Microprocessors

Fall 2020

2. Architecture

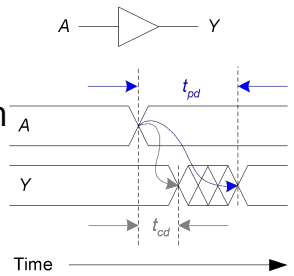
Tentative Weekly Schedule

- Week x1 - Introduction to Course
- **Week x2 - Architecture**
- Week x3 - Assembly Language Introduction
- Week x4 - Assembly Language Memory and Control
- Week x5 - Embedded C, Toolchain and Debugging
- Week x6 - Interrupts
- Week x7 - Timers
- Week x8 - Modulation
- Week x9 - Serial Communications I
- Week xA - Serial Communications II
- Week xB - Analog Interfacing
- Week xC - Memory and DMA
- Week xD - RTOS
- Week xE - Wireless Communications

Review: Transition from Logic Design to Microprocessors

For Combinational Logic:

- **Propagation delay:** t_{pd} = max delay from input to output
- **Contamination delay:** t_{cd} = min delay from input to output



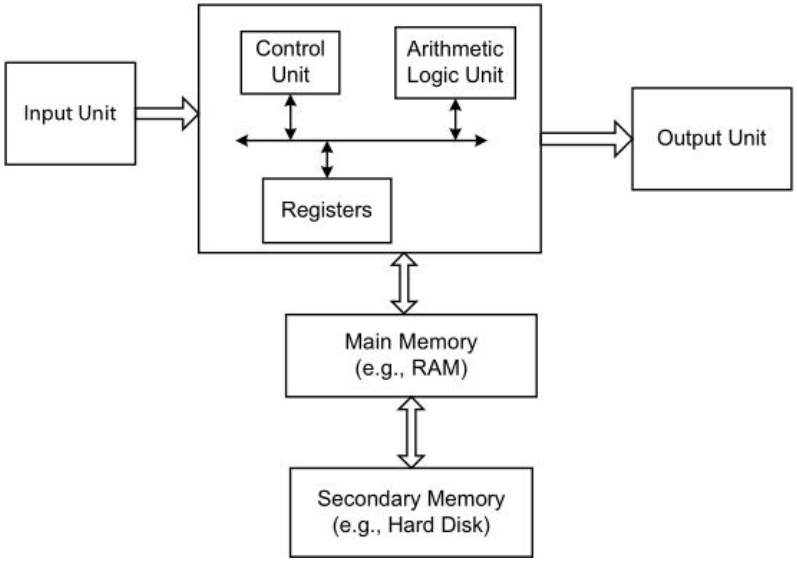
- In *Logic Design course*, we cared about **timing within a clock cycle** and tried to **optimize** what we can achieve **within that clock cycle**.

- In *Microprocessors course*, we don't care about single clock cycle, but we care about **how fast** we can **compute** and try to **optimize number of clock cycles**.

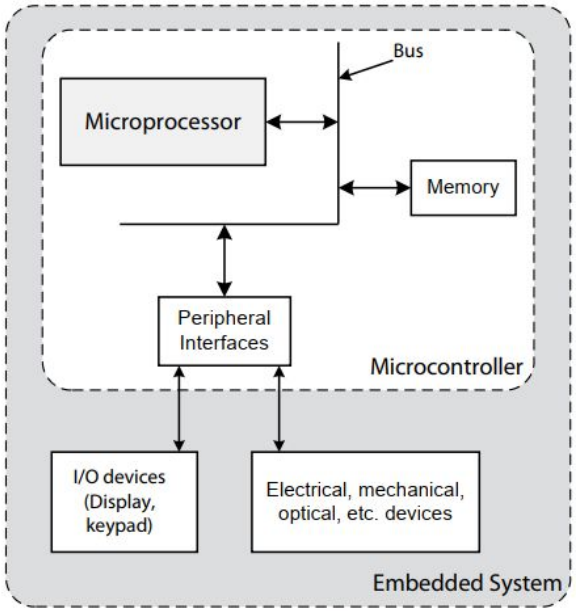
Review: Classes of Computers

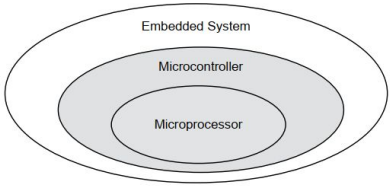
- **Desktop computers**
  - General purpose, variety of software
  - Subject to cost, performance tradeoff
- **Server computers**
  - Network based
  - High capacity, performance, reliability
  - Range from small servers to building sized
- **Embedded computers**
  - Hidden as components of systems
  - Strict power, performance, cost constraints

Review: General Purpose Computer



Review: Embedded System





Review: What is inside a Microcontroller?

