Interrupt Servicing in ARM Processors

---

1 Overview

In this lecture, we will be working on interrupts (INT). Interrupts are a powerful method of timely response to internal or external events. Some sample events are:

- Timer has counted to zero
- ADC has completed conversion
- New data has arrived in the UART
- GPIO pin voltage has risen to logic high
- Data transfer complete
- Division by zero exception
- Illeagal instruction was encountered in the program
- Processor voltage critically low
- Processor reset
- etc. There are around 250 possible events.

As you may see from this list, the events are not related with the program flow. They occur independently. They are caused either by external interactions such as data transfer, or they are internal events such as timer has counted to zero. One thing in common to all interrupts is that the main program flow does not know when (or even, 'if') the events will happen.

They work as follows:

1. We associate an event with an interrupt service routine (ISR); a special function.
2. When the event occurs, the associated interrupt service routine is directly called **by the hardware**.

When the event takes place, an interrupt is **asserted**. The normal execution of the main program is automatically **interrupted** by processor hardware and **the associated ISR is called**. When the ISR completes execution, main program is resumed from where it left off. Main program does not get notified of the interruption.

Since there are many possible events, each INT has a designated number. The number of the INT identifies the event that causes it.

The interrupt mechanism:

- Makes it possible to rapidly respond to events.
- Relieves the main program from the overhead of constantly checking for events.

Think of how we answer the telephone. When we hear the ringer, we **interrupt** whatever we are doing and respond to the telephone call. Another possible method for responding is called **polling**. In polling, frequent checks whether an event has occurred, must be inserted in the main program. A real life analogy is: Assume that the telephone had no ringer. We would need to check every few seconds if a call was incoming, by lifting the telephone handset. This obviously causes an unacceptable overhead. Interrupts remove the need for this overhead.

2 The method and the hardware

The general method of servicing interrupts is **called vectored interrupt processing**, and handled by the "**Nested Vectored Interrupt Controller" (NVIC)** module in the ARM microcontroller family. All possible events in the processor are categorized in a table called the "**Interrupt Vector Table**". The vector table relates each event with a function. When the event occurs, the associated interrupt service function (**"interrupt service routine", ISR** is called.

Each event has two important bits (or flags). One is the **interrupt flag** which is set whenever the event happens. The programmer is free to query this flag whenever they want. They can also reset the interrupt flag if they want. The other is **interrupt enable** bit. If this bit is set, an interrupt is asserted whenever the interrupt flag is set. If this bit is not set, the interrupt flag sill not cause an interrupt. Therefore, the programmer can decide to ignore or attend to each event. To ignore the event, the corresponding interrupt enable bit is reset. All interrupts can also be disabled permanently or temporarily using the **global interrupt enable (GIE)** bit (see the processor datasheet for how to do this).

The enabled interrupts are serviced as follows, and is summarized in Fig 1:

1. The **processor hardware** checks for the occurrance of each event at each clock cycle (several million times/second).
2. Event occurs.
3. The processor hardware looks up the event and matches the event with an INT number. Assume it is INT2.
4. The processor hardware checks if the global interrupt enable (GIE) bit and the **interrupt enable bit for that interrupt** is set.
5. The processor hardware determines the associated interrupt service routine (say: `ISR2()`) by looking it up in the interrupt vector table.
6. `main()` program is suspended.
7. Execution branches to function `ISR2()`.
8. After `ISR2()` completes, execution returns to the same place that `main()` was suspended.
9. `main()` is **not** notified that the event has happened.

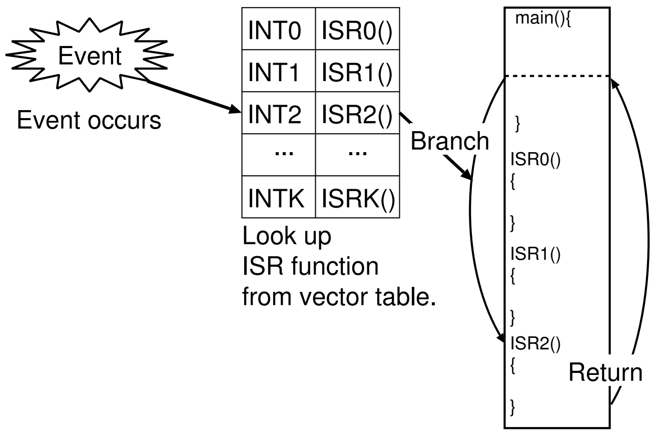


Figure 1: Overview of the INT servicing sequence.

