Interrupt Programming

Young Won Lim
10/31/21

Please send corrections (or suggestions) to youngwlim@hotmail.com.

This document was produced by using LibreOffice.

Based on

ARM System-on-Chip Architecture, 2nd ed, Steve Furber

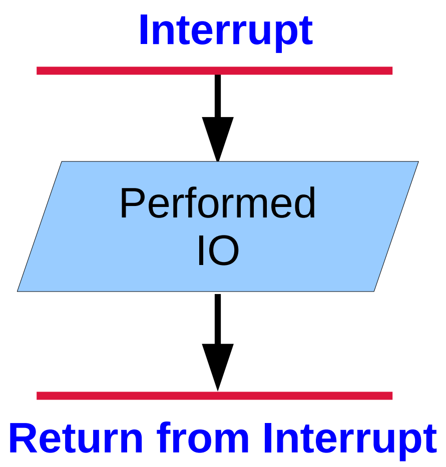
Introduction to ARM Cortex-M Microcontrollers
– Embedded Systems, Jonathan W. Valvano

Digital Design and Computer Architecture,
D. M. Harris and S. L. Harris

ARM assembler in Raspberry Pi
Roger Ferrer Ibáñez

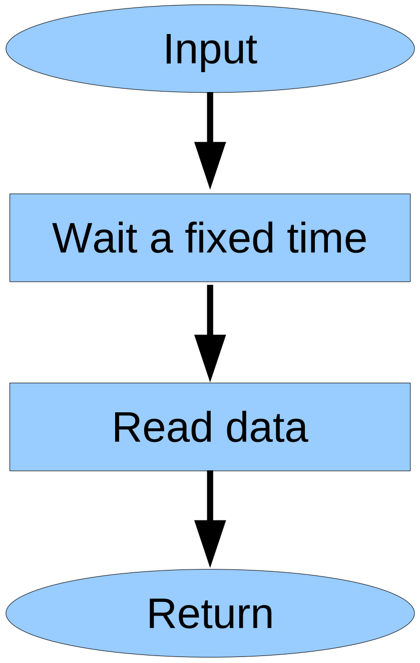
https://thinkingeek.com/arm-assembler-raspberry-pi/

Interrupt Service Routine



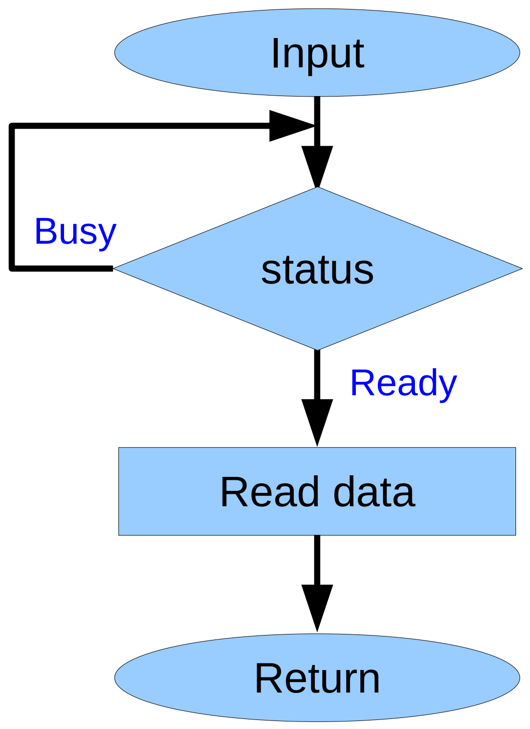
Introduction to ARM Cortex-M Microcontrollers Embedded Systems, Jonathan W. Valvano

Input - Blind Cycle



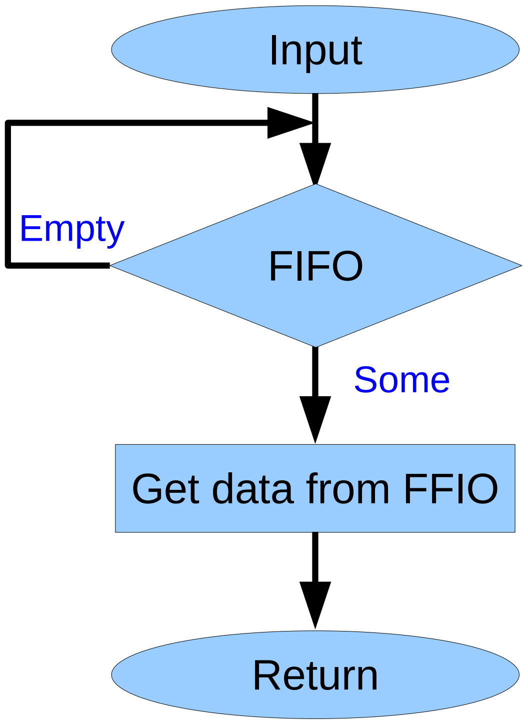
Introduction to ARM Cortex-M Microcontrollers Embedded Systems, Jonathan W. Valvano

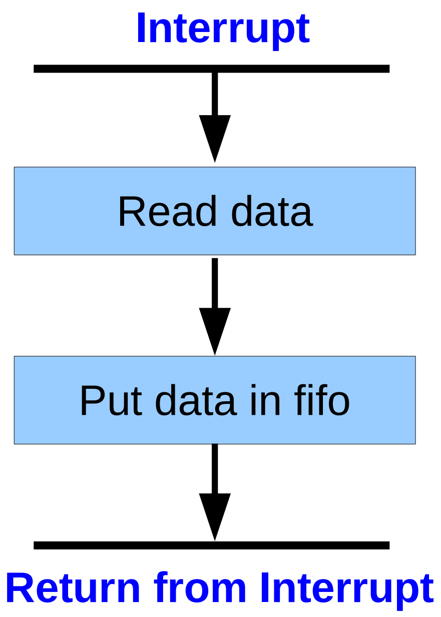
Input - Busy Wait Cycle



Introduction to ARM Cortex-M Microcontrollers Embedded Systems, Jonathan W. Valvano