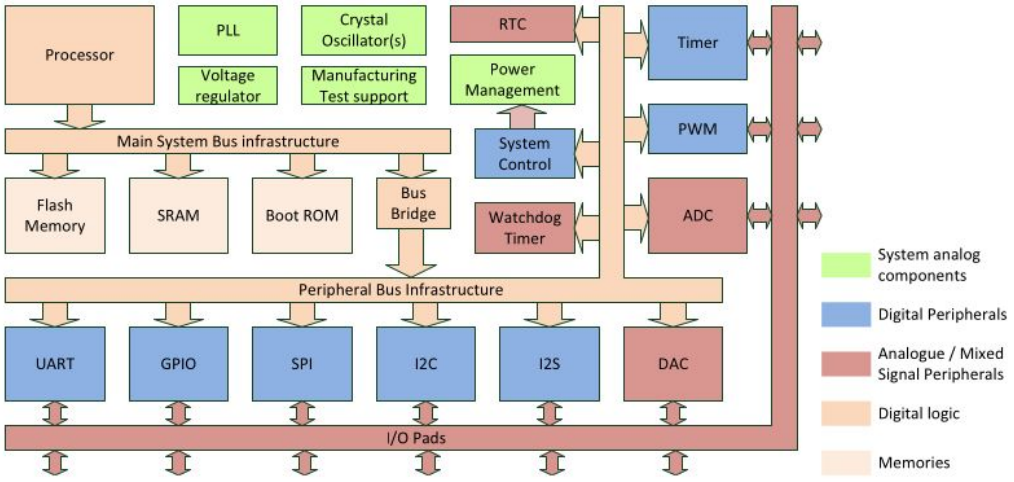


Figure 1.9
A simple microcontroller.

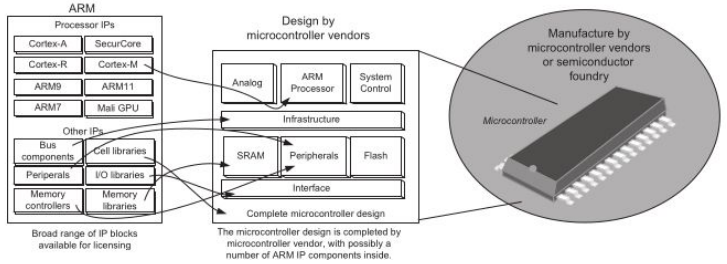
Review: Characteristics of microprocessors

- Low power
- Fast interrupt response
- High code density
- Debug
- OS support
- Ease of use
- High software portability and reusability
- Upgrade and downgrade path
- Tool chain support
- Low cost

Let's talk about ARM...

ARM: Brief history

- Acorn Computers, Ltd is founded in Cambridge, UK in 1978
- Worked on projects using 6502 processor
- Created Acorn RISC Machine (ARM1) to meet the fast growing demand in industry
- Later founded ARM Company



ARM: Recently...

Forbes

97,090 views | Sep 13, 2020, 07:21pm EDT

It's Official– NVIDIA Acquires Arm For $40 Billion To Create What Could Be A Computing Juggernaut

Forbes

1,311 views | Sep 16, 2020, 08:09am EDT

Apple Bet The Farm On Arm. Now NVIDIA's In Charge, Does It Stick Or Twist?

ARM: Overview of Processor Families

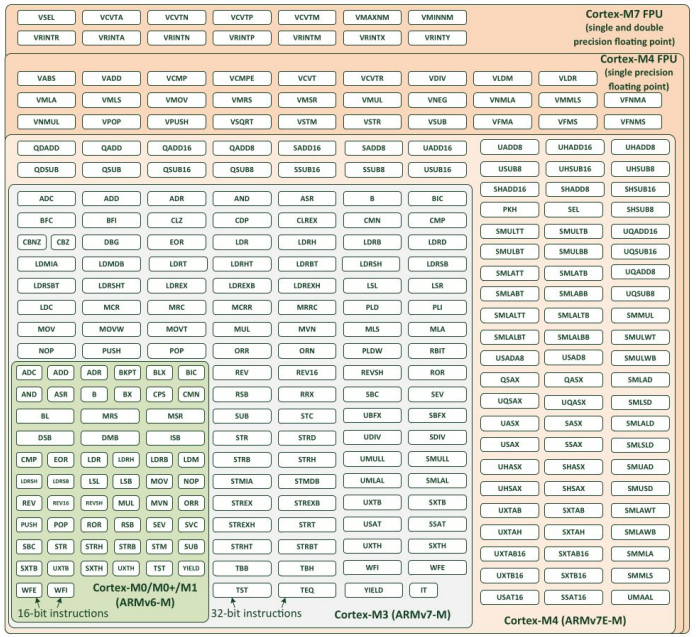
There are three branches in ARM Cortex family cores:

