NTURACING

MCU Programming

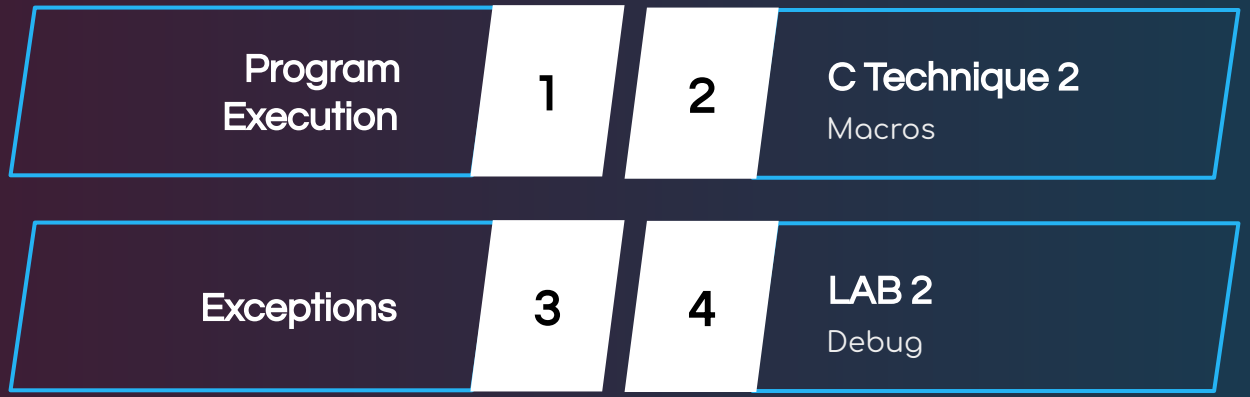
W2: Program Execution

by 羅紀翔

2024-25 FSAE Season

Agenda

NTURACING



NTURACING

Program Execution

References: CH4, 7, 8, *The Definitive Guide*; Ch 20, 22, *Mastering STM32*; Stack Memory, Code Inside Out: https://codeinsideout.com/blog/stm32/stack-memory/

Execution

NTURACING

CPU Instruction

- Essential instructions:
  - **I/O**: Move data from/to memory
  - **Arithmetic** : Addition, subtraction, …
  - **Logical** : AND, OR, NOT, …
  - **Control flow** : Jump, branch, …
- Instruction can be written by assembly
- Basic syntax: `[label] mnemonic [operands]`
- Checkout Compiler Explore (https://godbolt.org/) to see what assembly does the source code compiles to

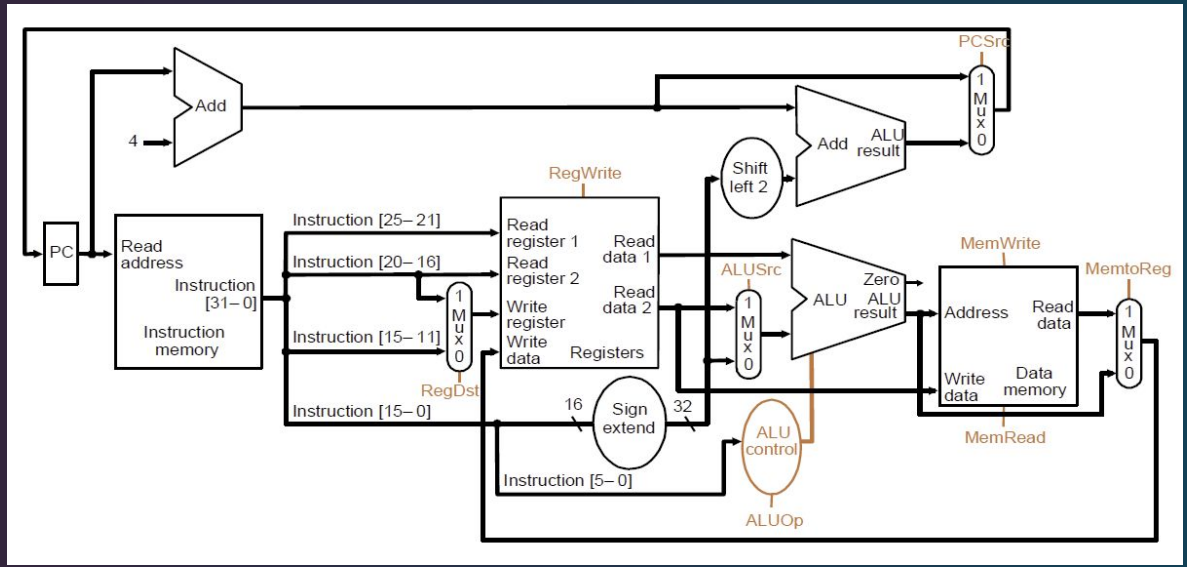
```
.LC0:
        .ascii  "hello world\000"
main:
        push    {fp, lr}
        add     fp, sp, #4
        ldr     r0, .L3
        bl      puts
        mov     r3, #0
        mov     r0, r3
        sub     sp, fp, #4
        pop     {fp, lr}
        bx      lr
.L3:
        .word   .LC0
```