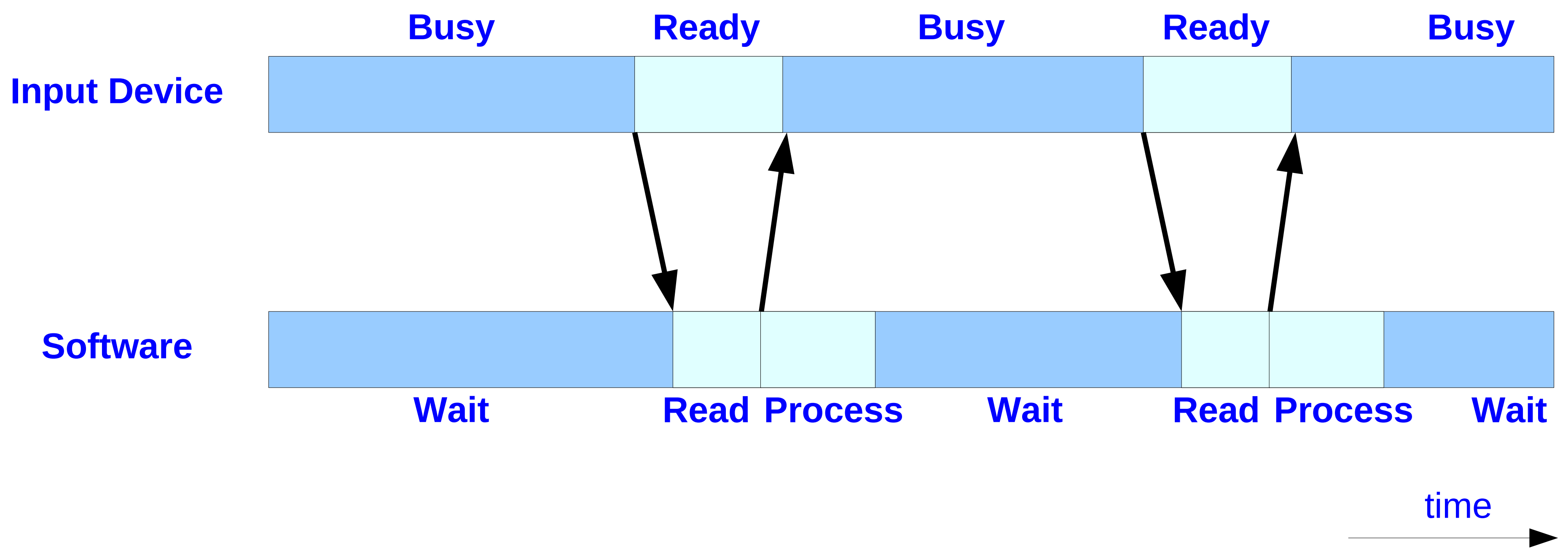
Input - Interrupt





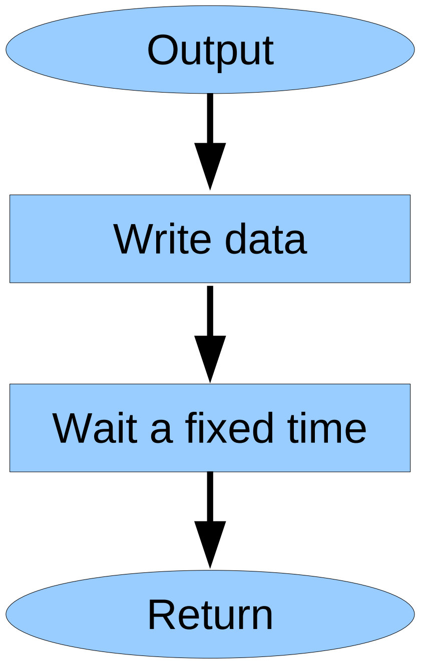
Introduction to ARM Cortex-M Microcontrollers Embedded Systems, Jonathan W. Valvano

IO Bound Input Interface



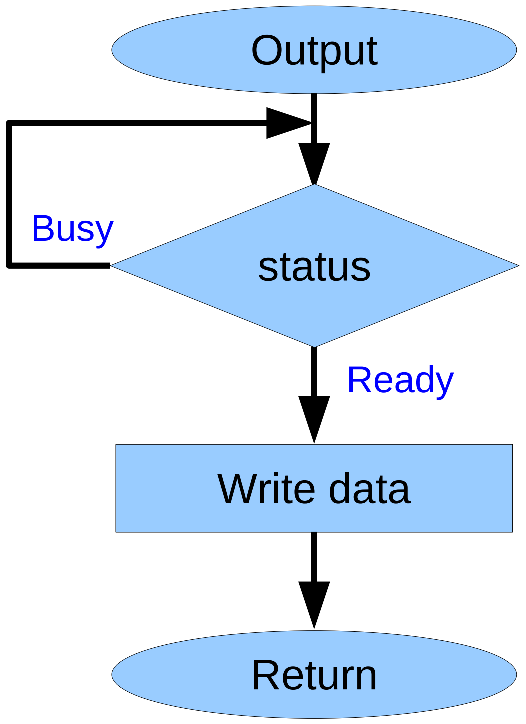
Introduction to ARM Cortex-M Microcontrollers Embedded Systems, Jonathan W. Valvano

Output - Blind Cycle



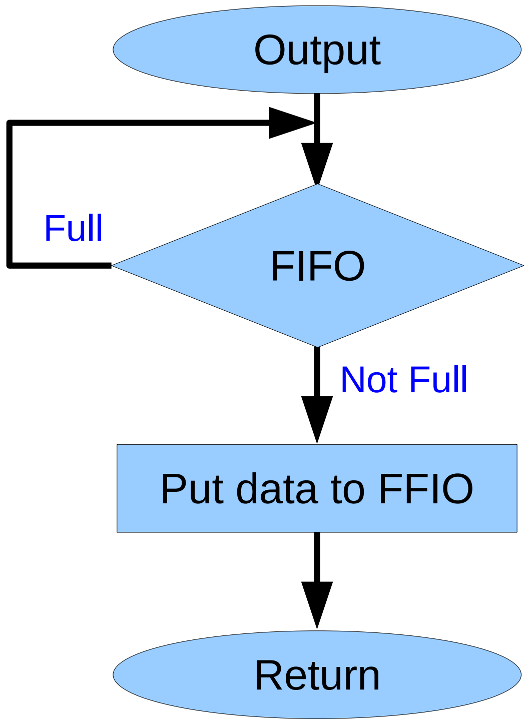
Introduction to ARM Cortex-M Microcontrollers Embedded Systems, Jonathan W. Valvano

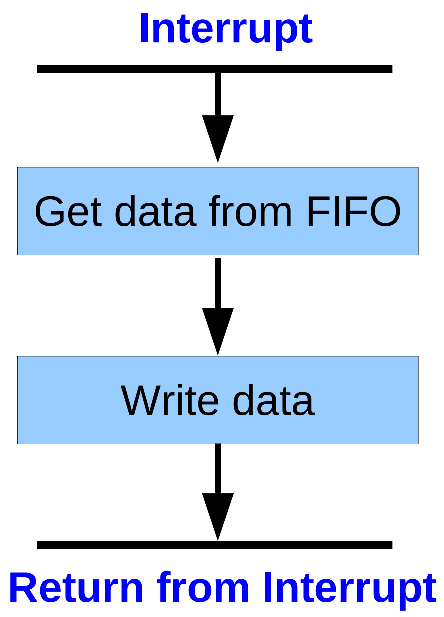
Output - Busy Wait Cycle



Introduction to ARM Cortex-M Microcontrollers Embedded Systems, Jonathan W. Valvano

