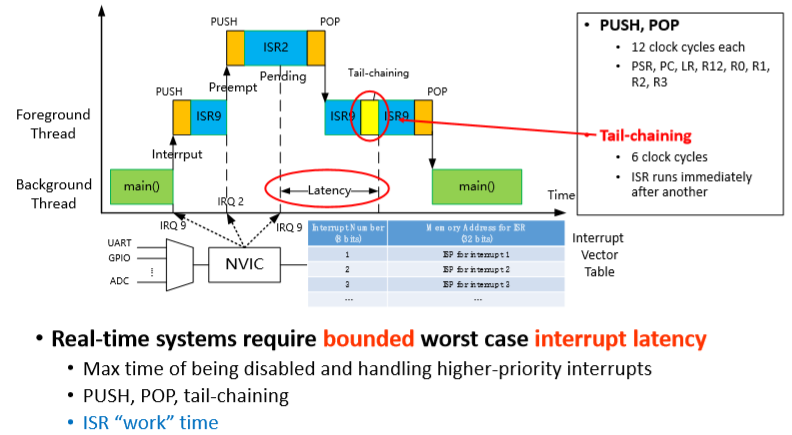
**Interrupt Processing II**

Summary in Interrupt I:

* • (Mostly peripheral) events requiring attention  interrupt requests
* • Microcontroller stops to run interrupt service routine (ISR) • ISR can be executed between any two instructions
* • Microcontroller returns to code prior to interrupt
* • More efficient than polling for handling asynchronous events

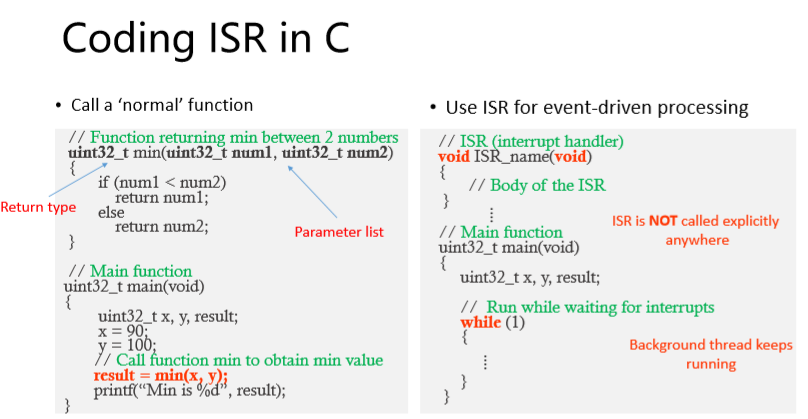
Tail-chaining =  handling pending interrupts without repeating the stacking

Interrupt vector table

* Interrupt vector: starting memory address of an ISR
* • Table lookup via interrupt number (8 bits, signed)

Nested vectored interrupt controller (NVIC)

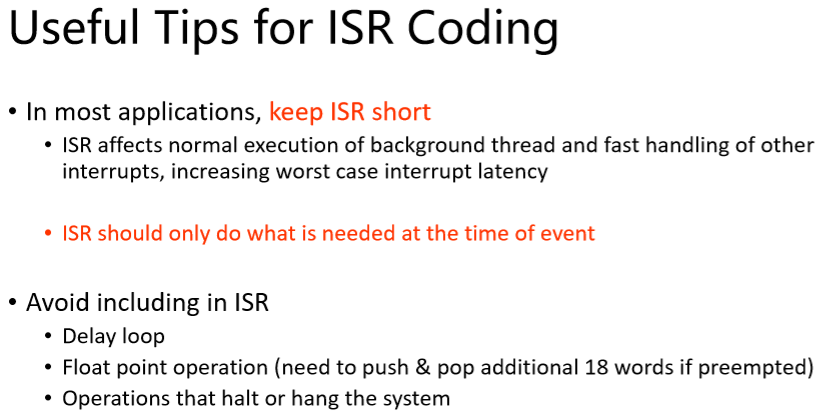
* Prioritizes and handles all interrupts
* • Preempt priority number for preemption
* • Sub-priority number for ordering interrupts with same preempt priority

**ISR as a Function**

• At interrupt, microcontroller stops to run ISR from a new address

* Similar to a ‘normal’ function call in C program
* Code ISR as a C function

• ‘Normal’ function vs. ISR

* ‘Normal’ function call is user planned (programmed)
* Interrupt is asynchronous

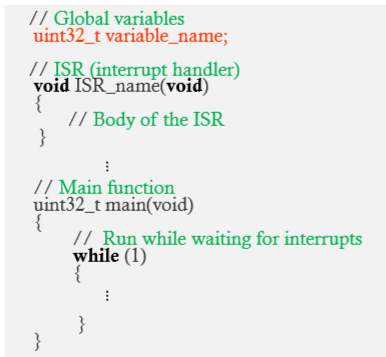
Occurrence time may be unpredictable

ISR = do not know the exact time to call it.