Execution

NTURACING

Executing Instruction

- Single cycle MIPS CPU
- Fetch, decode, execute cycle
- Take **Computer Architecture** to learn more



Execution

NTURACING

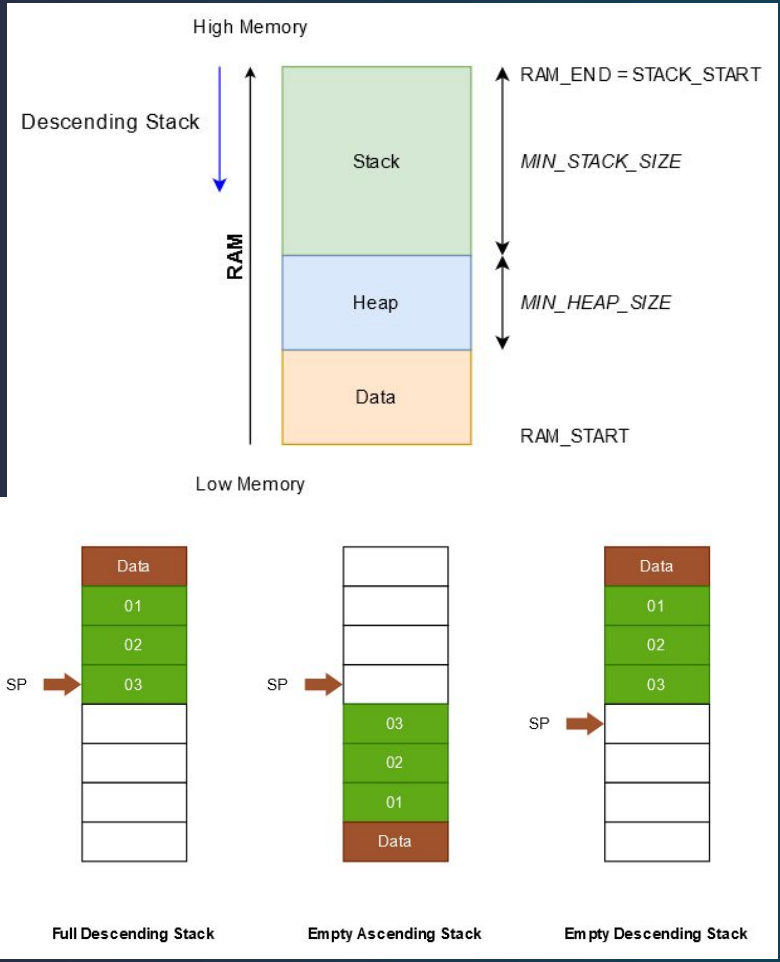
Features for High-Level Languages

- “A programming language with strong abstraction form the details of the computer.” (form Wikipedia: High-level programming language)
  - C is just a thin layer of abstraction over assembly
- Automatic memory management: stack, heap (only stack in C)
- Function call
- Runtime
  - Intepretation, VM
  - Exception
  - Library
  - C requires data and zero initialization

