CpE 213
Digital Systems Design

Lecture 23

Thursday 11/13/2003

UMR UNIVERSITY OF MISSOURI-ROLLA
The Name. The Degree. The Difference.

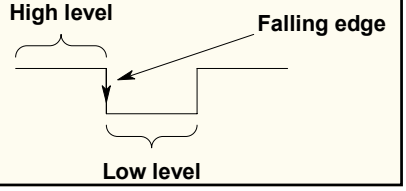
Announcements

- Rules for exam 2:
  - You can work in groups. Your writeup should be done completely on your own. Mention names of anyone you collaborated with.
  - Exam will be take-home, due at 4:30 pm on Friday 11/14.
  - Drop off at my office or leave with Katie (next door).
  - Email or slide under my door at your own risk.
  - Late submission will NOT be accepted.
- No class on Thursday Dec. 4.

Interrupts on the 8051
Chapter 6 of ISM

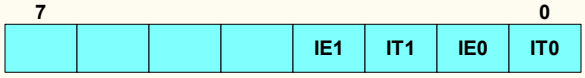
Activation of External Interrupts

- There are two activation levels for external hardware interrupts:
  (1) Level triggered.
  (2) Edge triggered.
- In the level-triggered mode, external interrupt pins are normally at high level and if a low-level signal is detected, it triggers the interrupt.
- Level-triggered interrupt is the default mode upon reset of the 8051.
- An edge-triggered interrupt is caused by a high to low transition (falling edge) being detected at the interrupt pin.
- To make edge interrupts functional, we must program the bits of the TCON register.



TCON's Interrupt-Related Bits

TCON (Timer Control) Register



Symbol	Function
IE1	External interrupt 1 edge flag. Set by hardware when a falling edge is detected on INT1; cleared by software or by hardware when CPU vectors to interrupt service routine.
IT1	External interrupt 1 type flag. Set/cleared by software for falling edge/low-level activated external interrupt.
IE0	External interrupt 0 edge flag.
IT0	External interrupt 0 type flag.

Interrupt Example

```
static int counter;
void Ex0Isr(void) interrupt 0 using 1{
  counter++;  }
void main(void) {
 IT0= 1; //enable neg edge triggered int0
 EX0= 1; //enable external interrupt 0
 EA= 1;  //enable global interrupts
 while(1) P1= 5*P2;
 }
```

Ex0 Interrupt Service Routine

```
static int counter;
void Ex0Isr(void) interrupt 0 using 1{
  counter++;  }
```

- Declare counter global so both ISR and Main can ‘see’ it.
- *Interrupt 0* makes this Ex0’s ISR
  - Ex0 interrupts vector to location C:0003
- *using 1* causes code to use Register Bank 1
  - Context switches to bank 1

Ex0 ISR in 8051 ASM

```
0000 C0E0           PUSH    ACC
0002 0500     R     INC     counter+01H
0004 E500     R     MOV     A,counter+01H
0006 7002           JNZ     ?C0005
0008 0500     R     INC     counter

000A D0E0 ?C0005:   POP     ACC
000C 32             RETI
```

- Note how ACC is saved and restored
- RETI re-enables interrupts

Interrupt Initialization Code

```
IT0= 1; //enable neg edge triggered int0
EX0= 1; //enable external interrupt 0
EA= 1;  //enable global interrupts
```

- IT0 is bit 0 of Timer Control Reg (TCON)
- EX0 and EA are bits in Interrupt Enable Register (IE)
- Code is executed once to configure and enable interrupts

Main Program

```
0006 E5A0 ?C0002:  MOV     A,P2
0008 75F005        MOV     B,#05H
000B A4            MUL     AB
000C F590          MOV     P1,A
000E 80F6          SJMP    ?C0002
```

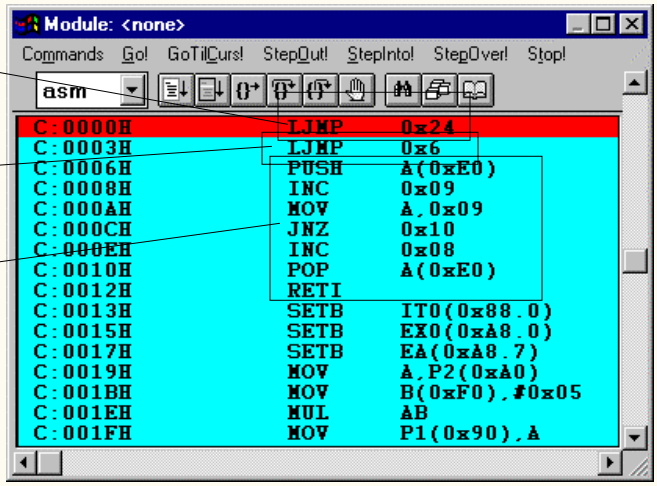
- This program gets interrupted by falling edges on INT0 (P3.2)
- Interrupts can occur at any time but are acknowledged only between instructions

