CpE 213
Digital Systems Design
Interrupts

Lecture 20

Friday 11/11/2005

Overview

- Project deadline reminders
  - Demos today
  - Reports Monday
  - Peer reviews Wednesday
- Interrupts (Chapter 11)

Interrupts on the 8051
Chapter 11

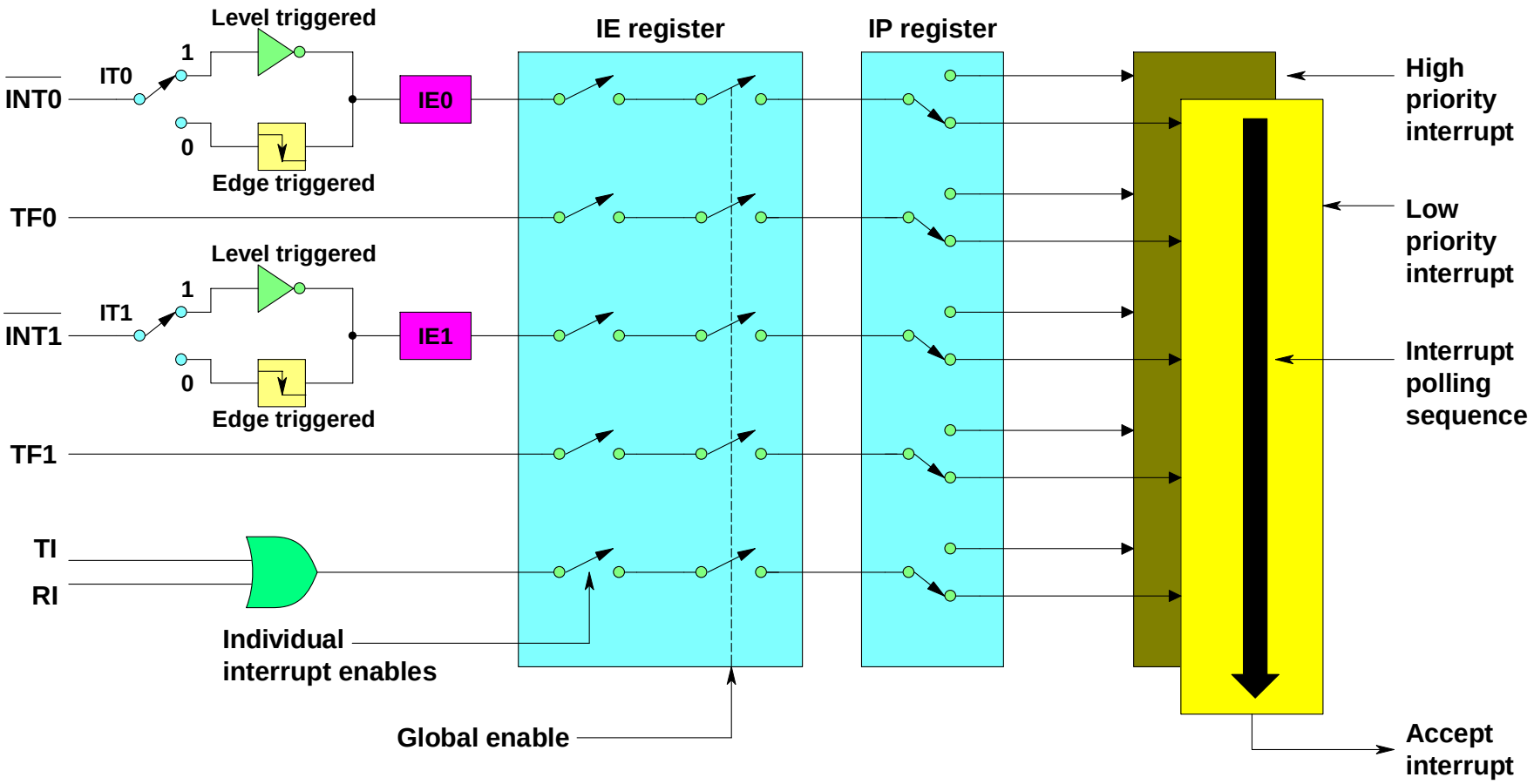
Review

- Interrupt: An unexpected (asynchronous) hardware-induced subroutine call (may also be referred to as an event or as an exception)
- Real-Time: Handling of interrupts (“events”) must be completed within fixed time constraints (“mission critical timing”)

Interrupt Sources for 8051

- The 8051 has 5 sources of interrupts:
  - 2 external and 3 internal sources
  - External INT0 pin (P3.2) — can connect to any external hardware.
  - Timer 0 — TF0 flag (internal).
  - External INT1 pin (P3.3) — can connect to any external hardware.
  - Timer 1 — TF1 flag (internal).
  - Serial port: just finished transmitting a character or have just received a character — RI&TI (internal).
- A signal from any of these sources can trigger execution of an ISR.
- Some manufacturers consider RESET to be an interrupt source as well.