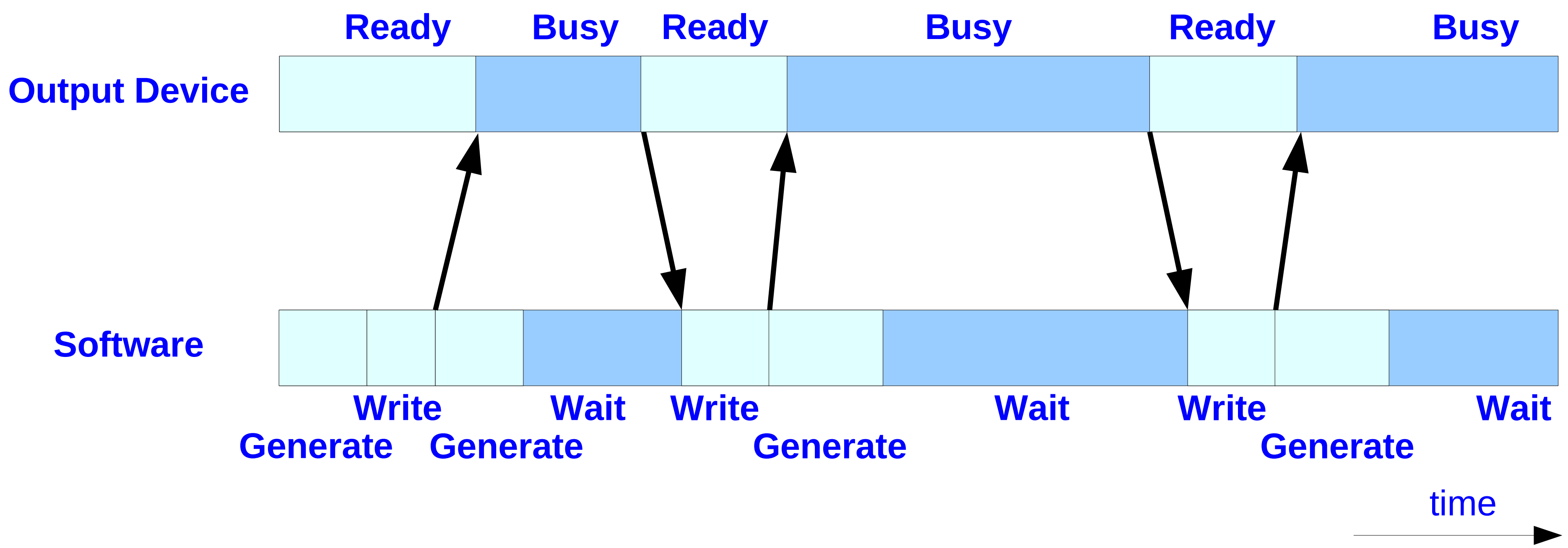
Output - Interrupt





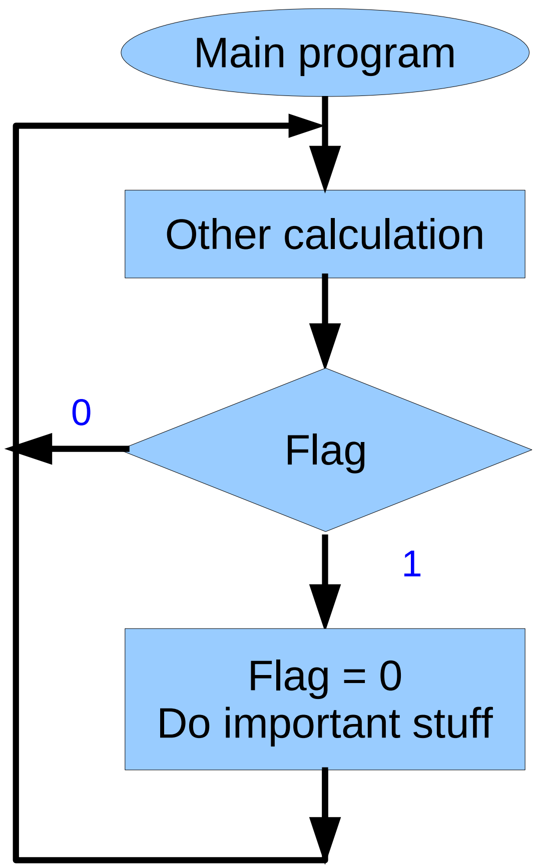
Introduction to ARM Cortex-M Microcontrollers Embedded Systems, Jonathan W. Valvano

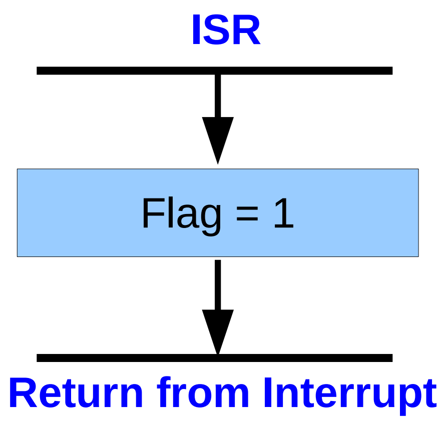
IO Bound Output Interface



Introduction to ARM Cortex-M Microcontrollers Embedded Systems, Jonathan W. Valvano