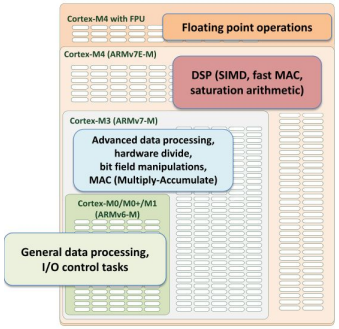
- **Cortex-A** - Application processors. Designed as fully functional computers capable of running complex operating systems. Commonly used in mobile phones, tablets, and laptops such as Raspberry PI, Android phones, iPhones, iPads, ...
- **Cortex-R** - Real-time processors. Designed to be fast reacting to events. Have low interrupt latency and more deterministic in tight situations. Often used in critical systems where data interpretation is essential such as medical devices, car systems, basebands and low-level device controllers, such as hard drive controllers.
- **Cortex-M** - Microcontroller series. Designed to be ultra-low-power and small form-factor. Runs at a way slower clock speed then A and R series. They are used in robotic systems and small consumer electronics.

ARM: Cortex-M applications

Processor	Applications
Cortex-M0, Cortex-M0+ processors	General data processing and I/O control tasks. Ultra low power applications. Upgrade/replacement for 8-bit/16-bit microcontrollers. Low-cost ASICs, ASSPs
Cortex-M1	Field Programmable Gate Array(FPGA) applications with small to medium data processing complexity. (For high-complexity data processing there are FPGAs with built-in Cortex-A processors such as Xilinx Zynq-7000 and some of the Altera Arria V SoCs and Cyclone V SoCs).
Cortex-M3	Feature-rich/high-performance/low-power microcontrollers. Light-weight DSP applications.
Cortex-M4	Feature-rich/high-performance/low-power microcontrollers. DSP applications. Applications with frequent single precision floating point operations.
Cortex-M7	Feature-rich/very high performance power microcontrollers. DSP applications. Applications with frequent single or double precision floating point operations.

ARM: Cortex-M Processor Family





Let's talk about computer architecture...

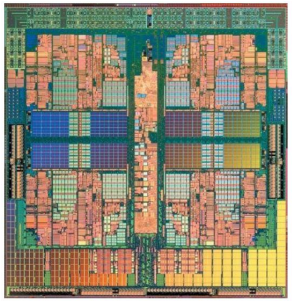
Architecture vs. Microarchitecture

- **Instruction Set Architecture (Architecture)**
  - programmer's view of a computer (memory & register)
  - operations (instructions and how they work)
  - execution semantics (interrupts)
  - input/output
  - data types/sizes
  - ARM, x86, MIPS, SPARC, PowerPC
  - ARMv6-M Architecture Reference Manual
- **Microarchitecture (Organization)**
  - tradeoffs on how to implement ISA for some metric (speed, energy, cost)
  - pipeline depth, number of pipelines, cache size, silicon area, peak power, execution ordering, bus widths, ALU widths..
  - different implementations of Intel i3, i5, i7 cores of 2010-2020

Same Arch & Different Microarch

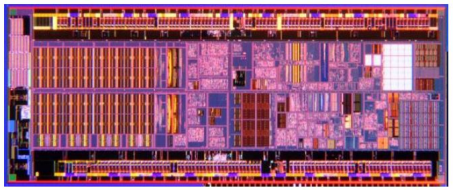
AMD Phenom x4

- x86 ISA
- Quad core
- 125W
- Decode 3 Instructions/Cycle/Core
- 64 KB L1 I/D Cache
- 512 KB L2 Cache
- Out-of-order
- 2.6Ghz



Intel Atom

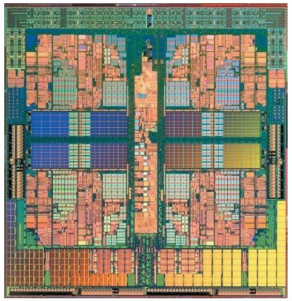
- x86 ISA
- Single Core
- 2W
- Decode 2 Instructions/Cycle/Core
- 32 KB L1 I, 24 KB D Cache
- 512 KB L2 Cache
- In-order
- 1.6Ghz



Different Arch & Different Microarch

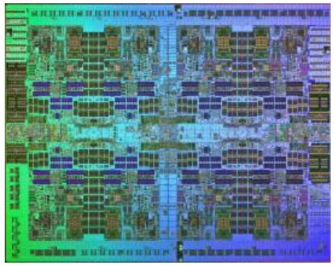
AMD Phenom x4

- x86 ISA
- Quad core
- 125W
- Decode 3 Instructions/Cycle/Core
- 64 KB L1 I/D Cache
- 512 KB L2 Cache
- Out-of-order
- 2.6Ghz