Overview of 8051 Interrupt Structure



Interrupt Priority

- Each interrupt source can be individually programmed to one of two priority levels, by setting or clearing a bit in the IP register.
- A low priority interrupt can be interrupted by a higher priority interrupt, but not by another low priority interrupt.
- A high priority interrupt cannot be interrupted by any other interrupt source.
- If two requests of different priority levels are received simultaneously, the request of higher priority is serviced.
- Default is all low priority so only one interrupt at a time.

Simultaneous Interrupts of Same Priority

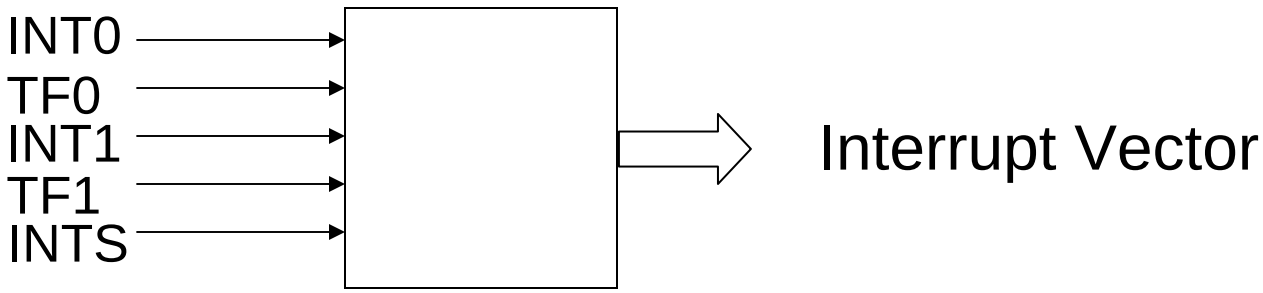
- If interrupts of the same priority level are received at the same time, an internal polling sequence determines which interrupt is serviced.
- The “priority within level” structure is only used to resolve simultaneous requests of the same priority level.

Interrupt Priority within Level Polling Sequence

Priority Within Level	Source
1 (Highest)	External Interrupt 0
2	Timer 0
3	External Interrupt 1
4	Timer 1
5 (Lowest)	Serial Port

Tie breaker

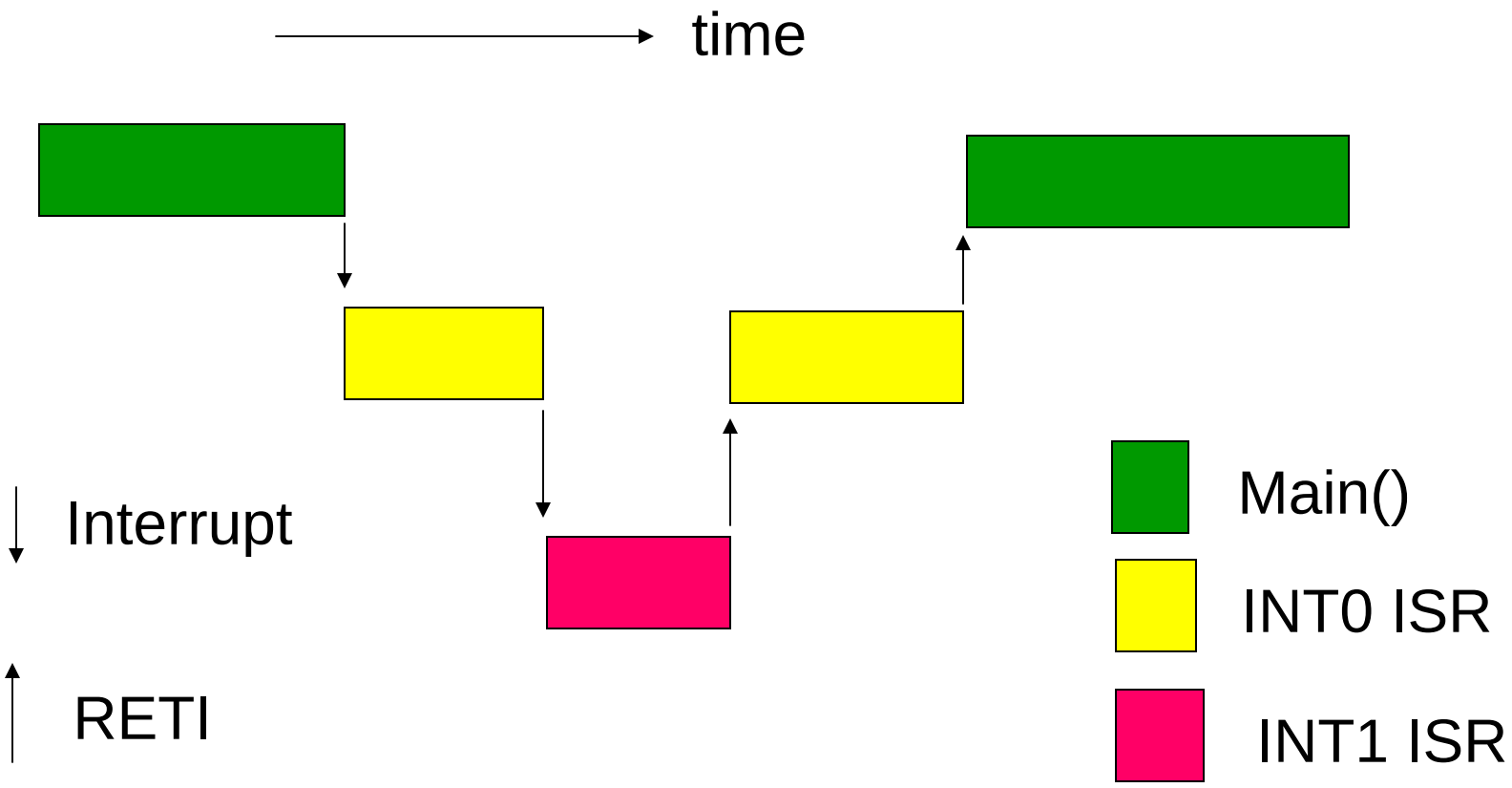
- Essentially a priority encoder
- Interrupt sources in, interrupt vector out