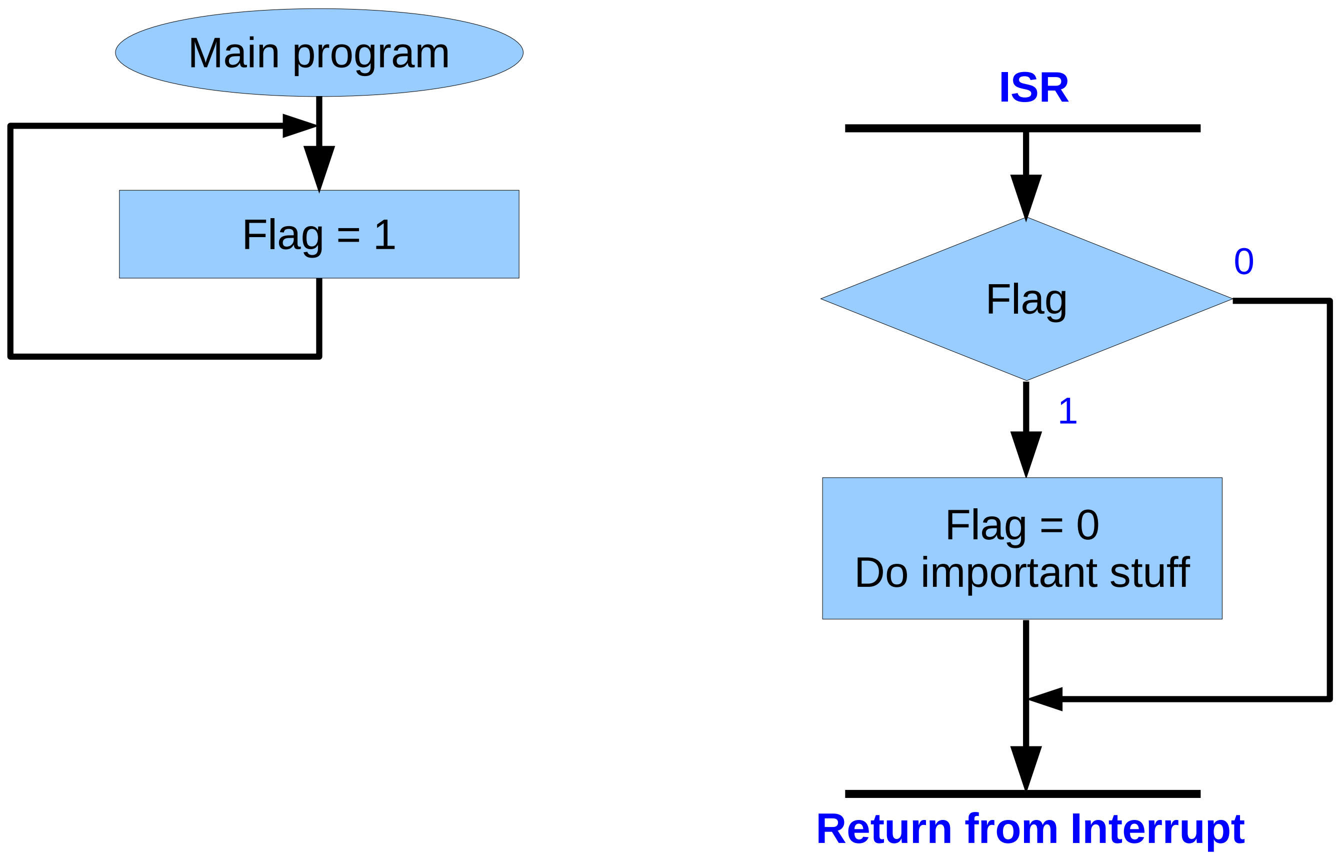
Semaphore to synchronize threads





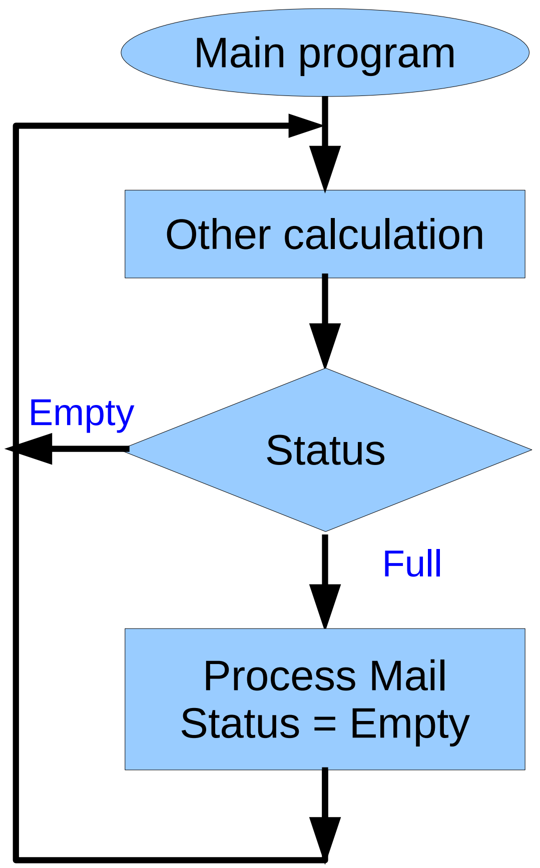
Introduction to ARM Cortex-M Microcontrollers Embedded Systems, Jonathan W. Valvano

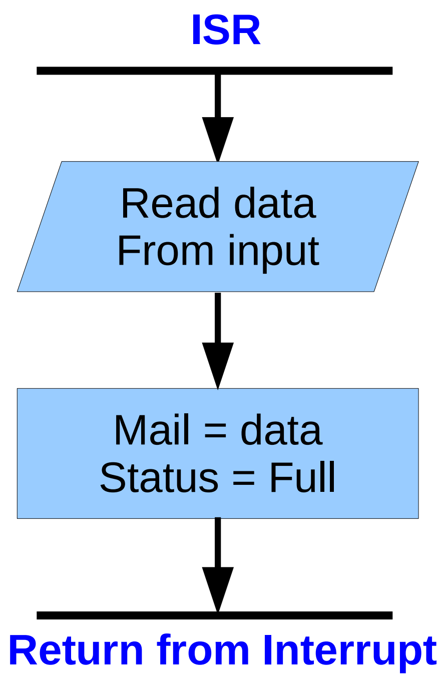
Semaphore to synchronize threads



Introduction to ARM Cortex-M Microcontrollers Embedded Systems, Jonathan W. Valvano

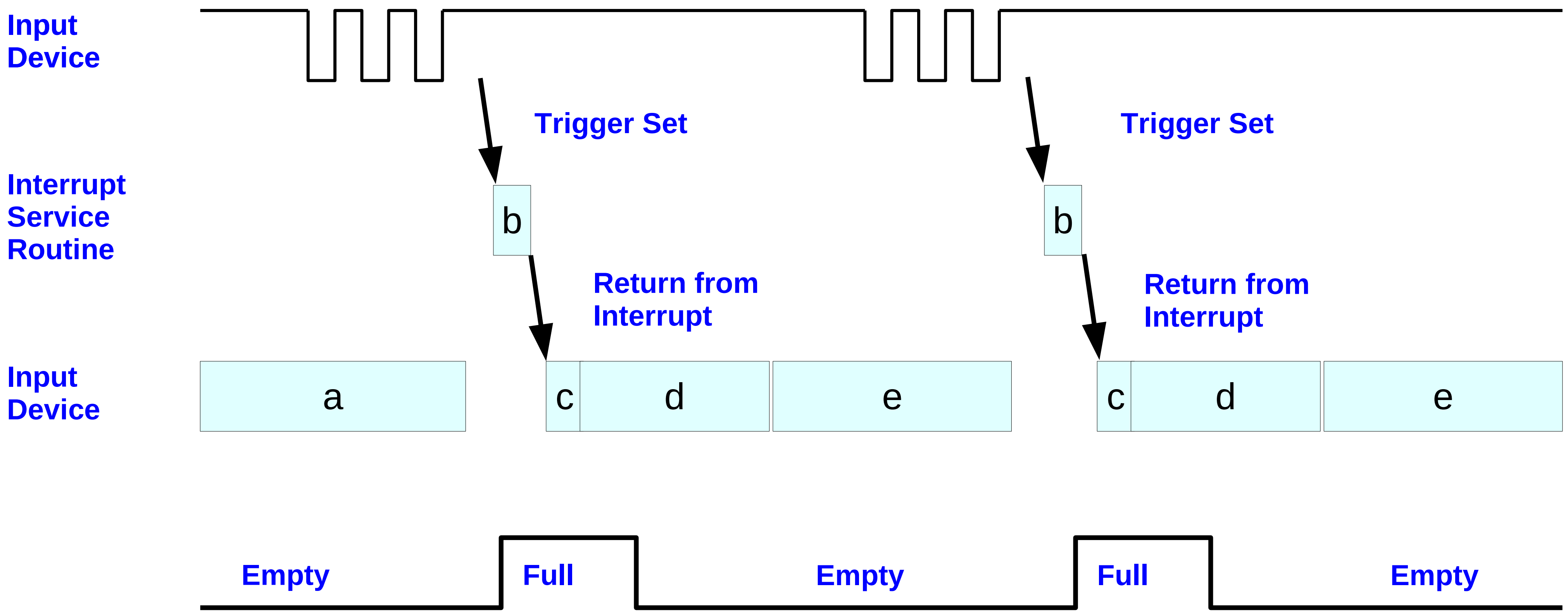
Mailbox





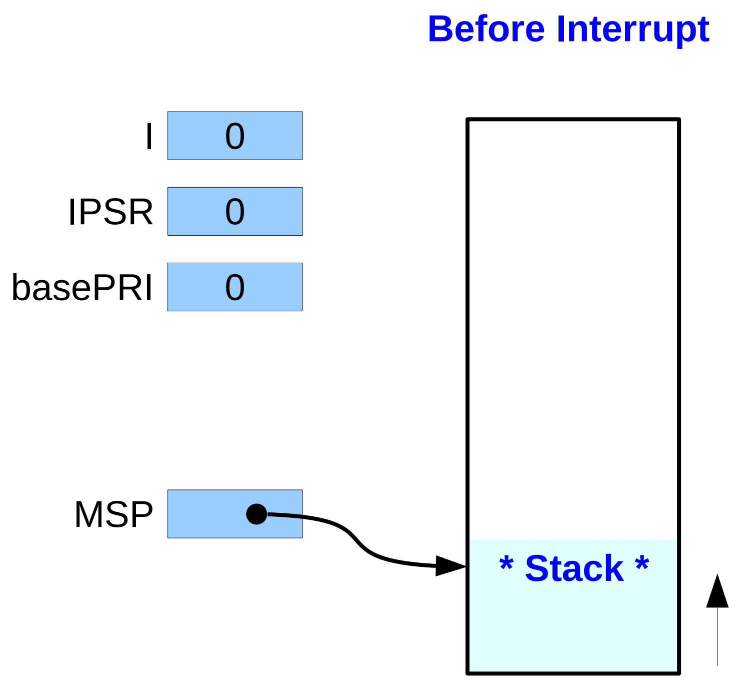
Introduction to ARM Cortex-M Microcontrollers Embedded Systems, Jonathan W. Valvano