3 Sample functions for interrupt servicing

In this lecture, 3 sample projects about interrupts are provided:

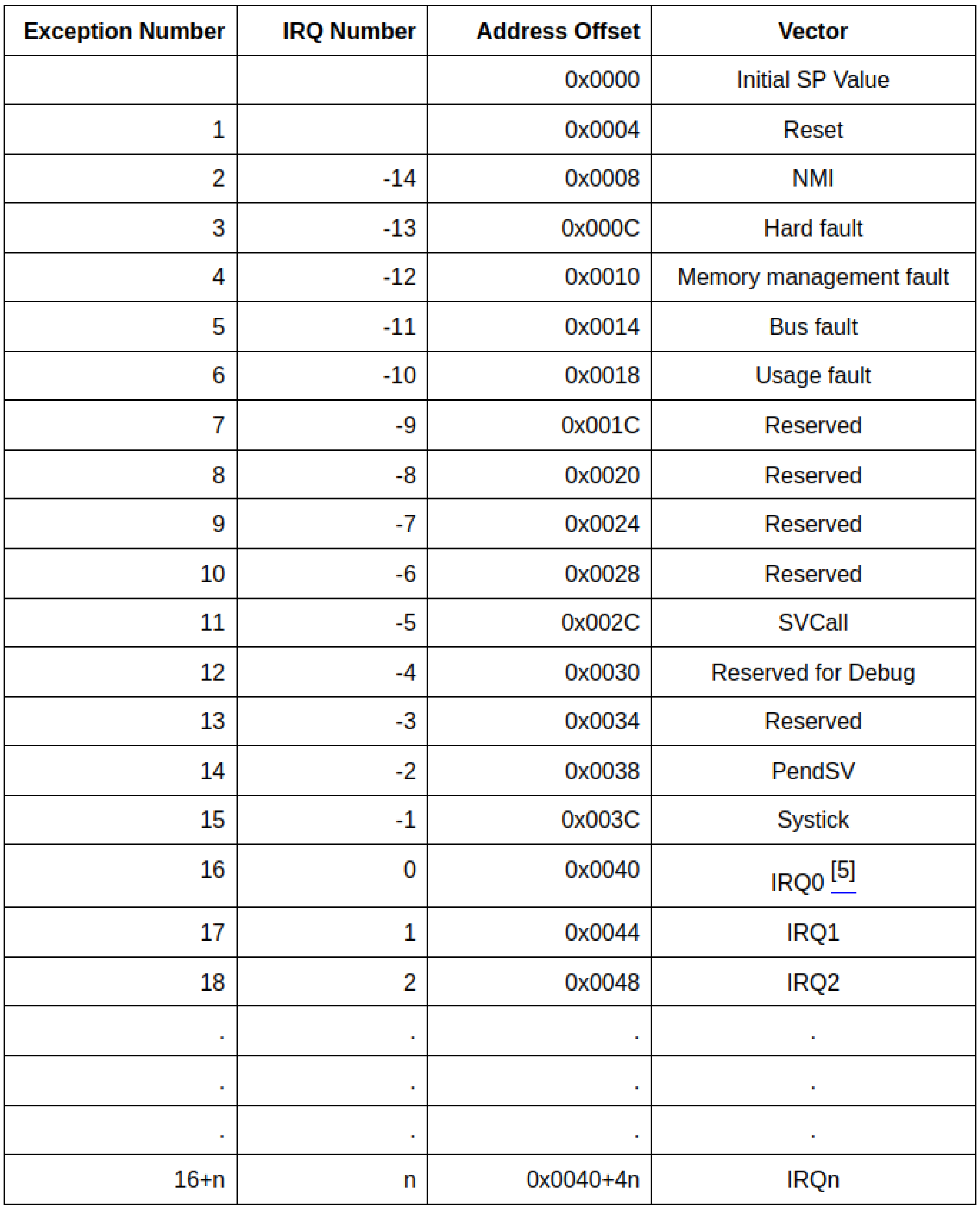
- `Blink_Systick_PSL`: A simple periodic INT generation. An interrupt is generated every millisecond using the SysTick peripheral that is common to all ARM processors.
- `MRT_INT`: Another simple periodic INT generation, this time using the MRT peripheral that is specific to LPC microcontroller family.
- `pin_interrupt`: An INT is issued when a button is pressed.

There are comprehensive explanations in the `Readme.txt` files or `PDF` files in the respective folders.

4 Final remarks

The interrupt vector table in ARM M0 processors looks like Fig.2. The first 16 entries are exaclt the same in all ARM processors. The remaining ones are implemented differently by each manufacturer. Note that the convention puts negative numbers for the first 16 entries in the table and starts numbering from '0' after that. The interrupts with positive numbers can be found in `Chapter 4: LPC82x Nested Vectored Interrupt Controller` of the reference manual, in `Table 5.Connection of interrupt sources to the NVIC`. Negative numbered interrupts are just a naming convention and have no significance for the programmer.

Ahmet Onat 2022



Exception Number	IRQ Number	Address Offset	Vector
		0x0000	Initial SP Value
1		0x0004	Reset
2	-14	0x0008	NMI
3	-13	0x000C	Hard fault
4	-12	0x0010	Memory management fault
5	-11	0x0014	Bus fault
6	-10	0x0018	Usage fault
7	-9	0x001C	Reserved
8	-8	0x0020	Reserved
9	-7	0x0024	Reserved
10	-6	0x0028	Reserved
11	-5	0x002C	SVCall
12	-4	0x0030	Reserved for Debug
13	-3	0x0034	Reserved
14	-2	0x0038	PendSV
15	-1	0x003C	Systick
16	0	0x0040	IRQ0 [5]
17	1	0x0044	IRQ1
18	2	0x0048	IRQ2
.	.	.	.
.	.	.	.
.	.	.	.
16+n	n	0x0040+4n	IRQn

Figure 2: ARM interrupt vector table entries.