Priority Encoder

INT0	TF0	INT1	TF1	INTS	Vector
1	-	-	-	-	0x0003
0	1	-	-	-	0x000B
0	0	1	-	-	0x0013
0	0	0	1	-	0x001B
0	0	0	0	1	0x0023

Interrupt operation



Interrupt Priority Register (IP)

IP **Address = B8H** **Reset Value = XX00 0000B**
Bit Addressable

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

Symbol	Function
- - -	Reserved for future used.
- - -	Reserved for future used.
PS	Serial port interrupt priority bit.
PT1	Timer 1 interrupt priority bit.
PX1	External interrupt 1 priority bit.
PT0	Timer 0 interrupt priority bit.
PX0	External interrupt 0 priority bit.
	Priority Bit = 1 assigns high priority. Priority Bit = 0 assigns low priority.

Group Exercise

- a) Program the IP register to assign the highest priority to INT1, then
- b) Discuss what happens if INT0, INT1, and TF0 are activated at the same time.

Group Exercise

- a) Program the IP register to assign the highest priority to INT1

  ```
  MOV IP,#00000100B or SETB IP.2
  ```

- b) Discuss what happens if INT0, INT1, and TF0 are activated at the same time.

  INT1 was given high priority, so it will be serviced first, then INT0, then TF0.

Group Exercise

- Assume that after reset, the interrupt priority is set by the instruction "`MOV IP,#00001100B`." Discuss the sequence in which the interrupts are serviced.

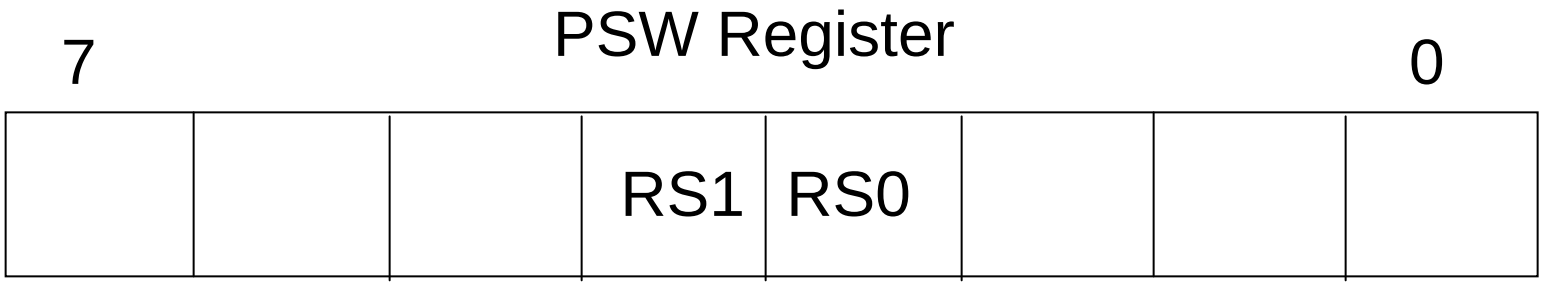
Group Exercise

- Assume that after reset, the interrupt priority is set by the instruction "`MOV IP,#00001100B`." Discuss the sequence in which the interrupts are serviced.
- Instruction sets INT1 and TF1 to high priority. According to the polling table:

Highest priority	INT1
	TF1
	INT0
	TF0
Lowest priority	INTS

Context Switching

- 8051 has four register banks:
  - D:0-7 bank 0
  - D:8-F bank 1
  - D:10-17 bank 2
  - D:18-1F bank 3
- PSW= PSW | 0x10; //use reg bank 2



Interrupt Handling

- Upon activation of an interrupt, the 8051 goes through the following steps:
  - The current instruction completes execution.
  - The PC is saved (pushed) on the stack, low-byte first.
  - The current interrupt status is saved internally.
  - Interrupts at equal or lower priorities are blocked.
  - PC is loaded with the vector address of the ISR.
  - The ISR executes.
- Upon executing the RETI instruction
  - The PC is popped from the stack.
  - The old interrupt status is restored.
  - Execution of the main program continues where it left off.

Return from ISR

- ISR - Interrupt Service Routine
- A normal RET can’t be used!
  - Interrupts would stay disabled
- RETI re-enables interrupts at that level and returns to interrupted program

Programming ISRs in Keil

- Keil compiler implements a function extension that explicitly declares a function as an interrupt handler.
- The extension is **interrupt**, and it must be followed by an integer specifying which interrupt the handler is for.
- For example:

```
/* This is a function that will be called
whenever a serial interrupt occurs. Note that
before this will work, interrupts must be
enabled.*/
void serial_int (void) interrupt 4 {
  ...
}
```

Interrupts in Keil

- Number following **interrupt** keyword is calculated by subtracting 3 from the interrupt vector address and dividing by 8 (don't memorize).

Interrupt	Vector Address	Interrupt Number
External 0	0003h	0
Timer 0	000Bh	1
External 1	0013h	2
Timer 1	001Bh	3
Serial	0023h	4

Caution about ISR Addresses

- Note that there is very little code space between the various interrupt handler addresses. They are only 8 bytes apart!
- For example if consecutive interrupts such as “external 0” and “timer 0” or “timer 0” and “external 1” are being used.
- If the first interrupt routine is more than 7 bytes long, then that routine will have to execute a jump to some other memory location where the ISR can be completed without overlapping with the starting address of the next ISR.
- Otherwise, you have to make sure that all your ISRs are limited to 8 bytes of code.
- Note that the main program, which must begin at address 0000h, must jump over any interrupt addresses that are in use.

Interrupts in Keil

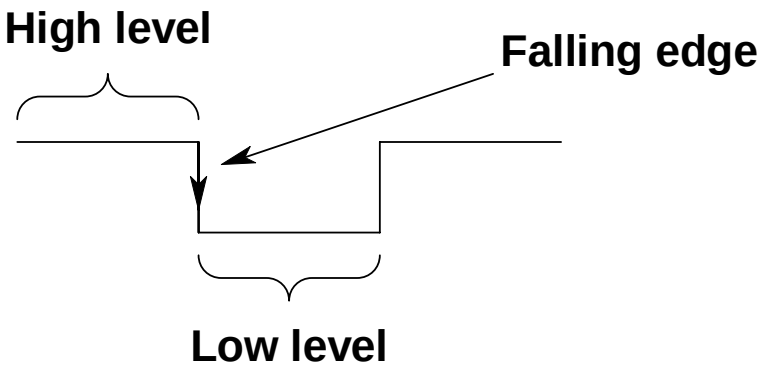
- Optional extension to interrupt: **using**
- Tells compiler to change to a new register bank prior to executing ISR instead of pushing all registers to stack.
- Reduces interrupt latency, should be used when quick execution is important.
- Example:

```
void serial_int (void) interrupt 4 using 1{
...}
```

- Interrupts of the same priority can use the same register bank. Why?