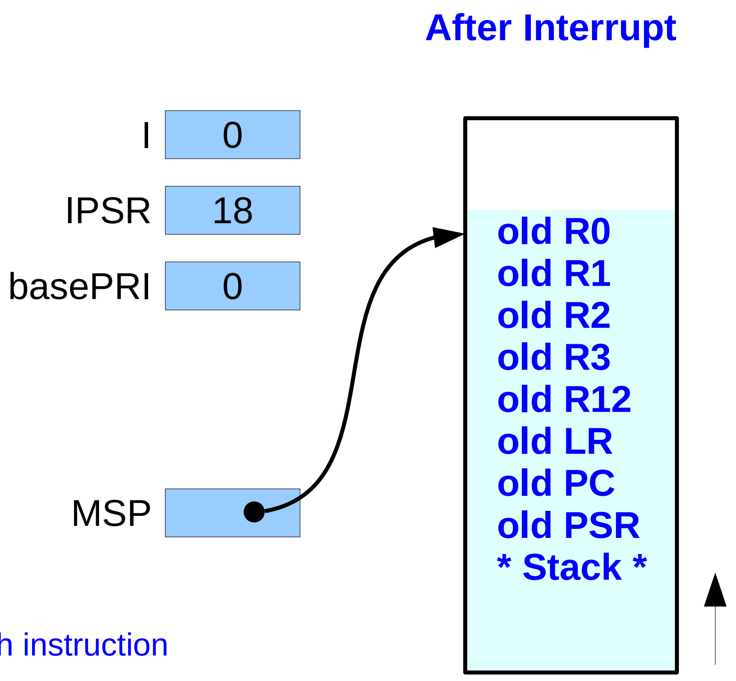
IO Bound Output Interface



Introduction to ARM Cortex-M Microcontrollers Embedded Systems, Jonathan W. Valvano

IO Bound Output Interface





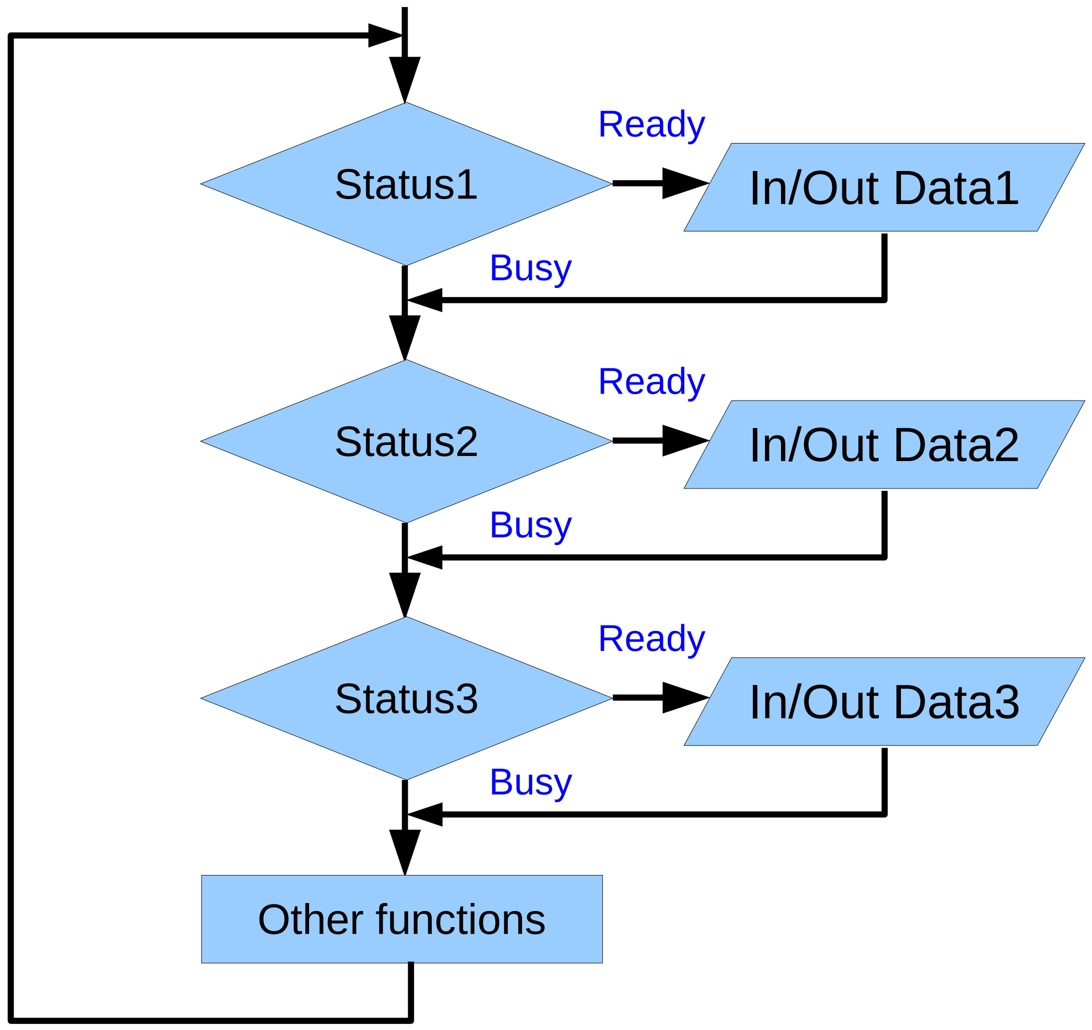
Context Switch Finish instruction

- Push registers
- PC = {0x00000048}
- Set IPSR = 18
- Set LR = 0xFFFFFFF9

Use MSP as stack pointer

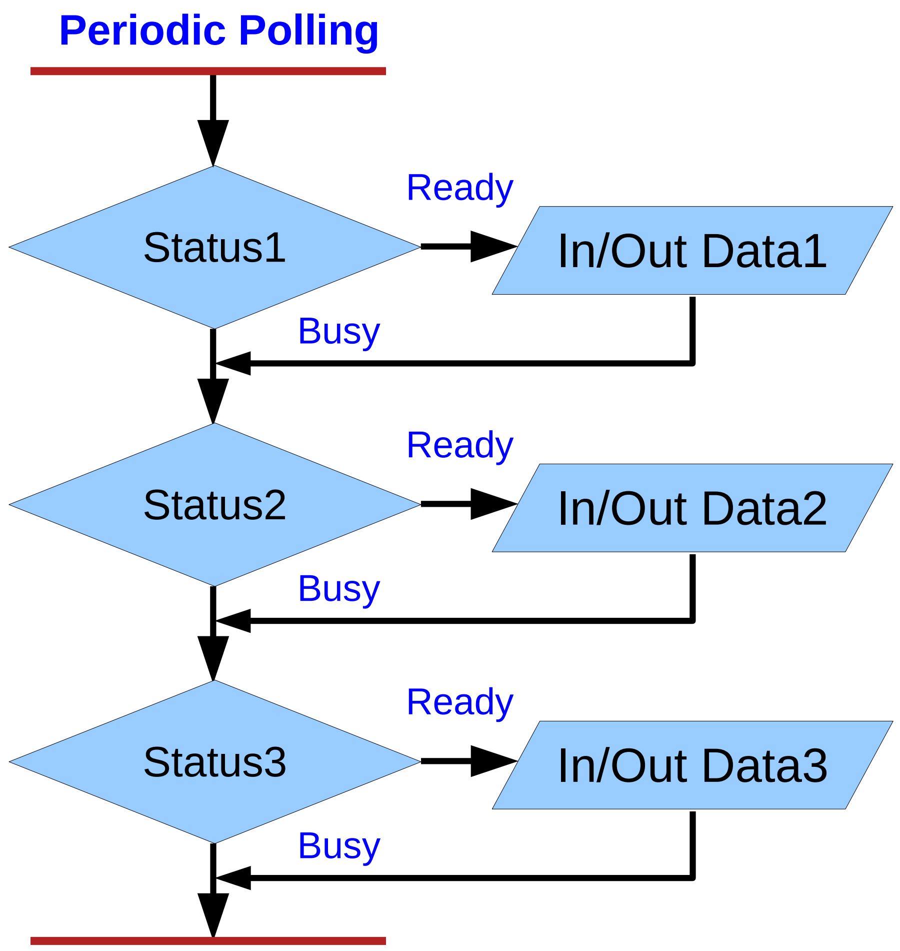
Introduction to ARM Cortex-M Microcontrollers Embedded Systems, Jonathan W. Valvano