Execution

Stack

- Stores intermediate values in execution
- Stores function arguments
- Saves execution contex when procedure (function) call or during ISR
- Cortex-M is full descending stack

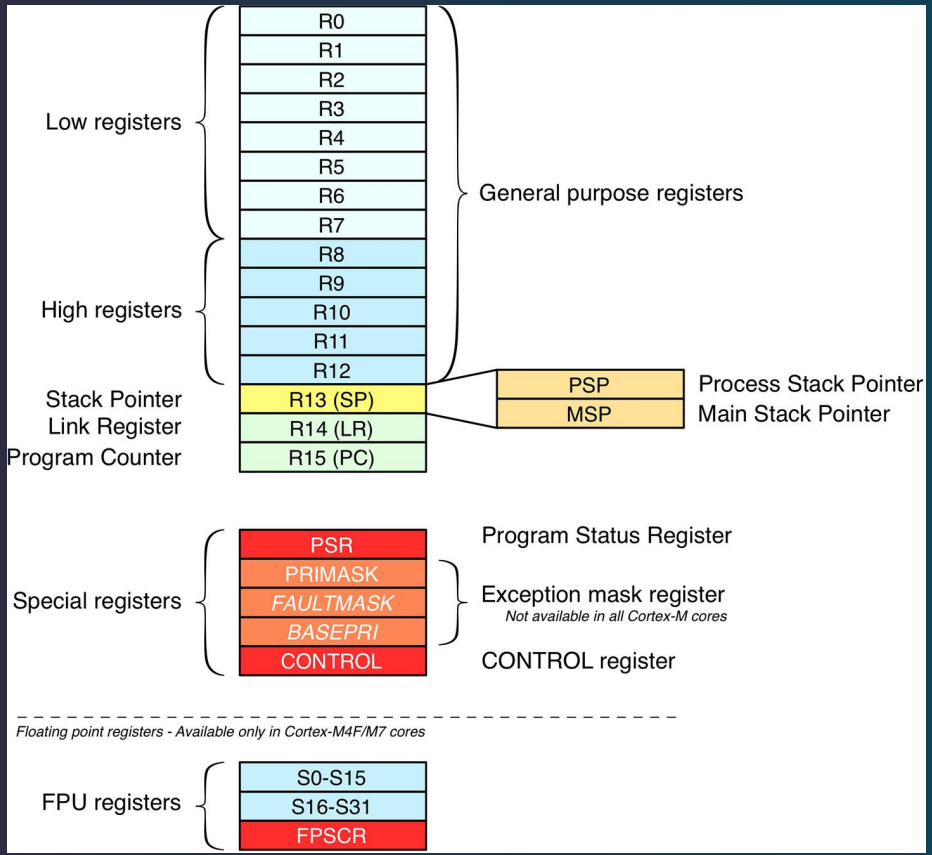


Execution

NTURACING

Recall: Registers

- Frame pointer (FP): R7, if used (typically), stores the start of the function frame
- Stack pointer (SP): the top of the stack
  - Shadowed (Banked), usually used by embedded OS to differentiate privileged and unprivileged stacks
- Link register (LR): return address of functions
- Program counter (PC): location of the next instruction



Execution

NTURACING

Function Call

- Calling convection, aka Application binary interface (**ABI**) (c.f. API)
- ARM Architecture Procedure Call Standard (AAPCS)
- **Caller-saved** : R0~R3, R12, can be freely used by functions
  - R12 is caller-saved since it's used as scratch register that the compiler can use it to store temporary values and does not need to saved it to the stack
- **Callee-saved** : other registers, must be saved (typically in the stack) before used
- Arguments: R0~R3
- Return value: R0 (and R1 if return value is 64-bit)
- Ref. Calling convection, Wiki: https://en.wikipedia.org/wiki/Calling_convention#ARM_(A32)