Cannot predict when it will be called

ISR = has not input or return

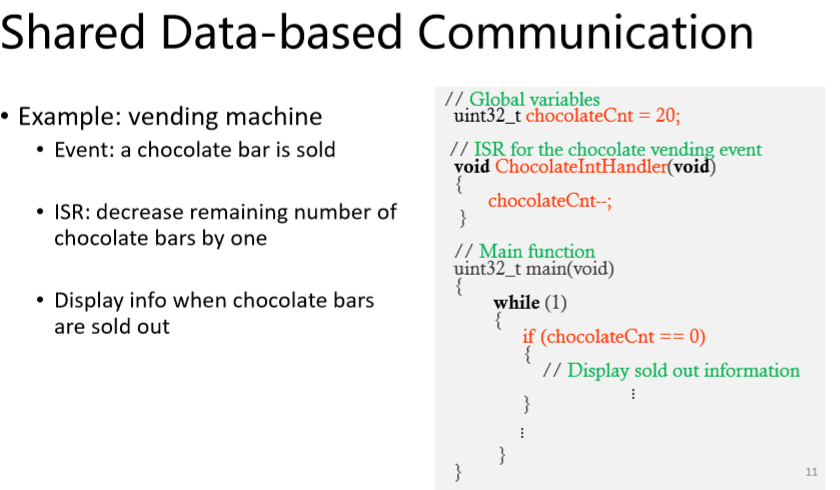
ISR = should be short (**do not use any delay loop** with ISR) – don’t not use any floating point operations. Use global variables

**Inter-thread communication**

• Inter-thread communication through global memory

* Global variables defined outside of all functions

• Global variables

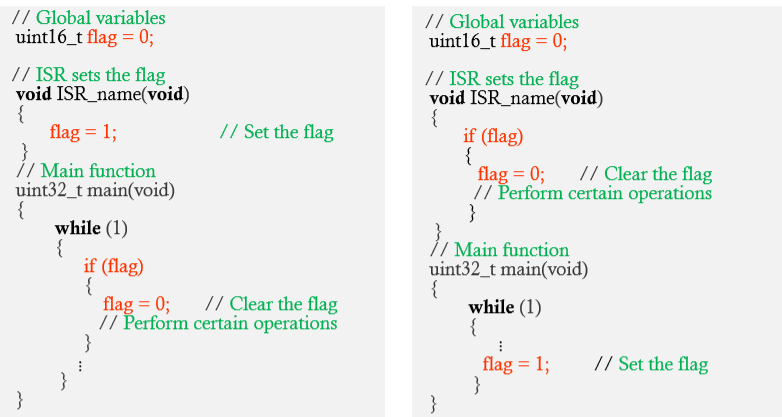
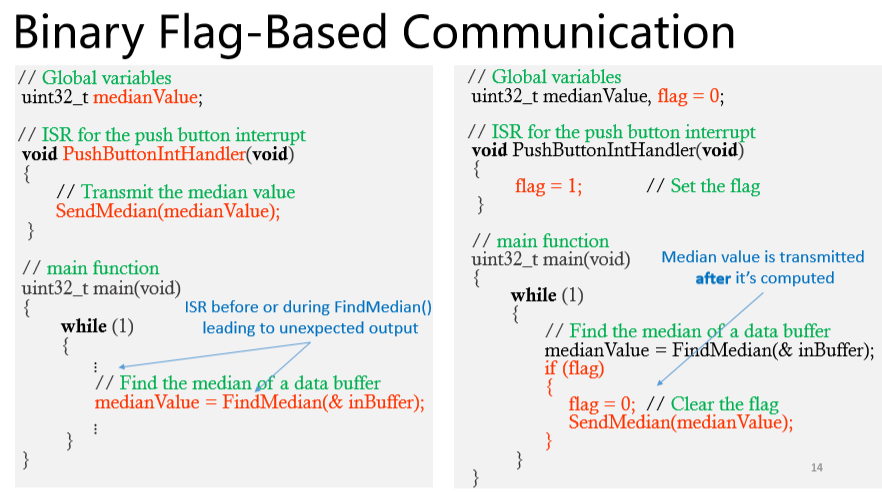
*  Data • Binary flag • Mailbox (binary flag + data) • Circular buffer …