Behind the scenes

Startup vector

Ex0 vector

Ex0ISR



Example

```
#include reg51.h

char getCharacter (void);
void sendCharacter (char);

void serial_int (void) interrupt 4{
   static char chr = '\0';
   if (RI == 1){
        chr = SBUF;
        RI = 0;
        TI = 1;}
   else if (TI == 1) {
        TI = 0;
        if (chr != '\0') {
               if (chr == '\r')
                      chr = '\n';
                      SBUF = chr;
                      chr = '\0'; } } }
```

Example (continued)

```
main(){
   SCON = 0x50; /* mode 1, 8-bit uart, enable receiver */
   TMOD = 0x20; /* timer 1, mode 2, 8-bit reload */
   TH1  = 0xFE; /* reload value for 2400 baud */
   ET0  = 0;
   TR1  = 1;    /* start the timer */
   TI   = 1;    /* clear the buffer */

   ES   = 1;
   EA   = 1;
   P0 = 0;

   while (1==1){
        unsigned int  i;
        for(i = 0; i < 60000; i++) {;}
        P0 = P0 + 1;
   }
}
```

Another Example

```
#include <REG51.H>
unsigned char count=0;
void counter(void) interrupt 0 {
   count++;
}
main(){
   unsigned char x=0;
   int i;

   IT0=0;
   INT0=1;
   PX0=0;
   EX0=1;
   EA=1;
```

(continued on next slide)

Example continued

```
for(i=1;i<=10;i++) {
      x++;

      if(x==5){
            x=0;
            INT0=1;
            INT0=0;
            INT0=1;
      }
   }
}
```

Example in ASM

```
;declare segments
mydata segment data
mycode segment code

;declare variables
rseg mydata
   x: DS 1
   count: DS 1
   stack: DS 20

;place jump to main at 1st code mem location
CSEG AT 0000H
   JMP main ; will jump to main function
```

Example continued

```
CSEG AT 0003H
       PUSH ACC
       PUSH PSW
       MOV A,count
       ADD A,#1
       MOV count, A
       POP PSW
       POP ACC
       RETI

rseg mycode
 main: MOV x,#00H
       mov count,#00H
       MOV SP, #stack
```

Example continued

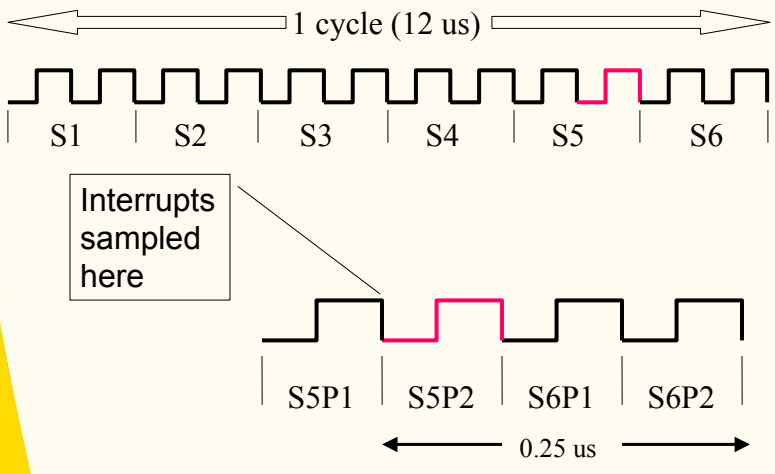
```
  SETB IT0
  SETB INT0
  CLR PX0
  SETB EX0
  SETB EA

 while:       INC x
  MOV A,#128
  CJNE A,x,while
  SETB INT0
  CLR INT0
  SETB INT0
  JMP while
END
```

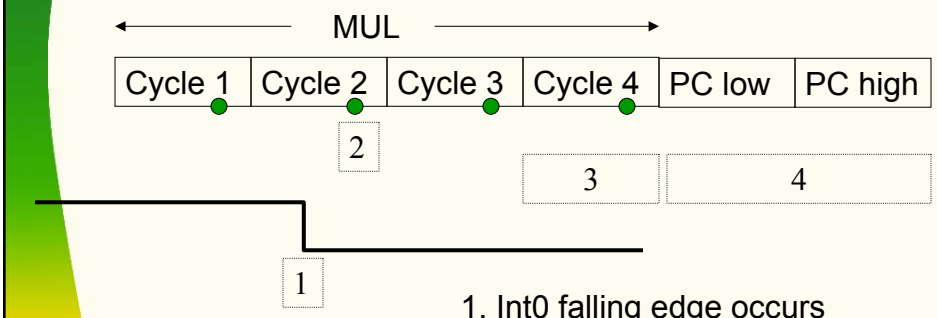
Interrupt Timing

- Interrupts sampled during S5P2
- Interrupts polled during last cycle of instruction
- PC pushed onto stack (2 cycles)
- Fetch first instruction of ISR
- A RETI or access to IE or IP lets one more instruction execute before pushing PC