Execution

NTURACING

Function Prologue and Epilogue

- Procedures to conform to calling convention
- Prologue
  - Push LR and FP
  - Push callee-saved registers
  - Allocate stack for local variables
- Epilogue
  - Deallocate stack
  - Pop callee-saved registers
  - Pop FP and LR
- Overhead of function calls!
  - Also heavily optimized in moden CPU and compiler

Execution

```c
int add(int a1, int a2, int a3, int a4, int a5) {
    return a1 + a2 + a3 + a4 + a5;
}
int five() {
    return add(1, 1, 1, 1, 1);
}
```

```
five:
        push    {fp, lr}
        add     fp, sp, #4
        sub     sp, sp, #8

        mov     r3, #1
        str     r3, [sp]        ; passing a5 via stack
        mov     r3, #1
        mov     r2, #1
        mov     r1, #1
        mov     r0, #1
        bl      add
        mov     r3, r0

        mov     r0, r3
        sub     sp, fp, #4
        pop     {fp, lr}
        bx      lr
```

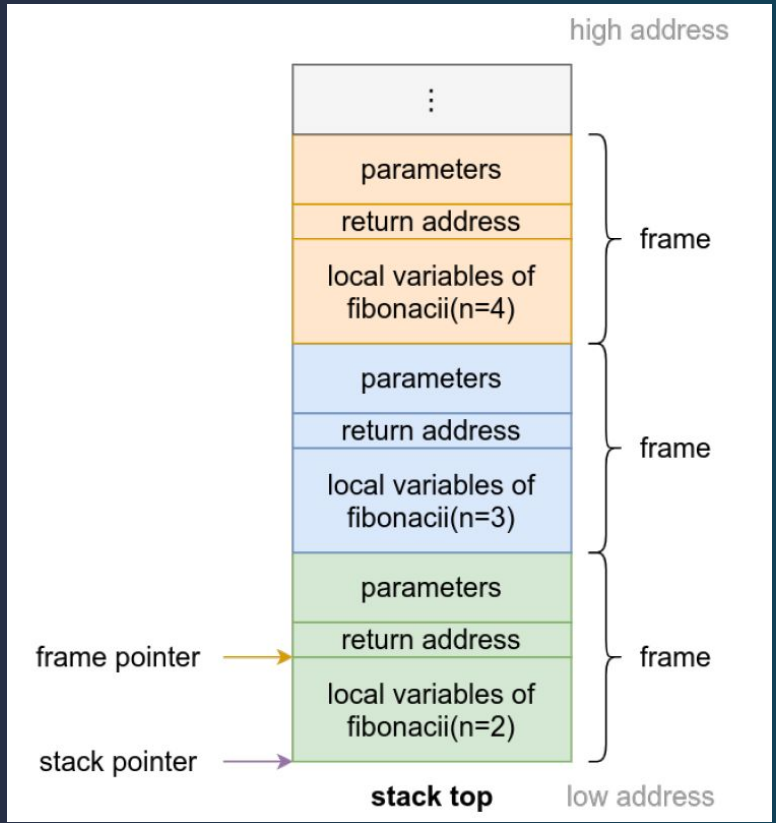
```
add:
        push    {fp}            ; push fp
        add     fp, sp, #0      ; set fp to the location of
                                ; previous fp
        sub     sp, sp, #20     ; allocate 20 bytes of stack
        str     r0, [fp, #-8]   ; save used registers to
stack
        str     r1, [fp, #-12]
        str     r2, [fp, #-16]
        str     r3, [fp, #-20]

        ldr     r2, [fp, #-8]   ; load a1-a4 from stack
        ldr     r3, [fp, #-12]
        add     r2, r2, r3
        ldr     r3, [fp, #-16]
        add     r2, r2, r3
        ldr     r3, [fp, #-20]
        add     r2, r2, r3
        ldr     r3, [fp, #4]    ; load a5 from stack
        add     r3, r2, r3

        mov     r0, r3
        add     sp, fp, #0      ; restore original sp form fp
        pop     {fp}
        bx      lr              ; return
```