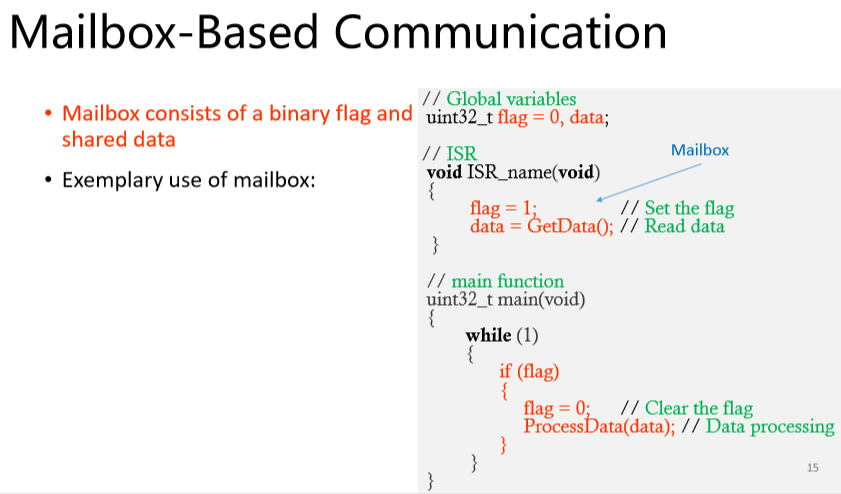
**Binary Flag-based Communication**

• Shared data-based communication is one way to synchronize threads

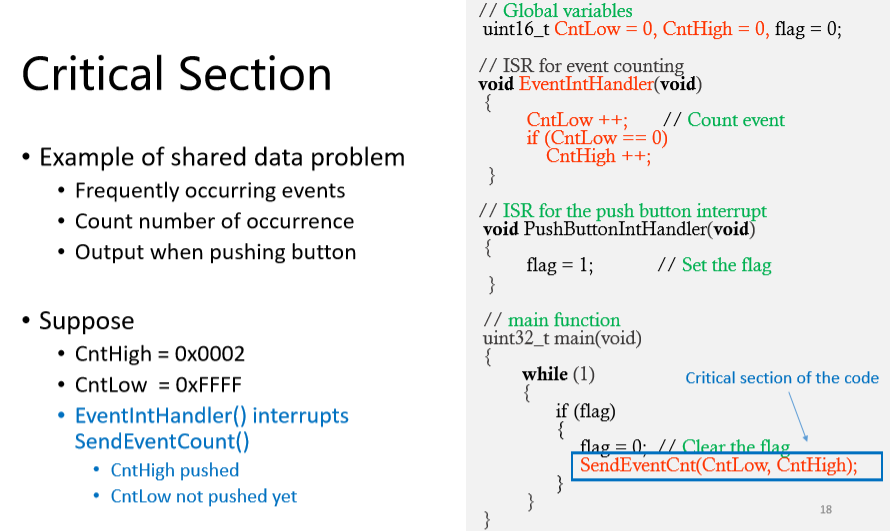
• Binary flag is another way

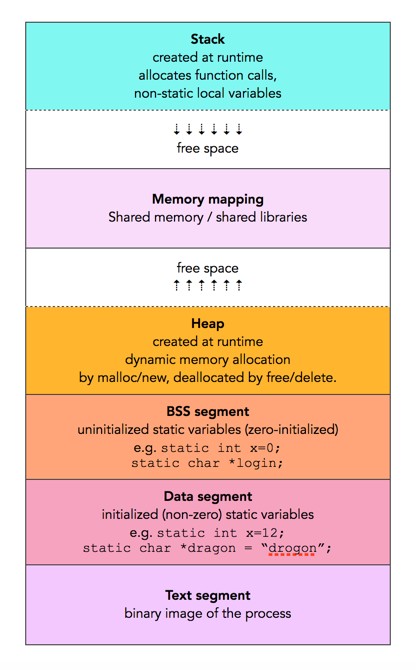
* Set flag for signaling permission to perform certain operations
* Remember to clear flag