Semaphore to synchronize threads



Introduction to ARM Cortex-M Microcontrollers Embedded Systems, Jonathan W. Valvano

Semaphore to synchronize threads



Introduction to ARM Cortex-M Microcontrollers Embedded Systems, Jonathan W. Valvano

Entering and exiting an exception handler

- Preserve the address of the next instruction.
- Copy CPSR to the appropriate SPSR
  one of the banked registers for each mode of operation.
- Force the CPSR mode bits to a value depending on the raised exception.
- Force the PC to fetch the next instruction from the exception vector table.
- Now the handler is running in the mode associated with the raised exception.
- When handler is done, the CPSR is restored from the saved SPSR.
- PC is updated with the value of (LR - offset) and
  the offset value depends on the type of the exception.

http://classweb.ece.umd.edu/enee447.S2019/ARM-Documentation/ARM-Interrupts-3.pdf

(9) Entering and returning exception handler

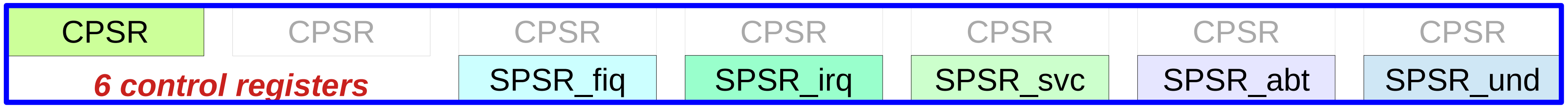
Entering exception handler

1. Save the address of the next instruction in the appropriate Link Register **LR**.
2. Copy **CPSR** to the **SPSR** of new mode.
3. Change the mode by modifying bits in **CPSR**.
4. Fetch next instruction from the vector table.

Exception	Returning Address
Reset	None
Data Abort	**LR** - 8
FIQ, IRQ, prefetch Abort	**LR** - 4
SWI, Undefined Instruction	**LR**

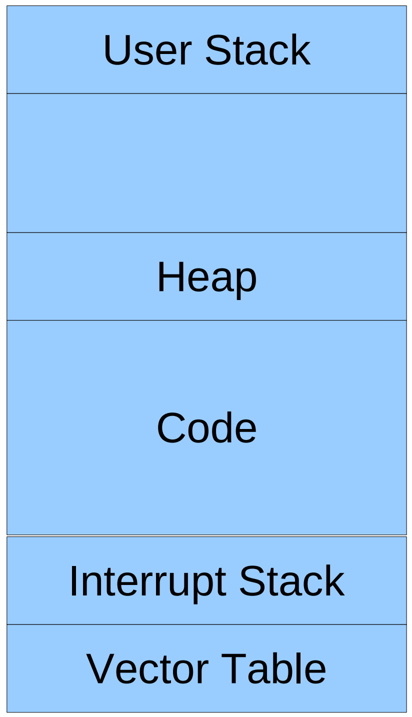
Leaving exception handler

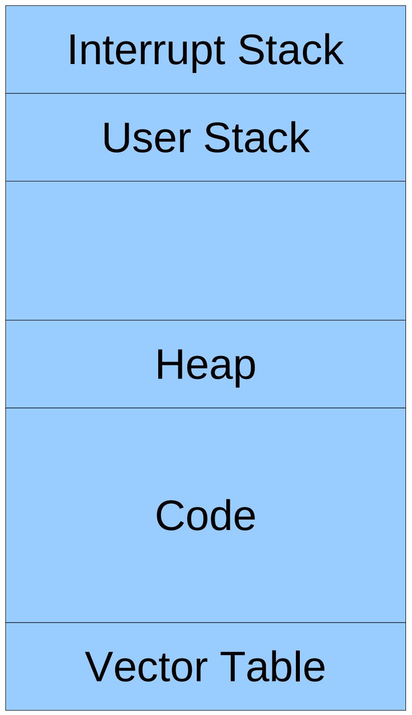
1. Move (**LR** - offset) to the **PC**.
2. Copy **SPSR** back to **CPSR**, this will automatically changes the mode back to the previous one.
3. Clear the interrupt disable flags (if they were set)



https://www.ic.unicamp.br/~celio/mc404-2013/arm-manuals/ARM_exception_slides.pdf

Interrupt stack





http://classweb.ece.umd.edu/enee447.S2019/ARM-Documentation/ARM-Interrupts-3.pdf

Interrupt Handling

1. Non-nested interrupt handling

- Handle and service individual interrupts sequentially.
- High interrupt latency.
- Relatively easy to implement and debug.
- Not suitable for complex embedded systems.

2. Nested interrupt handling

- Handle multiple interrupts without a priority assignment.
- Medium or high interrupt latency.
- Enable interrupts before the servicing of an individual interrupt is complete.
- No prioritization, so low priority interrupts can block higher priority interrupts.

3. Prioritized interrupt handling

- Handle multiple interrupts with a priority assignment mechanism.
- Low interrupt latency.
- Deterministic interrupt latency.
- Time taken to get to a low priority ISR is the same as for high priority ISR.

http://classweb.ece.umd.edu/enee447.S2019/ARM-Documentation/ARM-Interrupts-3.pdf

1. Non-nested interrupt handling

the simplest interrupt handler.

Interrupts are disabled until control is returned back to the interrupted task.

only one interrupt can be served at a time
not suitable for complex embedded systems
which most probably have more than one interrupt source
and require concurrent handling.

the steps taken to handle an Interrupt:

Handle and service individual interrupts sequentially.

- high interrupt latency.
- relatively easy to implement and debug.
- not suitable for complex embedded systems.

http://classweb.ece.umd.edu/enee447.S2019/ARM-Documentation/ARM-Interrupts-3.pdf

1. Non-nested interrupt handling

Initially interrupts are disabled,
when IRQ exception is raised and
the ARM processor disables
further IRQ exceptions from occurring.

The mode is changed to the new mode
depending on the raised exception.

The register CPSR is copied to the SPSR of the new mode.