Execution

NTURACING

Function Frame

- Stack frame saves the currently running function
- Infinite recursion will cause stack overflow
  - You can also see the overhead of recursion
- FP in the stack can be used to debug function call stack
- FP is optional as compiler knows how many stack the function requires, could be optimized via **frame pointer omission** , however, function call stack can't be debugged in this case



Execution

NTURACING

Bootup Sequence - Hardware

- Reset: Set all CPU and peripherals registers to predefined values
- After reset: Cortex-M loads SP and PC from 0x00000000 and start execution
- STM32 uses hardware remap to map 0x00000000 to different addresses depending on boot select pins:

Boot mode selection pins		Boot mode	Aliasing
BOOT1	BOOT0		
x	0	Main Flash memory	Main Flash memory is selected as the boot space
0	1	System memory	System memory is selected as the boot space
1	1	Embedded SRAM	Embedded SRAM is selected as the boot space

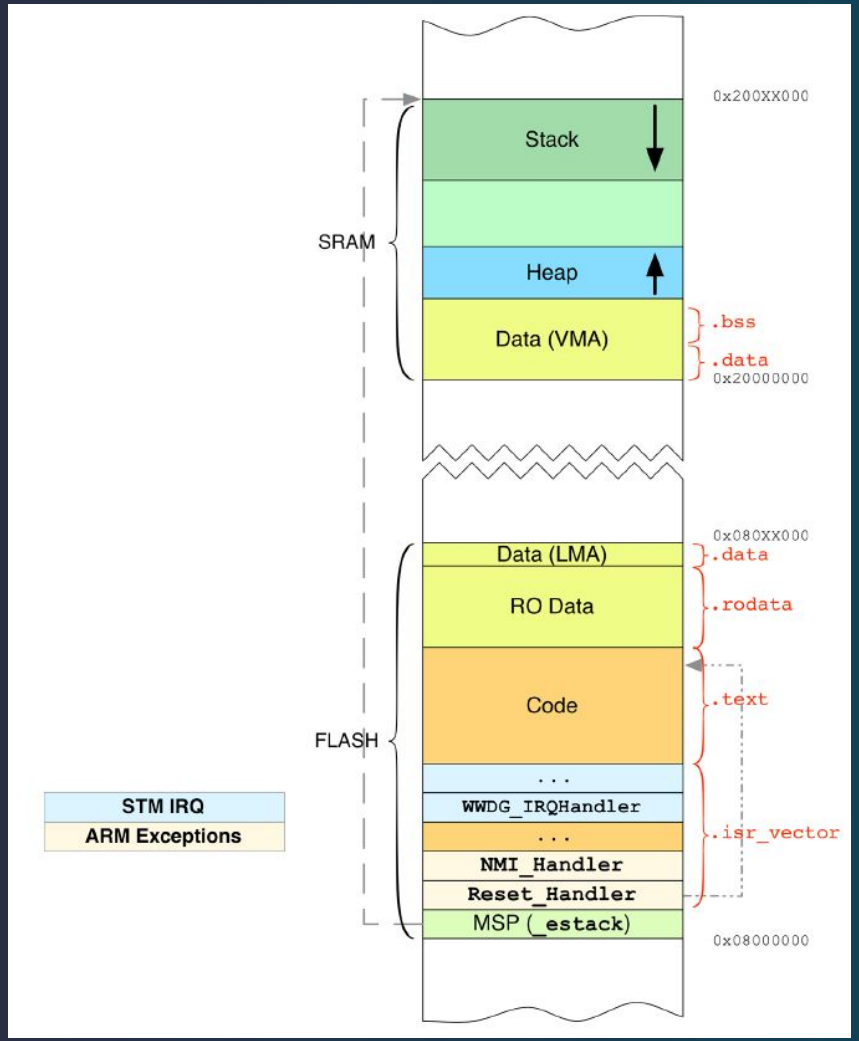
- Flash: 0x08000000
- ST's bootloader: 0x1fff0000
- SRAM: 0x20000000

Execution

Recall: Program in Memory Layout

- Memory sections
  - **.text** : instructions
  - **.data** : initialized data
  - **.bss** (blocks started by symbols): uninitialized data
  - **.rodata** : const data and strings
- **.isr_vector** :
  - Entry point for exceptions
- Startup code
  - **Data initialization** : .data is copied from flash to SRAM
  - **Zero initialization** : .bss is set to all zeros

Qustion: How startup code works?