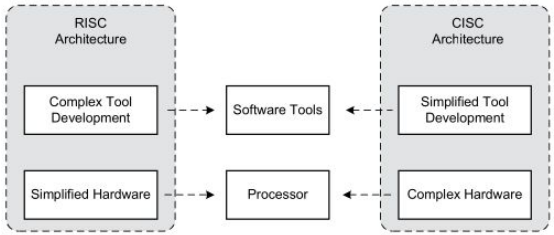
IBM POWER7

- Power ISA
- Eight Core
- 200W
- Decode 6 Instructions/Cycle/Core
- 32 KB L1 I/D Cache
- 256 KB L2 Cache
- Out-of-order
- 4.25Ghz



Complexity based ISA Classification

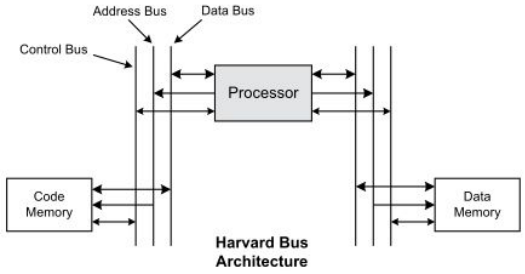
- **Complex instruction set computers (CISC)**
  - Uses single complex instruction that can perform many operations usually requiring multiple cycles
  - Usually supports varying instruction length
  - More complex hardware, simplified compiler
- **Reduced instruction set computers (RISC)**
  - Uses simplified instructions that can use many instructions to perform an operation usually completing one instruction per cycle
  - More simplified hardware, complex compiler to transform code

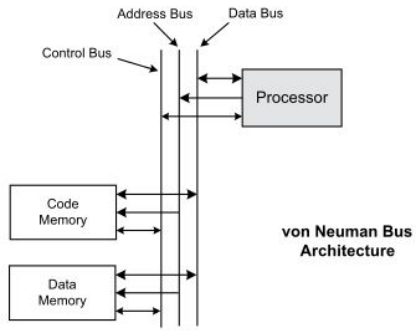


Memory Interface-Based Architecture Classification

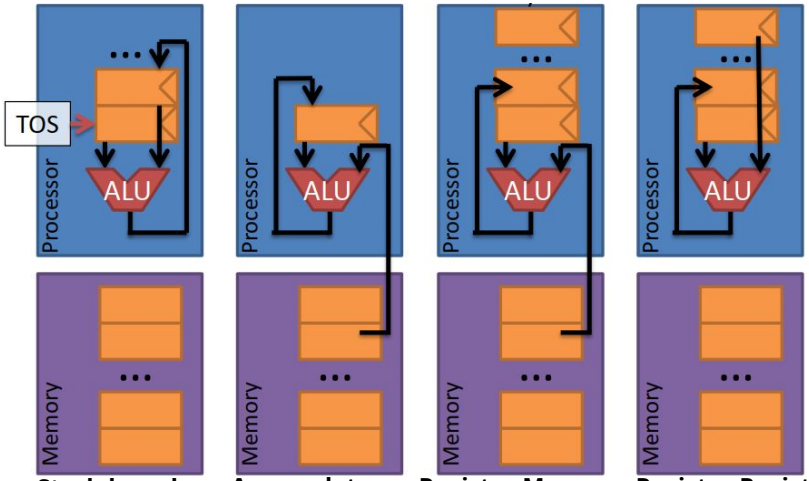
There are two widely used memory interface architectures

- **von Neumann** - uses a common bus for both data and code
- **Harvard** - uses separate buses for data and code





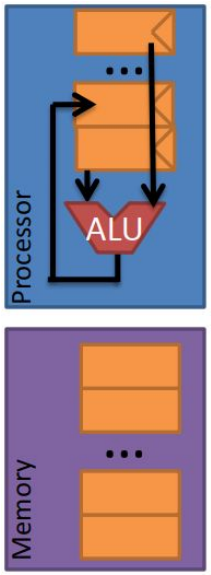
Where do operands come and results go?