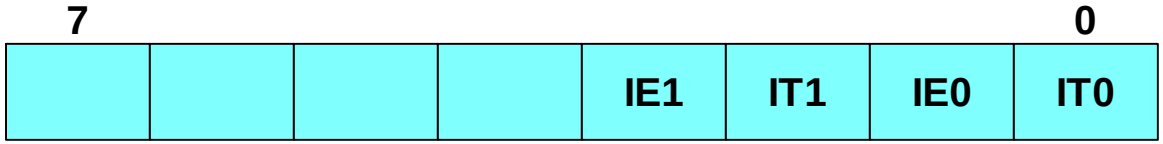
Activation of External Interrupts

- 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
```

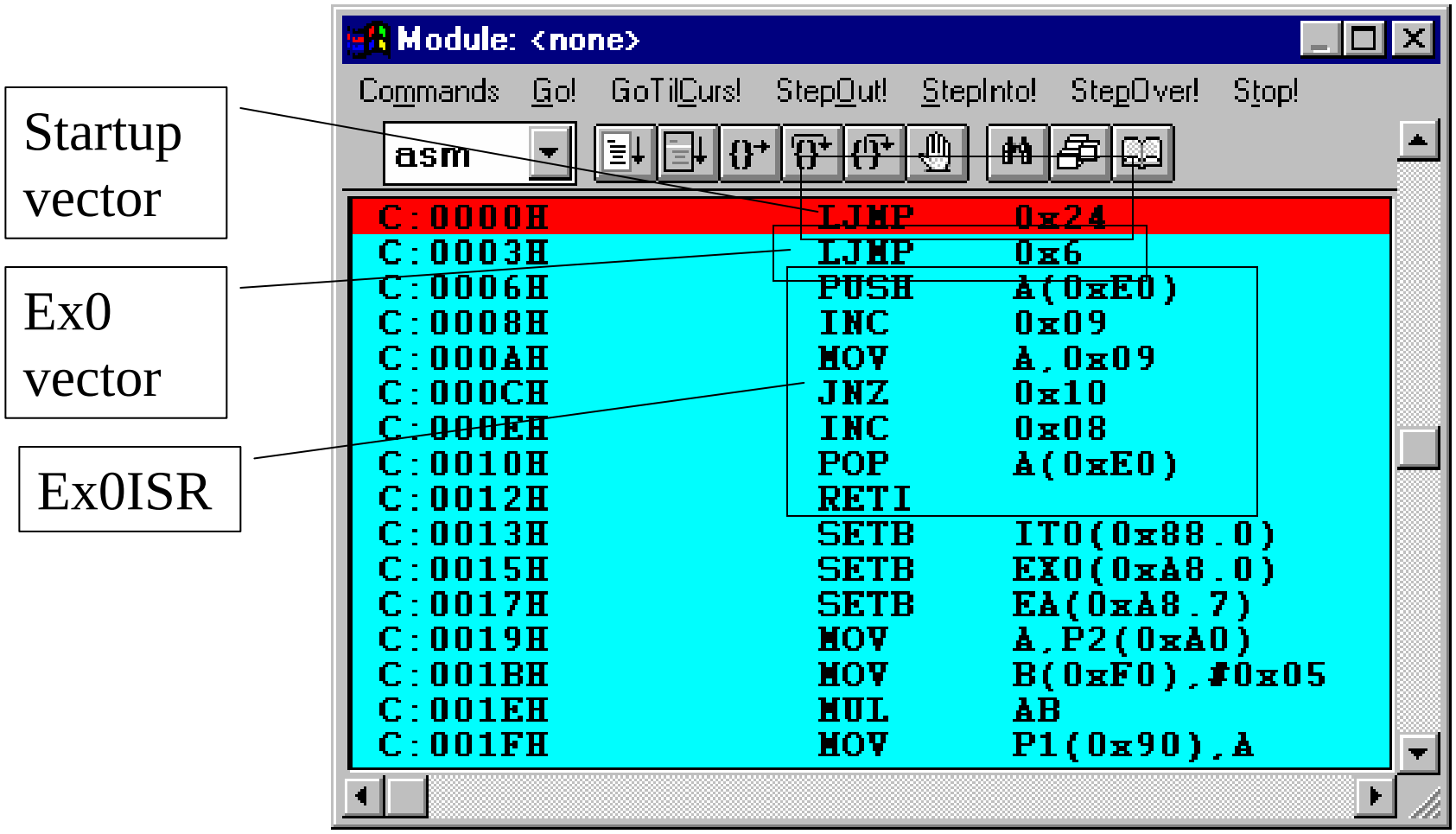
- IT0 is bit 0 of Timer Control Reg (TCON)
- EX0 and EA are bits in Interrupt Enable Register (IE)
- Code is executed once to configure and enable interrupts

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



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;
   }
}
```

For the next lecture

- Review lecture notes and Chapter 11.