Interrupt Timing



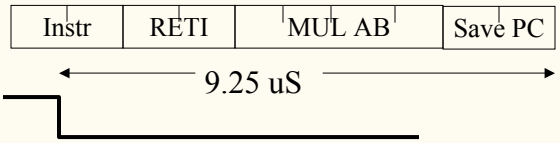
Interrupt Timing



- Interrupts Sampled

1. Int0 falling edge occurs
2. Ex0 interrupt detected
3. Interrupts polled. Start Ex0ISR
4. Push PC on stack (LCALL 3)

Worst Case Interrupt Timing



1. Level 1 event occurs just after sample in second to last cycle of instruction.
2. Interrupt sampled in last cycle of Instr
3. Interrupt polled in last cycle of RETI
4. RETI allows one more instruction: 4 cycle MUL
5. Plus two cycles to save PC
6. Equals 9.25 uS worst case (or is it really?)

Multitasking

- Main (background) with interrupts (foreground) is a simple *multitasking* system
- Preemptive vs non-preemptive tasking
  - non-preemptive: tasks executed in order
  - preemptive: one task can interrupt another
  - like single level vs multilevel priority interrupts

Non-Preemptive Tasking

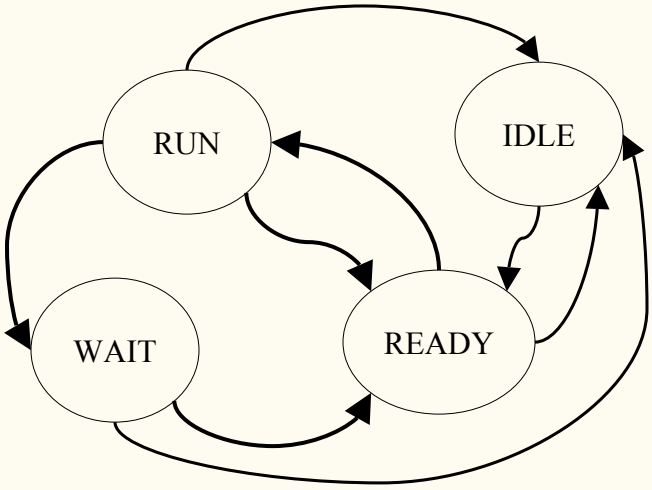
- Tasks executed in sequential order
- Sometimes called a ‘message loop’
- Examples:
  - X - windows
  - Microsoft Windows ™
- Tasks must be well-behaved
- A task executes until it must wait
- Easy for one task to ‘hog’ the CPU

Pre-emptive Tasking

- One task may interrupt another
- Tasks are assigned a priority
- A task executes until it must wait or a higher priority task becomes ready
- Examples:
  - QNX (a *real time* unix for x86 architecture)
  - RTX51/RTXtiny (RTOS for 8051)
  - VRTX (RTOS for large microprocessors)

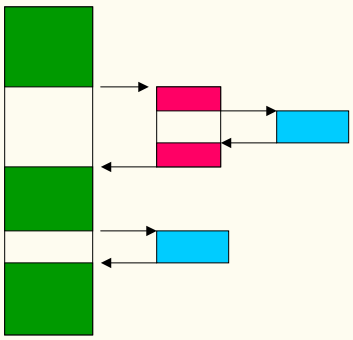
Task States

- Controlled by *task scheduler*

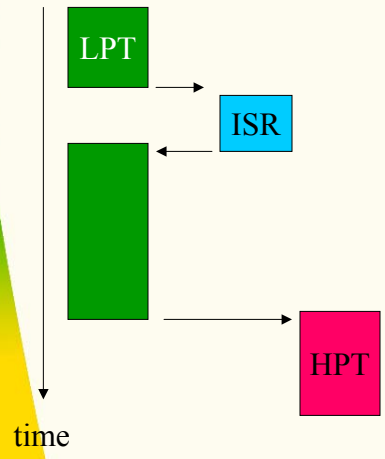


Main Program with interrupts

- Background task (main)
- Foreground interrupt service routines
  - timer ticks
  - I/O interrupts

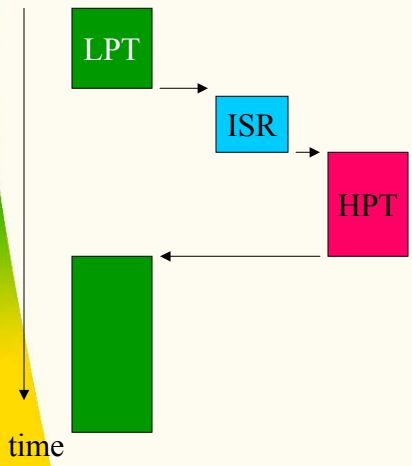


Non-Preemptive Multitasking



- Low priority task executes
- ISR makes high priority task ready
- LPT finishes
- Relinquishes control to HPT
- HPT executes...

Preemptive Multitasking



- Low priority task executes
- ISR makes high priority task ready
- Scheduler enables HPT
- HPT finishes
- LPT resumes execution...