Interrupts

- Interrupt
    - Interrupt handler (ISR)
    - ISR Vector

- Types of Interrupts
    - Internal and External
    - Maskable and Non-Maskable Interrupts
    - Vectored and Non-Vectored Interrupts

- ISR Prototype in C51 Compiler
- Interrupt# and Vector Table
- Register Bank for ISR

- Interrupt System in AT89C51ED2

- An example ISR
- Importance of NOAREGS pragma
- Link between ISR function and Interrupt
- Disassembly view of ISR and Context Switching Process

- SFR – Setting up Interrupt Sources and Priorities
    - Interrupt Enable Registers (IE)
        - Enabling an interrupt source
        - Executing time-critical code
        - Polling sequence of interrupts (Default Priorities of Interrupts)
    - Interrupt Priorities - Interrupt Priority Register (IP)

- Interrupt Overheads
    - Interrupt Latency
    - Context Saving Time
    - Time to identify the (Device Id Code) ISR to be executed
    - ISR execution time
    - Context Restore Time

- Interrupt Deadline

- Hands-on (Pseudo-code)
    - Toggle LED (L24) on external interrupt (EX0 / EX1)
    - Generation of Square Waveform of 200us period on timer overflow interrupt (ET0 / ET1)
    - Receive and Send (echo-back) byte on serial port interrupt (ES)
    - Using external interrupt (EX0 / EX1)) to change the direction of stepper motor rotation (clockwise / anti-clockwise)

ISR Prototype in C51 Compiler

- The C51 compiler lets you write interrupt service routines in C.
- The compiler generates very efficient entry and exit code and accommodates register bank switching.
- Interrupt routines are declared as follows:

void *function_name* (void) interrupt *interrupt_number* [using *register_bank*]

- The interrupt_number determines the interrupt vector address of the interrupt function.

Interrupt# and Vector Table

- Use the following table to determine the interrupt number.

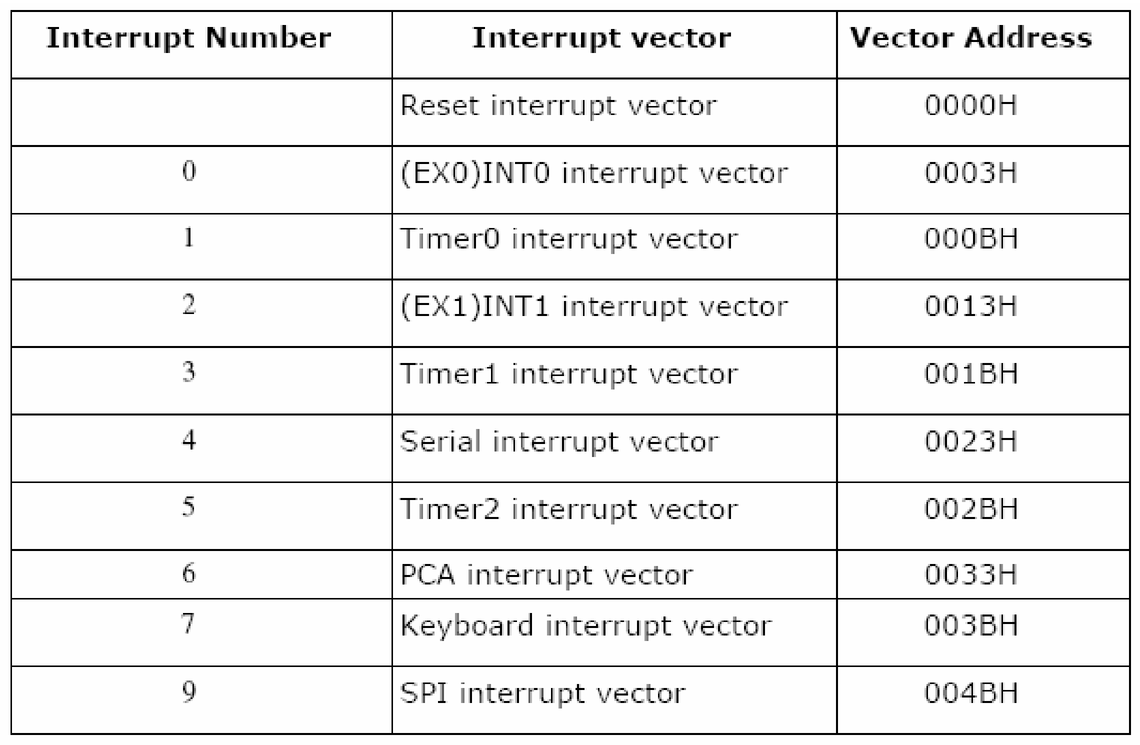
Interrupt Number	Address	Interrupt Number	Address
0 (EXTERNAL INT 0)	0003h	16	0083h
1 (TIMER/COUNTER 0)	000Bh	17	008Bh
2 (EXTERNAL INT 1)	0013h	18	0093h
3 (TIMER/COUNTER 1)	001Bh	19	009Bh
4 (SERIAL PORT)	0023h	20	00A3h
5 (TIMER/COUNTER 2)	002Bh	21	00ABh
6 (PCA)	0033h	22	00B3h
7	003Bh	23	00BBh
8	0043h	24	00C3h
9	004Bh	25	00CBh
10	0053h	26	00D3h
11	005Bh	27	00DBh
12	0063h	28	00E3h
13	006Bh	29	00EBh
14	0073h	30	00F3h
15	007Bh	31	00FBh