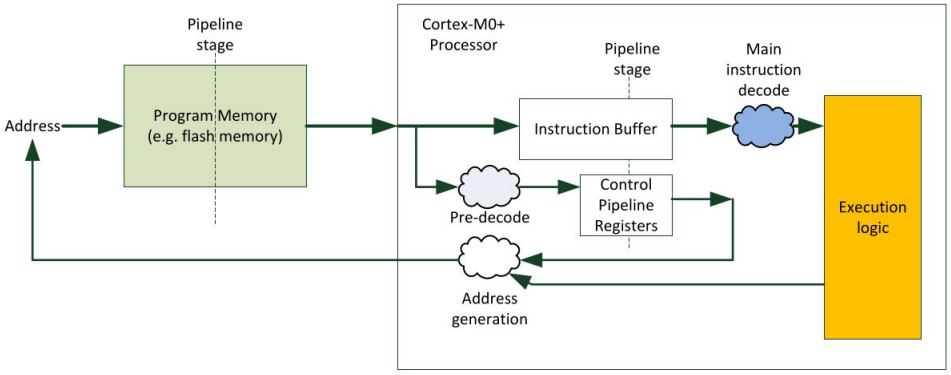
Stack-based	Accumulator	Register-Memory	Register-Register
PUSH A	LOAD A	LOAD R1, A	LOAD R1, A
PUSH B	ADD B	ADD R3, R1, B	LOAD R2, B
ADD	STORE C	STORE R3, C	ADD R3, R1, R2
POP C			STORE R3, C

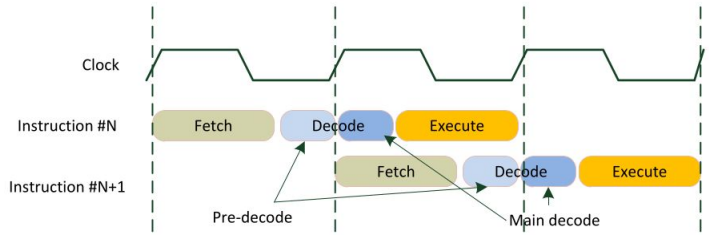
Register-Register Architecture

- Also called **load-store architecture**
- Register-Register architecture **operands need to be in registers** (or immediates)
- Special instructions are used to get and send data from / to memory. (`LDR`, `STR`)
- Common in RISC architectures.
- 2 or 3 explicitly named operands are allowed. (`ADD R1, R2, R3`)