Then the PC is set to the correct entry in the vector table
and the instruction there will direct the PC to the appropriate handler.

http://classweb.ece.umd.edu/enee447.S2019/ARM-Documentation/ARM-Interrupts-3.pdf

1. Non-nested interrupt handling

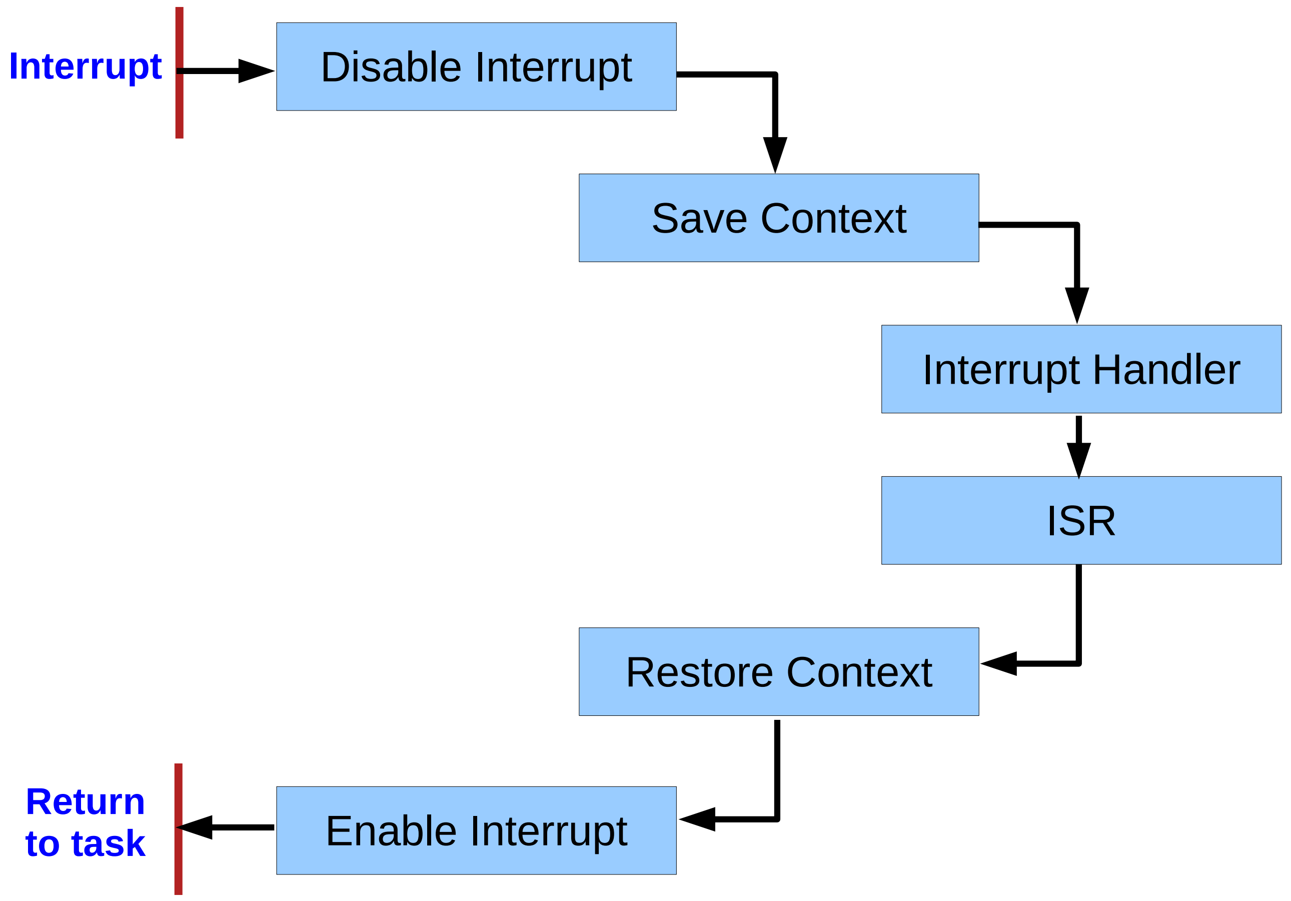
then the context of the current task is saved a subset
of the current mode non banked register.

then the interrupt handler executes some code
to identify the interrupt source and
decide which ISR will be called.
Then the appropriate ISR is called.

finally the context of the interrupted task is restored,
interrupts are enabled again
and the control is returned to the interrupted task.

http://classweb.ece.umd.edu/enee447.S2019/ARM-Documentation/ARM-Interrupts-3.pdf

1. Non-nested interrupt handling



http://classweb.ece.umd.edu/enee447.S2019/ARM-Documentation/ARM-Interrupts-3.pdf

2. Nested interrupt handling

In the nested interrupt handling scheme handling
more than one interrupt at a time is possible.

this is achieved by re-enabling interrupts
before the handler has fully served the current interrupt.

This feature increases the complexity of the system
but decreases the latency.

The scheme should be designed carefully
to protect the context saving and restoration
from being interrupted.

should balance between efficiency and safety
by using defensive coding style
that assumes problems will occur.

http://classweb.ece.umd.edu/enee447.S2019/ARM-Documentation/ARM-Interrupts-3.pdf

2. Nested interrupt handling

handle multiple interrupts without a priority assignment.

- Medium or high interrupt latency.
- enable interrupts before the servicing of an individual interrupt is complete.
- no prioritization, so low priority interrupts can block higher priority interrupts.

The goal of nested handling is

- to respond to interrupts quickly and
- to execute periodic tasks without any delays.

Re-enabling interrupts requires
switching out of the IRQ mode to user mode
to protect link register from being corrupted.

http://classweb.ece.umd.edu/enee447.S2019/ARM-Documentation/ARM-Interrupts-3.pdf

2. Nested interrupt handling

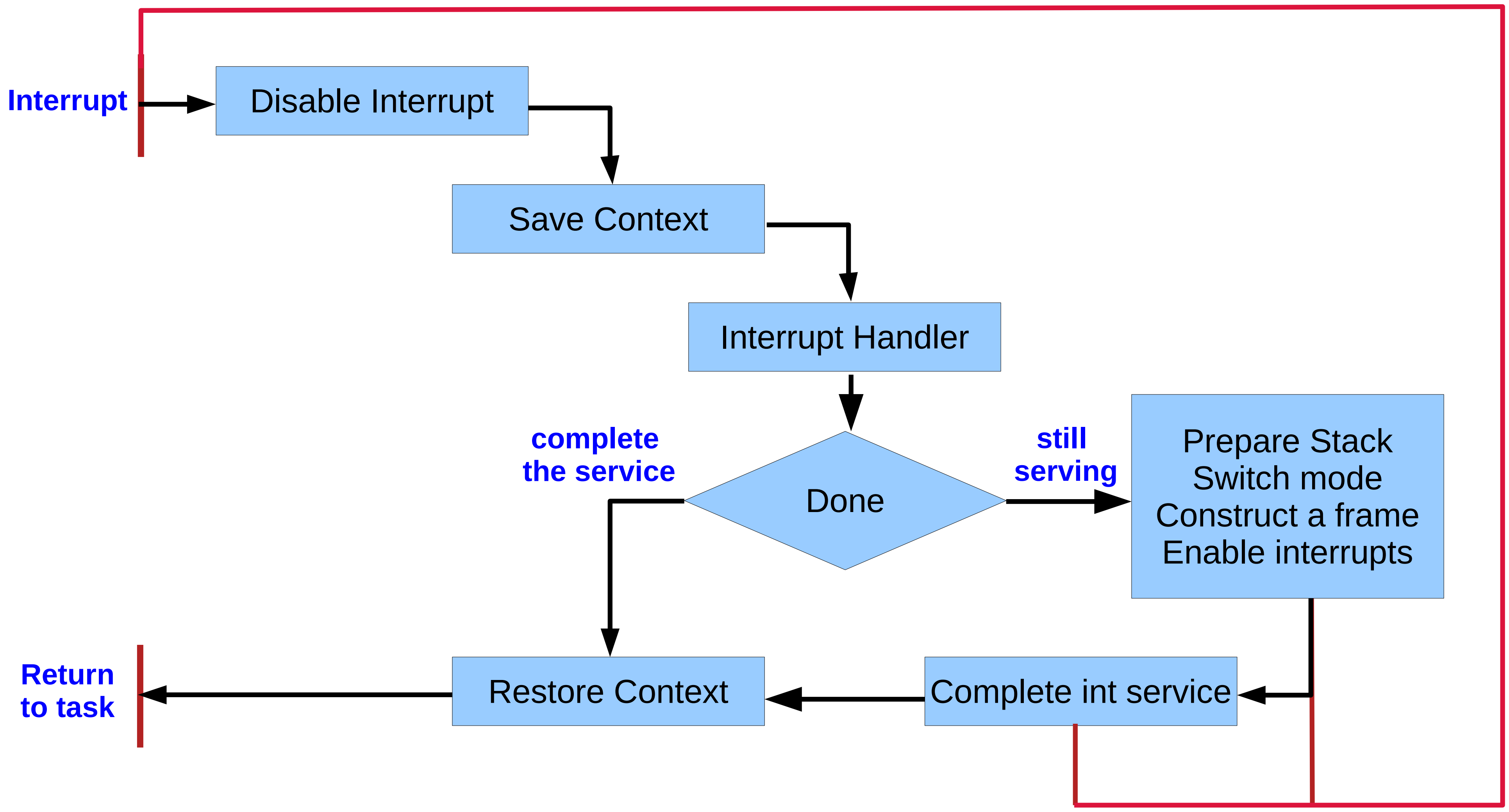
Also performing context switch requires
emptying the IRQ stack
because the handler will not perform switching
if there is data on the IRQ stack,
so all registers saved on the IRQ stack
have to be transferred to task stack.