Register Banks for ISR

- The using attribute lets you specify a register_bank that is selected during the execution of the interrupt function.
- Small interrupt routines might be more efficient without the using attribute, since they use the default register bank 0.
- You should compare the assembly code generated both with and without the using directive to see which is more efficient in your application.

Interrupt system in AT89C51ED2

- The AT89C51ED2 has a total of 7 interrupt vectors,
  - two external interrupts (/INT0 and /INT1),
  - three timer interrupts (timer0, 1 and 2),
  - the serial port interrupt,
  - SPI interrupt and
  - the PCA interrupt (Programmable Counter Array)



Interrupt Number	Interrupt vector	Vector Address
	Reset interrupt vector	0000H
0	(EX0)INT0 interrupt vector	0003H
1	Timer0 interrupt vector	000BH
2	(EX1)INT1 interrupt vector	0013H
3	Timer1 interrupt vector	001BH
4	Serial interrupt vector	0023H
5	Timer2 interrupt vector	002BH
6	PCA interrupt vector	0033H
7	Keyboard interrupt vector	003BH
9	SPI interrupt vector	004BH

An example ISR

- The following example program shows a typical interrupt function:

```
#include <reg51.h> // Special Function Registers of 80C51 CPU
#pragma NOAREGS    // do not use absolute register symbols (ARx)
                   // for functions called from interrupt routines.

static void HandleTransmitInterrupt (void) {
:
:
}
static void HandleReceiveInterrupt (void) {
:
:
}
#pragma AREGS      // for other code it is save to use ARx symbols

static void com_isr (void) interrupt 4 using 1 {
  if (TI) HandleTransmitInterrupt ();
  if (RI) HandleReceiveInterrupt ();
}
```

- In the example above an interrupt service routine for interrupt number 4 is defined.
- The name of the interrupt function is com_isr.