Execution

NTURACING

Bootup Sequence - Setup C Runtime

- Referred as crt0 in Linux (https://en.wikipedia.org/wiki/Crt0)
- Different frameworks/ OSes have different implementations, here we discuss STM32CubeIDE
- Recall: Linker script
  - Refer to *_FLASH.ld in STM32CubeIDE
  - Defines _sdata, _edata, etc. to mark the ranges of some sections
- Recall: PC is loaded from the second entry of isr_vector, which is **Reset_Handler** and starts execution form there
- Startup code (Reset_Handler)
  - Written in assembly as C environment is not ready
  - Performs data and zero initialization and jump to main()
  - Refer to start_*.s in Core/Startup/ in STM32CubeIDE

NTURACING

C Technique 2: Macros

References: Macro Metaprogramming (https://mailund.dk/posts/macro-metaprogramming/)

Macros

NTURACING

Macros

- #define
- # (stringizing operator)
- ## (token pasting operator)
- Use examples from:
  - Basics: Macro Metaprogramming (https://mailund.dk/posts/macro-metaprogramming/)
  - Advanced: C Preprocessor tricks, tips, and idioms (https://github.com/pfultz2/Cloak/wiki/C-Preprocessor-tricks,-tips,-and-idioms)
  - Zephyr RTOS utilities macros (https://docs.zephyrproject.org/latest/doxygen/html/group__sys-util.html)

NTURACING

Exceptions

References: CH7, 8, *The Definitive Guide*; Ch7, *Mastering STM32*

Exceptions

NTURACING

Recall: Exceptions

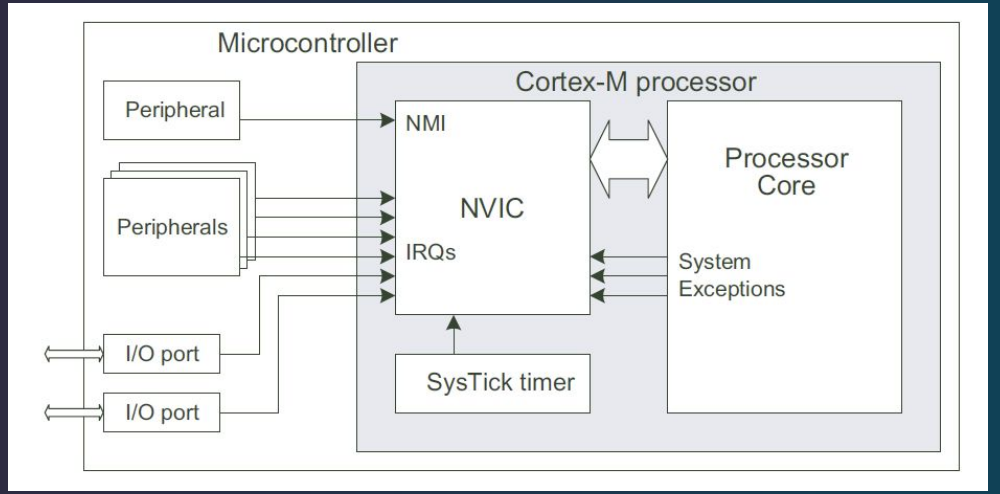
- Hardware: interrupt, software: fault
- Suspends current CPU execution and enters interrupt service routine (ISR), functions to handle exceptions
- Various types to represent different events

Exceptions

NTURACING

Nested Vectored Interrupt Controller (NVIC)

- Interrupt controller in Cortex-M
- **Nested** : An exception can be handled while another is executing
- **Vectored** : Supports multiple exceptions by a vector table
- Exception number 1~15 are system exception and 16 and above for external interrupt used by peripherals via Interrupt Request (**IRQ**) lines



