- Project proposal due next week!
- If we did not get back to you on your project ideas, they were all OK. Pick one you like the best and get to work.