Another example ISR for Timer 2 overflows interrupt:

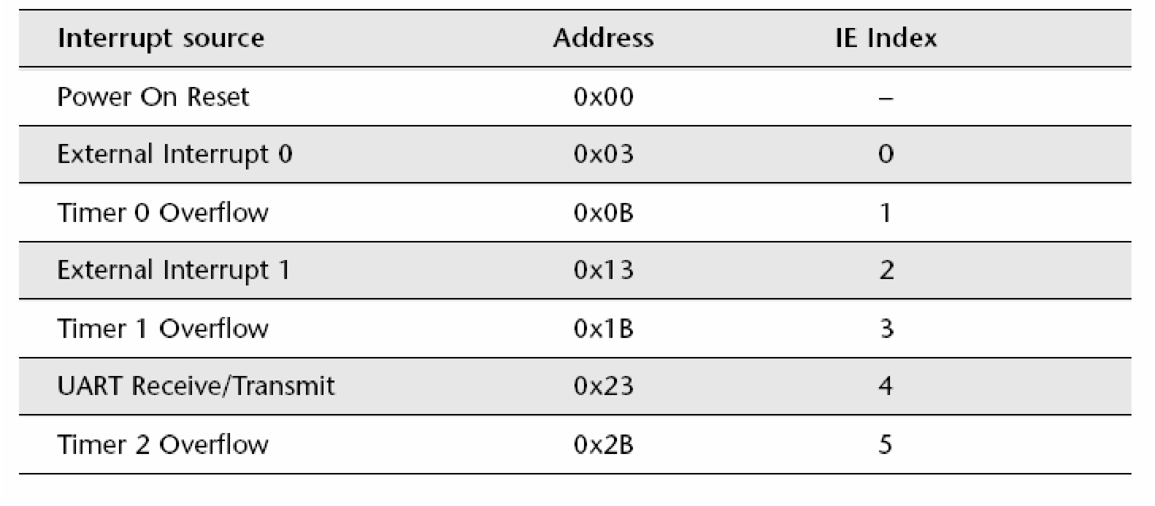
```
Void X() interrupt 5
{
        //place required code here for the ISR

}
```

Importance of NOARGEGS pragma

- Functions that are invoked from an interrupt procedure should be compiled with the NOAREGS directive. This ensures that the compiler does not generate absolute register accesses.

Link between ISR function and interrupt

- The link between this ISR function and the timer overflow is made using the Keil keyword ***'interrupt'*** (included after the function header in the function definition):
- To understand where the '5' comes from, note that the interrupt numbers used in ISRs directly correspond to the enable bit index of the interrupt source in the 8051IE SFR.
- That is, bit 0 of the IE register will be linked to a function using 'interrupt 0'
- Table below shows the link between the interrupt sources and the required interrupt numbers for the original 8051/8052.
- 8051 interrupt sources. Please note that many 8051s have further interrupt sources: refer to the manufacturer's documentation for details of the required interrupt numbers



Interrupt source	Address	IE Index
Power On Reset	0x00	–
External Interrupt 0	0x03	0
Timer 0 Overflow	0x0B	1
External Interrupt 1	0x13	2
Timer 1 Overflow	0x1B	3
UART Receive/Transmit	0x23	4
Timer 2 Overflow	0x2B	5

Disassembly View of Context switching process

- Once the interrupt is invoked, the entry code saves CPU registers and selects register bank 1, when the interrupt service routine exits, the CPU registers are restored.

- The following listing shows the code generated by the C51 compiler for the above interrupt routine. Note that the register bank context is swapped on entry to the interrupt routine and is restored on exit.

-

```
                ; FUNCTION com_isr (BEGIN)
0000 C0E0   PUSH  ACC       ; Save the Accumulator and Data
Pointer
0002 C083   PUSH  DPH
0004 C082   PUSH  DPL
0006 C0D0   PUSH  PSW       ; Save PSW (and the current Register Bank)
0008 75D008 MOV   PSW,#08H   ; This selects Register Bank 1
:
:
0052 D0D0   POP   PSW       ; Restore PSW (and prior reg bank)
0054 D082   POP   DPL
0056 D083   POP   DPH
0058 D0E0   POP   ACC       ; Restore the Accumulatorand DPTR
005A 32     RETI
         ; FUNCTION com_isr (END)
```

SFR - Setting up Interrupt Sources and Priorities

Interrupt Enable Registers

- The 8051 provides interrupt services for many on-chip peripheral events.