Exceptions

NTURACING

isr_vector

- Entry points for each exceptions
- Typically starts at 0x00000000, can be relocated by modifying VTOR register of NVIC
- In practive, it's an array of function pointers of prototype void isr_func(void)
- Defined in start_*.s in Core/Startup/ in STM32CubeIDE
- The first section in flash, defined in *_FLASH.ld in STM32CubeIDE

Number	Exception type	Priority[a]	Function
1	Reset	-3	Reset
2	NMI	-2	Non-Maskable Interrupt
3	Hard Fault	-1	All classes of Fault, when the fault cannot activate because of priority or the Configurable Fault handler has been disabled.
4	Memory Management[c]	Configurable[b]	MPU mismatch, including access violation and no match. This is used even if the MPU is disabled or not present.
5	Bus Fault[c]	Configurable	Pre-fetch fault, memory access fault, and other address/memory related.
6	Usage Fault[c]	Configurable	Usage fault, such as Undefined instruction executed or illegal state transition attempt.
7	SecureFault[d]	Configurable	SecureFault is available when the CPU runs in *Secure state*. It is triggered by the various security checks that are performed. For example, when jumping from Non-secure code to an address in Secure code that is not marked as a valid entry point.
8-10	-	-	RESERVED
11	SVCall	Configurable	System service call with SVC instruction.
12	Debug Monitor[c]	Configurable	Debug monitor – for software based debug.
13	-	-	RESERVED
14	PendSV	Configurable	Pending request for system service.
15	SysTick	Configurable	System tick timer has fired.
16-[47/239/479][e]	IRQ	Configurable	IRQ Input

Exceptions

NTURACING

Exception States

- Exceptions have three binary states
  - Disabled (default) or enabled
  - Pending (waiting to be handled) or not pending
  - Active (being handled) or inactive
- Exceptions will only be handled if it's enabled and pending, then it will enter active state

Exception Masking

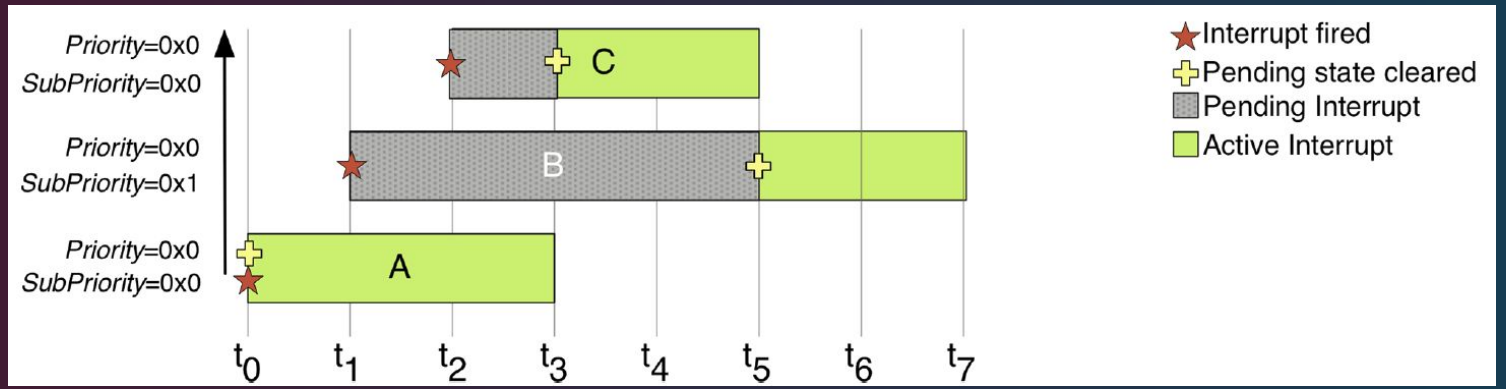
- Exceptions can be temporarily disabled (called **masking**): `__disable_irq()`, `__enable_irq()`
- Typically used by embedded OSes to perform critical sections
- Pending state is ignored if it's masked (but it will be handled right after when it's unmasked)
- Reset, NMI, hard fault, and SVCall are not maskable

Exceptions

NTURACING

Exception Priority

- A higher-priority exception (smaller number in priority level) **preempts** a lower-priority exception.
- Priority levels are up to 8 bits (4 bits in STM32) and split into **preemption priority** and **sub priority** of configurable lengths.
  - Preemption Priority: Higher-priority ones preempt lower-priority ones.
  - Sub Priority: Higher-priority ones will run first when fired at the same time.