ARM Cortex M0+ Two stage pipeline

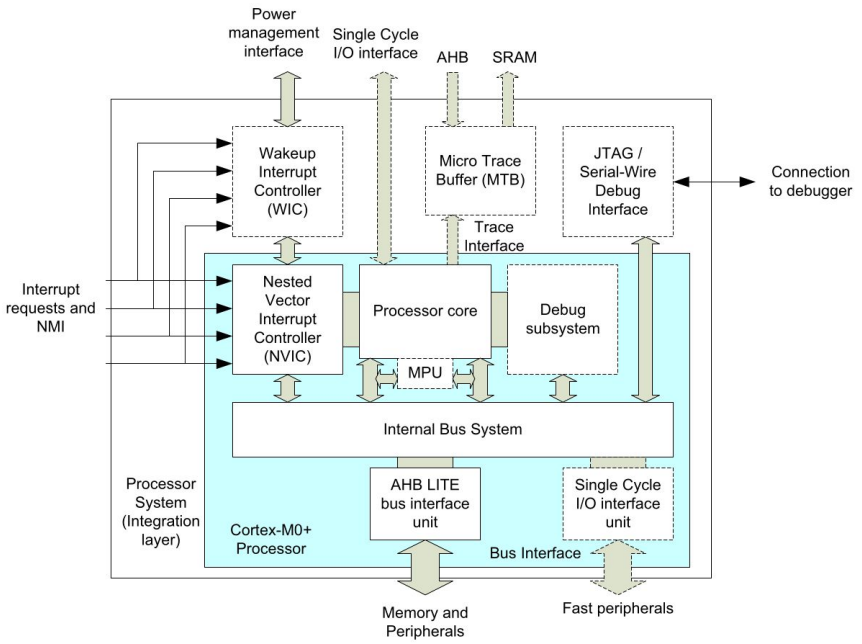




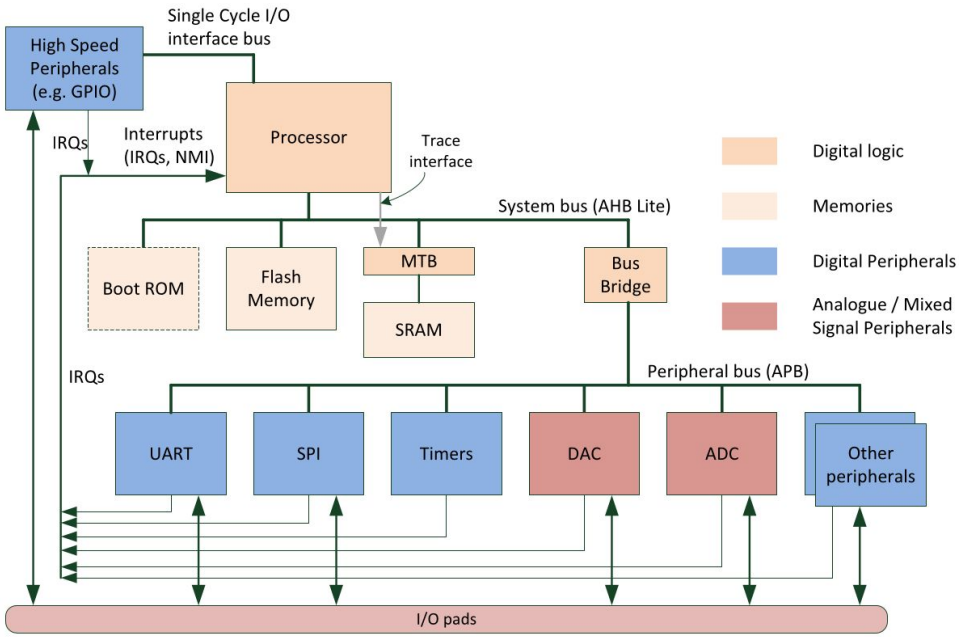
ARM Cortex M0+ Processors

- Cortex-M0+ instruction set is based on **Thumb ISA** which is built for more compact support and implements a **subset** of them.
- **Most instructions are 16 bit** (some are 32)
- 32-bit addressing support supporting **up to 4GB** of memory space
- System bus is based on **AHB-Lite** supporting 8-bit, 16-bit, and 32-bit data transfers.
- Peripherals can be connected to a simpler bus based on the **APB** protocol

ARM Cortex M0+ Simplified block diagram



ARM Cortex M0+ Typical System



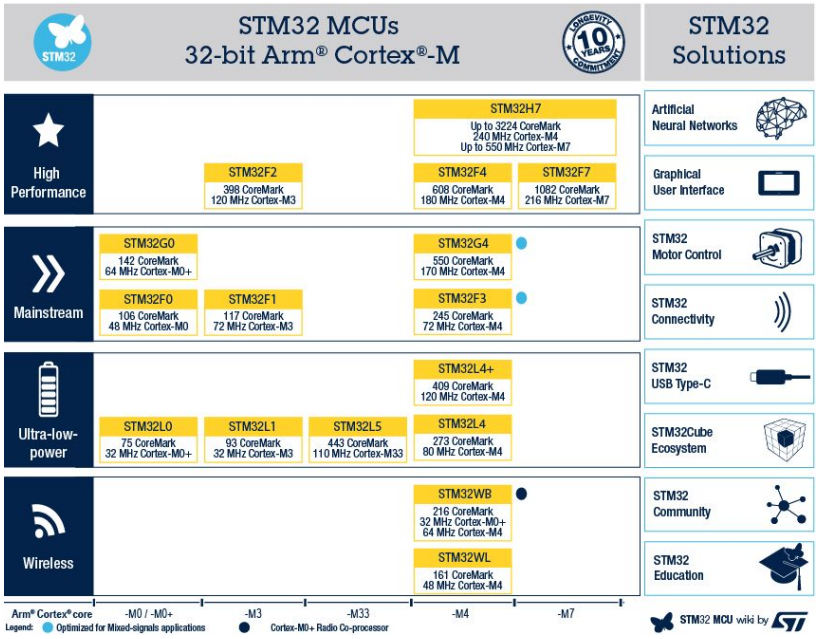
ARM Cortex M0+ based microcontrollers

- ABOV Semiconductor A31G11x, A31G12x, A31G314
- Cypress PSoC 4000S, 4100S, 4100S+, 4100PS, 4700S, FM0+
- Epson S1C31W74, S1C31D01, S1C31D50
- Holtek HT32F52000
- Microchip (Atmel) SAM C2, D0, D1, D2, DA, L2, R2, R3
- NXP LPC800, LPC11E60, LPC11U60
- NXP (Freescale) Kinetis E, EA, L, M, V1, W0
- Renesas Synergy S124, Synergy S128
- Renesas RE, RE01
- Silicon Labs (Energy Micro) EFM32 Zero, Happy
- ST STM32 L0, G0

The following chips have a Cortex-M0+ as a secondary core:

- Cypress PSoC 6200 (one Cortex-M4F + one Cortex-M0+)
- ST WB (one Cortex-M4F + one Cortex-M0+)

ST Microelectronics Cortex-M portfilo



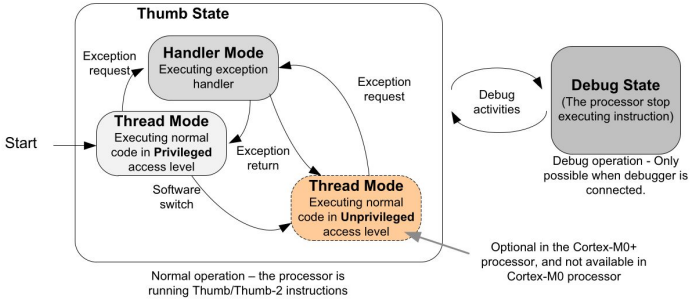
Microcontroller choice decisions

- Low Power / Energy Efficiency
  - low-power features (sleep modes)
  - small gate count (dynamic / static power)
  - technology size (leakage, voltage)
- High Code Density
  - smaller binary size (memory)
- Low (Interrupt) Latency
  - determinism (exact response time)
- Ease of Use
  - Development environment, debugger, software, middleware support
  - Software portability, scalability, reusability, flexibility
- Debugging support