- Interrupts are globally enabled and disabled using the EA bit of the Interrupt Enable (IE) SFR.

  - When EA is set to 1, interrupts are enabled.
  - When EA is set to 0, interrupts are disabled.

- Each interrupt is individually controlled through bits in the IE SFR.

- On some 8051 derivatives there may be more than one IE register.
  - Check the documentation for the chip you use to determine what interrupts is available.

- **Interrupt Enable Register (IEN)**
  - EA / EC / ET2 / ES / ET1 / EX1 / ET0 / EX0

Enabling an Interrupt Source

- In order for an interrupt service routine to execute when an interrupt occurs, the interrupt enable SFR for that interrupt must be set and the EA SFR must be set.

- By default at power-up, all interrupts are disabled.
    - This means that even if, for example, the TF0 bit is set, the 8051 will not execute the interrupt.

- Your program must specifically tell the 8051 that it wishes to enable interrupts and specifically which interrupts it wishes to enable.
    - Your program may enable and disable interrupts by modifying the IE SFR (A8h):

Bit	Name	Bit Address	Explanation of Function
7	EA	AFh	Global Interrupt Enable/Disable
6	-	AEh	Undefined
5	-	ADh	Undefined
4	ES	ACh	Enable Serial Interrupt
3	ET1	ABh	Enable Timer 1 Interrupt
2	EX1	AAh	Enable External 1 Interrupt
1	ET0	A9h	Enable Timer 0 Interrupt
0	EX0	A8h	Enable External 0 Interrupt

- As you can see, each of the 8051s interrupts has its own bit in the IE SFR.

- You enable a given interrupt by setting the corresponding bit.

- For example, if you wish to enable Timer 1 Interrupt, you would execute either:

MOV IE,#08h
or
SETB ET1

- Both of the above instructions set bit 3 of IE, thus enabling Timer 1 Interrupt.

- Once Timer 1 Interrupt is enabled, whenever the TF1 bit is set, the 8051 will automatically put "on hold" the main program and execute the Timer 1 Interrupt Handler at address 001Bh.

- However, before Timer 1 Interrupt (or any other interrupt) is truly enabled, you must also set bit 7 of IE.

- Bit 7, the Global Interrupt Enable/Disable, enables or disables all interrupts simultaneously.

    - That is to say, if bit 7 is cleared then no interrupts will occur, even if all the other bits of IE are set.

    - Setting bit 7 will enable all the interrupts that have been selected by setting other bits in IE.

Executing Time-Critical Code

- EA bit in IE SFR for running time-critical code
- This is useful in program execution if you have time-critical code that needs to execute.
- In this case, you may need the code to execute from start to finish without any interrupt getting in the way.
- To accomplish this you can simply clear bit 7 of IE (CLR EA) and then set it after your time critical code is done.
- So, to enable the Timer 1 Interrupt the most common approach is to execute the following two instructions

```
SETB ET1
SETB EA
```

- Thereafter, the Timer 1 Interrupt Handler at 01Bh will automatically be called whenever the TF1 bit is set (upon Timer 1 overflow).

Polling Sequence of Interrupts (Default Priorities of Interrupts)

- The 8051 automatically evaluates whether an interrupt should occur after every instruction.
- When checking for interrupt conditions, it checks them in the following order:
    - External 0 Interrupt
    - Timer 0 Interrupt
    - External 1 Interrupt
    - Timer 1 Interrupt
    - Serial Interrupt
- This means that if a Serial Interrupt occurs at the exact same instant that an External 0 Interrupt occurs, the External 0 Interrupt will be executed first and the Serial Interrupt will be executed once the External 0 Interrupt has completed.

Interrupt Priorities

Interrupt Priority Register

- In addition to the IE register, many 8051 derivatives allow you to assign a priority level to each interrupt using an Interrupt Priority (IP) SFR.
- Check the documentation for your microcontroller to determine how to use the interrupt priorities.
- The 8051 offers two levels of interrupt priority: high and low.
- By using interrupt priorities you may assign higher priority to certain interrupt conditions.