• ISR sets flag vs. main() sets flag

CntLow = 0xFFFF and CntHigh = 0x0002, “1” is added to CntHigh because…

CntLow = 0xFFFF + 1 = 0x10000 and thus CntHigh adds 1 (the 5th bit), then …

CntHigh = 0x0003

**Global vs Static variables**

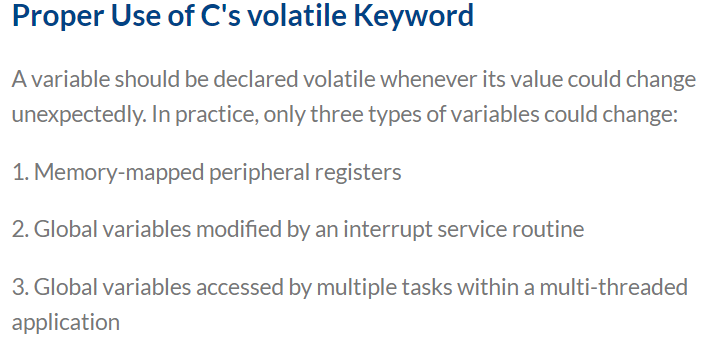
Global variables:

* Declared outside function
* Stored in **Data Segment** (Initialized)
* Accessed by other files using keyword **extern**
* Life = Until end of program

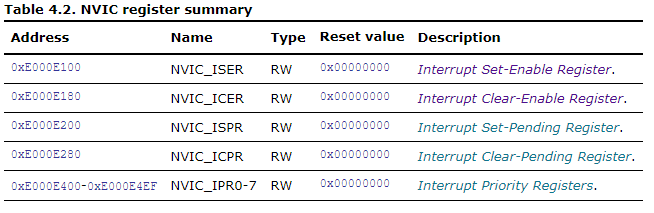
Static variables:

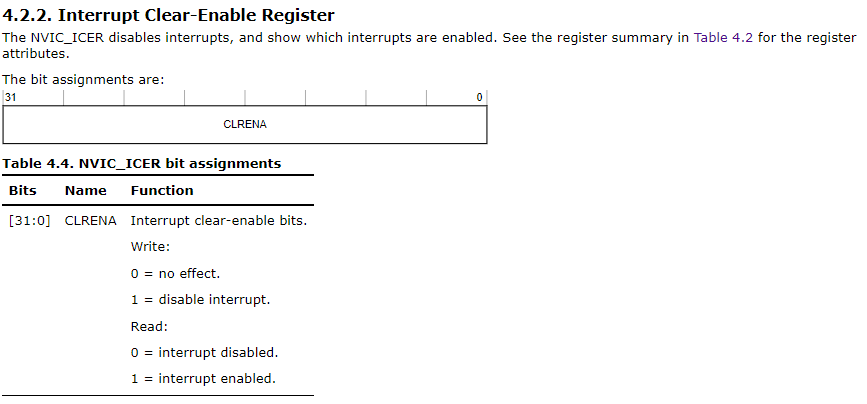
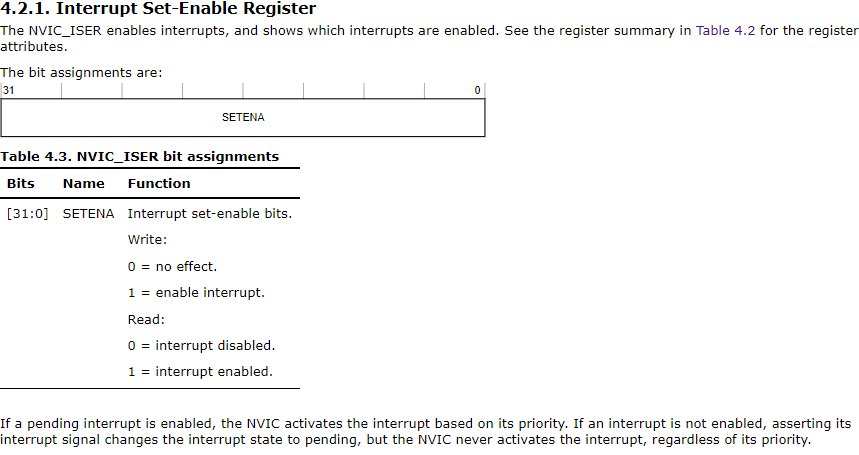
* Declared outside/inside function
* Initialized once
* Stored in **Data Segment** (Initialized)
* Outside function = Global scope of file (cannot be accessed in other files)
* Inside function = Global scope within function
* Life = Until end of program

**Volatile**

* Allows us to change a memory IO Port/Reg without optimisation (tells compile not to optimise code)
* Used with pointers mainly

**NVIC\_ISER** and **NVIC\_ICER** registers



<http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.dui0662b/Cihcajhj.html>