Operation Modes and States

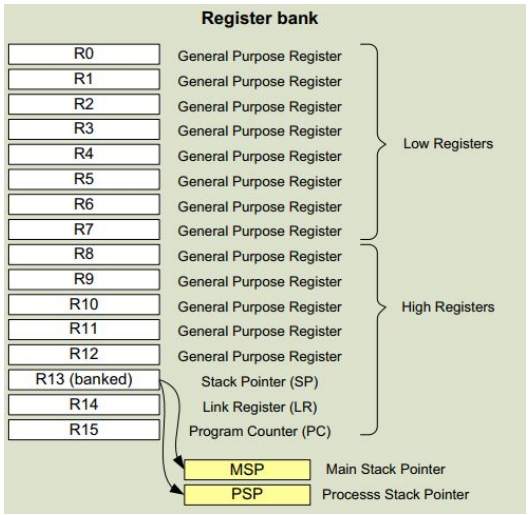
ARMv6-M architecture has two operation modes and two stages with privileged and unprivileged access levels.

- **Thumb state** is when the processor is running a program
  - **Thread mode**
  - **Handler mode**
- **Debug state** is when the processor is halted



Regular and Special Registers

- In Cortex-M0+, there are **16 32-bit** registers (13 general purpose and 3 specific use)
  - **R0 - R15** with **R0-R12** being general purpose registers.
  - due to size, *mostly* **only low registers** can be used with 16-bit Thumb code.
  - data needs to be loaded from memory to registers to be processed (**load/store**)

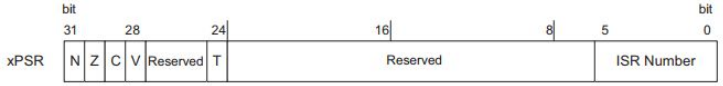


Special use registers

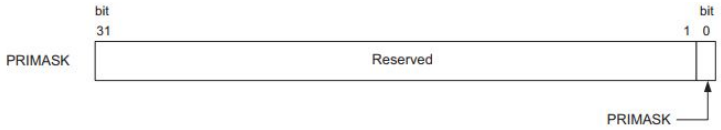
- **R13** - Stack Pointer (**SP**) - points to the top of the stack
  - **POP** and **PUSH** instructions are used to access memory.
  - Banked - **Main** (MSP) and Process Stack Pointer (PSP). Can be changed, but only one of them is visible
  - Special register access instructions **MRS** and **MSR** is used to change between them.
  - Access is 32-bits so **lowest 2 bits are always 0**.
- **R14** - Link Register (**LR**) - keeps the return address of a subroutine or function call
  - **Lowest bit** is usually set to 1 to indicate **Thumb** state. Otherwise it can imply an attempt to switch the processor to ARM state - which is not supported - tuhs can cause a fault.
- **R15** - Program Counter (**PC**) - points to the next instruction in line
  - **Lowest bit** is usually set to 1 to indicate **Thumb** state. Otherwise it can imply an attempt to switch the processor to ARM state - which is not supported - thus can cause a fault.

Special Registers

- **xPSR** - Program Status Register - holds information about program execution, exception number and the ALU flags.