An Example

- For example, you may have enabled Timer 1 Interrupt which is automatically called every time Timer 1 overflows.
- Additionally, you may have enabled the Serial Interrupt which is called every time a character is received via the serial port.
- However, you may consider that receiving a character is much more important than the timer interrupt.
- In this case, if Timer 1 Interrupt is already executing you may wish that the serial interrupt itself interrupts the Timer 1 Interrupt.
- When the serial interrupt is complete, control passes back to Timer 1 Interrupt and finally back to the main program.
- You may accomplish this by assigning a high priority to the Serial Interrupt and a low priority to the Timer 1 Interrupt.
- Interrupt priorities are controlled by the **IP** SFR (B8h). The IP SFR has the following format:

Bit	**Name**	**Bit Address**	**Explanation of Function**
7	-	-	Undefined
6	-	-	Undefined
5	-	-	Undefined
4	PS	BCh	Serial Interrupt Priority
3	PT1	BBh	Timer 1 Interrupt Priority
2	PX1	BAh	External 1 Interrupt Priority
1	PT0	B9h	Timer 0 Interrupt Priority
0	PX0	B8h	External 0 Interrupt Priority

- When considering interrupt priorities, the following rules apply:
  - Nothing can interrupt a high-priority interrupt--not even another high priority interrupt.
  - A high-priority interrupt may interrupt a low-priority interrupt.
  - A low-priority interrupt may only occur if no other interrupt is already executing.
  - If two interrupts occur at the same time, the interrupt with higher priority will execute first. If both interrupts are of the same priority the interrupt which is serviced first by polling sequence will be executed first.

Hands-on (Pseudo-code)

- Toggle LED (L24) on external interrupt (EX0 / EX1)

```
//main
        //Clear external interrupt flag (IE0=0)
        //Set interrupt type flag (IT0=1)
        //Enable global interrupt (EA=1)
        //Enable external interrupt (IE0 = 1)
        //Turn off LED
        //go into infinite loop

//ISR
//void ie0_isr(void) interrupt 0
        //toggle LED
        //reset external interrupt flag (IE0 or IE1 = 0)
```

- Generation of Square Waveform of 200us period on timer overflow interrupt (ET0 / ET1)

```
//main
        //configure timer 0 in 8b auto reload mode (TMOD=0x20)
        //load initial value of timer 0 (TH0=0xA4)
        //run the timer0 (TR0=1)

        //Enable the global interrupt (EA=1)
        //enable the timer 0 overflow interrupt (ET0=1)
        //Or set IEN0=0x82

//ISR
//void timer0_isr(void) interrupt 1
        //toggle the pin
```

- Receive and Send (echo-back) byte on serial port interrupt (ES)

```
//main
    //configure serial port for send/rx in mode-2 (SCON=0x50)
    //configure timer 1 for baud rate
        //timer 1 in 8b auto reload mode (TMOD=0x020)
        //load initial value of timer 1 (TH1=0xfd)
        //run the timer 1 (TR1=1)
    //enable global interrupt (EA=1)
    //enable serial interrupt (ES=1)
    //Or set IEN0=ox90
//ISR
//void uart_isr(void) interrupt 4
    //if TI flag is set
        //reset TI flag (TI=0)
    //if RI flag is set
        //read byte from SBUF into variable (var = SBUF)
        //reset the RI flag (RI=0)
        //assign the variable back to SBUF (SBUF=var)
```

- Using external interrupt (EX0 / EX1)) to change the direction of stepper motor rotation (clockwise / anti-clockwise)

```
//declare a static bit variable for direction (static bit Dir=0)
//main
    //enable global interrupt (EA=1)
    //enable external interrupt 0 (EX0=1)
    //check on dir flag and write data for CW or ACW direction

//ISR
//void ChangeDir(void) interrupt 0
    //toggle the direction bit
```