The part of the task stack used in this process
is called stack frame.

The main disadvantage of this interrupt handling scheme is that
it doesn’t differ between interrupts by priorities,
so lower priority interrupt can block higher priority interrupts.

http://classweb.ece.umd.edu/enee447.S2019/ARM-Documentation/ARM-Interrupts-3.pdf

2. Nested interrupt handling



http://classweb.ece.umd.edu/enee447.S2019/ARM-Documentation/ARM-Interrupts-3.pdf

3. Prioritized interrupt handling

In the prioritized interrupt handling scheme
the handler will associate a priority level
with a particular interrupt source.

A higher priority interrupt will take precedence over a lower priority interrupt.

Handling prioritization can be done by means of software or hardware.

In case of hardware prioritization the handler is simpler to design
because the interrupt controller will give the interrupt signal
of the highest priority interrupt requiring service.

But on the other side the system needs more initialization code at start-up
since priority level tables have to be constructed before the system being switched on.

http://classweb.ece.umd.edu/enee447.S2019/ARM-Documentation/ARM-Interrupts-3.pdf

3. Prioritized interrupt handling

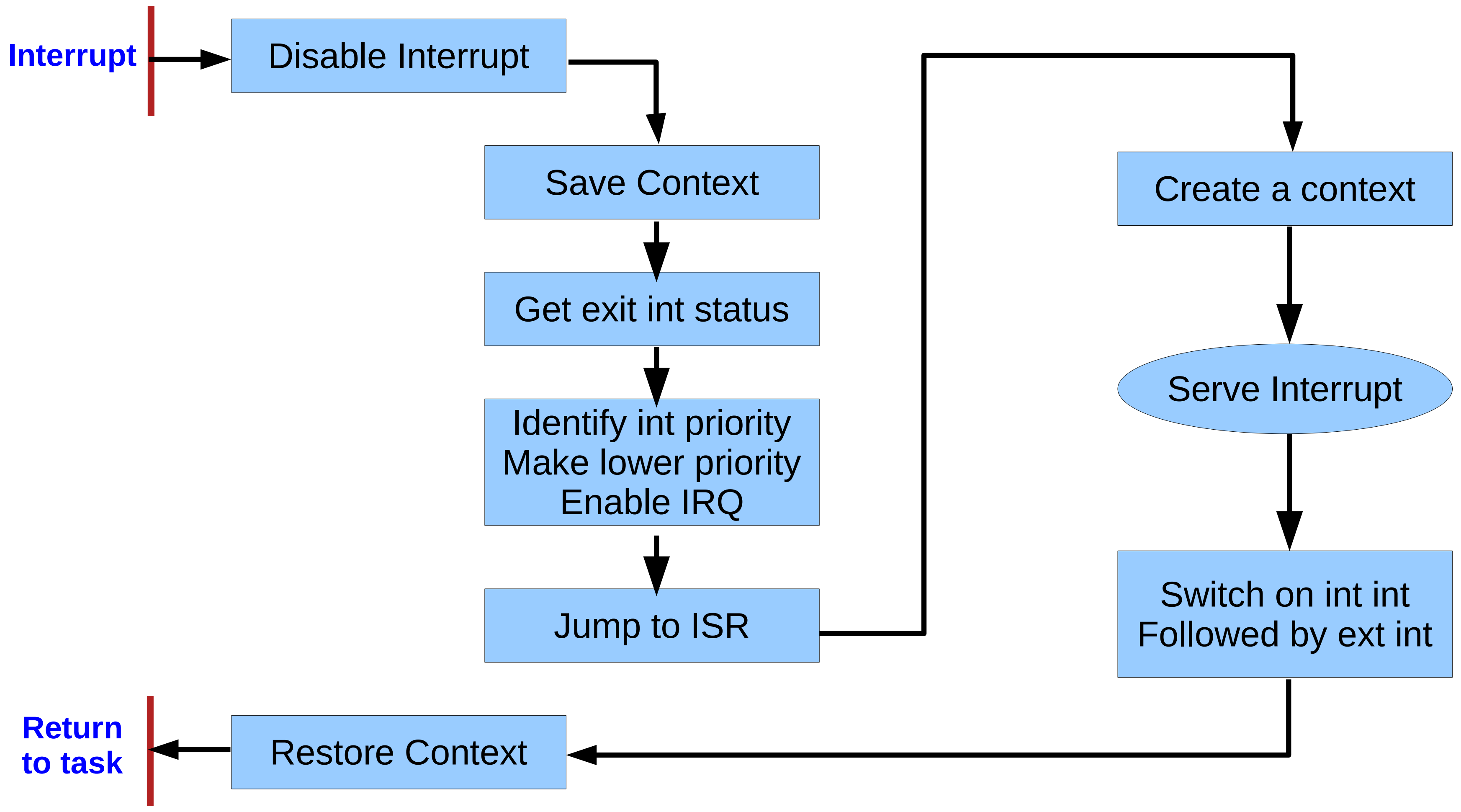
Handle multiple interrupts with a priority assignment mechanism.

- Low interrupt latency.
- Deterministic interrupt latency.
- Time taken to get to a low priority ISR is the same as for high priority ISR.

When an interrupt signal is raised,
a fixed amount of comparisons
with the available set of priority levels is done,
so the interrupt latency is deterministic
but at the same point this could be considered a disadvantage
because both high and low priority interrupts take the same amount of time.

http://classweb.ece.umd.edu/enee447.S2019/ARM-Documentation/ARM-Interrupts-3.pdf

3. Prioritized interrupt handling



http://classweb.ece.umd.edu/enee447.S2019/ARM-Documentation/ARM-Interrupts-3.pdf

References

[1] http://wiki.osdev.org/ARM_RaspberryPi_Tutorial_C
[2] http://blog.bobuhiro11.net/2014/01-13-baremetal.html
[3] http://www.valvers.com/open-software/raspberry-pi/
[4] https://www.cl.cam.ac.uk/projects/raspberrypi/tutorials/os/downloads.html