Exceptions

NTURACING

Exception Execution

- **Stacking** : When exceptions are handled, the caller-saved registers (r0~r3, r13) and LR and other control registers are saved to the stack, they will be **unstack** ed after ISR had finished so that the registers are the same before and after an exception is handled
- While some CPUs uses special instructions to return from ISR (which requires special support from the compiler or inline assembly), Cortex-M uses a special `EXC_RETURN` value loaded to LR to indicate returning form ISR
- Some optimizations:
  - **Tail-Chaining** : No stacking/unstacking when exceptions are handled in series
  - **Late Arrival** : Exceptions with higher priority is fired during stacking, then it will be handled first without additional stacking

NTURACING

LAB 2: Debug

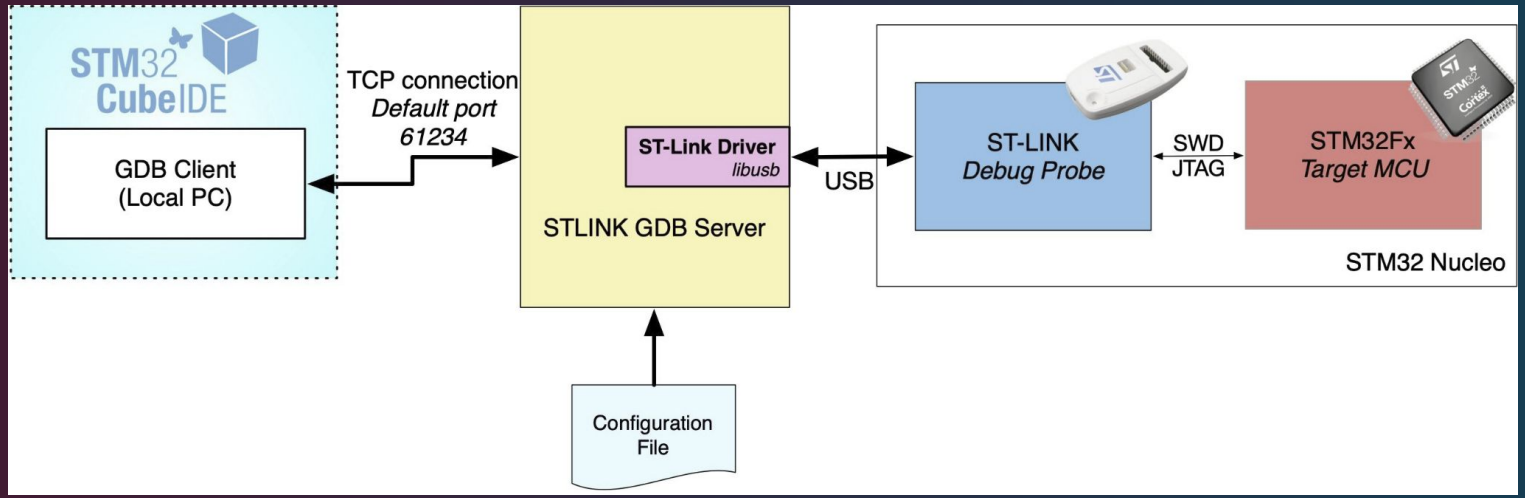
References: Ch 5, *Mastering STM32*

Debug

NTURACING

Debug Pipeline

- Debug communication protocols:
  - Joint test action group (JTAG)
  - Serial wire debug (SWD) (only for Cortex-M)



Debug

NTURACING

Debug Features

- Step over, step into, step out
- Breakpoints
- Variables
- Function call stack
- Memory
- Peripheral registers
- Cotrex-M live watch