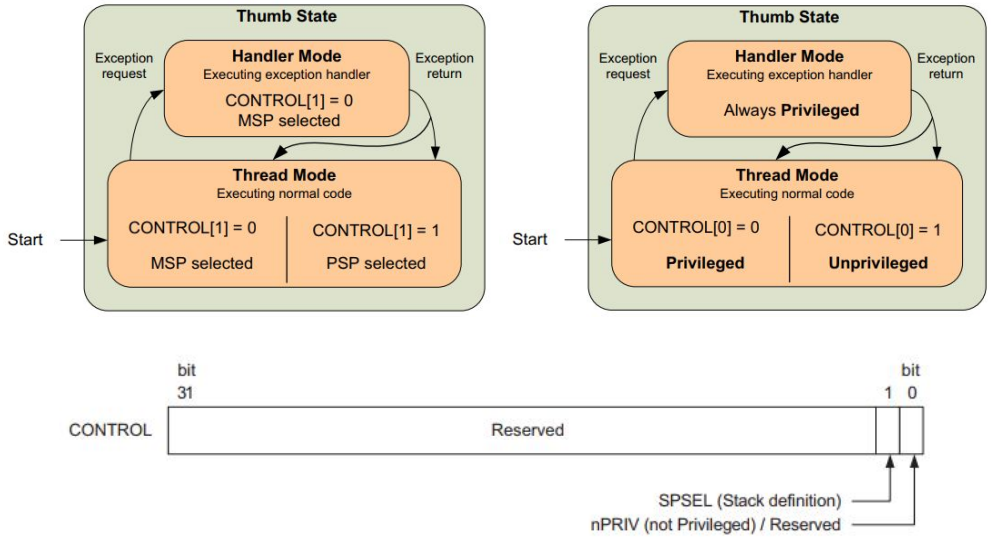


- **PRIMASK** - Interrupt Mask special Register is used to **block all interrupts except** *Non-Maskable Interrupt (NMI)* and the *HardFault* exception



- **CONTROL** - special register that is used to change processor mode and stack pointer

Controlling processor state with CONTROL



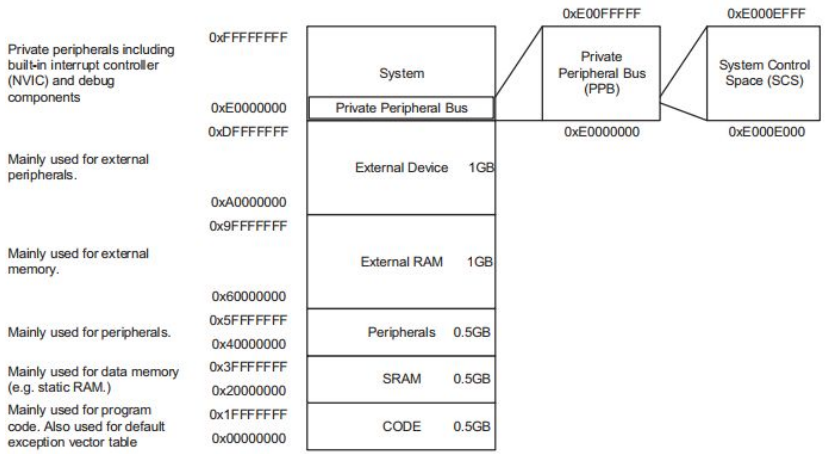
ALU Flags and examples

Flag	Descriptions
N (bit 31)	Set to bit[31] of the result of the executed instruction. When it is "1," the result has a negative value (when interpreted as a signed integer). When it is "0," the result has a positive value or equal zero.
Z (bit 30)	Set to "1" if the result of the executed instruction is zero. It can also be set to "1" after a compare instruction is executed if the two values are the same.
C (bit 29)	Carry flag of the result. For unsigned addition, this bit is set to "1" if an unsigned overflow occurred. For unsigned subtract operations, this bit is the inverse of the borrow output status.
V (bit 28)	Overflow of the result. For signed addition or subtraction, this bit is set to "1" if a signed overflow occurred.

Operation	Results, flags
0x70000000 + 0x70000000	Result = 0xE0000000, N = 1, Z = 0, C = 0, V = 1
0x90000000 + 0x90000000	Result = 0x20000000, N = 0, Z = 0, C = 1, V = 1
0x80000000 + 0x80000000	Result = 0x00000000, N = 0, Z = 1, C = 1, V = 1
0x00001234 − 0x00001000	Result = 0x00000234, N = 0, Z = 0, C = 1, V = 0
0x00000004 − 0x00000005	Result = 0xFFFFFFFF, N = 1, Z = 0, C = 0, V = 0
0xFFFFFFFF − 0xFFFFFFFC	Result = 0x00000003, N = 0, Z = 0, C = 1, V = 0
0x80000005 − 0x80000004	Result = 0x00000001, N = 0, Z = 0, C = 1, V = 0
0x70000000 − 0xF0000000	Result = 0x80000000, N = 1, Z = 0, C = 0, V = 1
0xA0000000 − 0xA0000000	Result = 0x00000000, N = 0, Z = 1, C = 1, V = 0

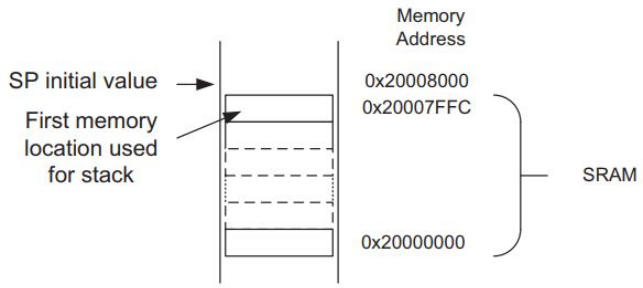
Memory System

- Cortex-M processors have 4 GB of memory address space.
- Memory space is architecturally defined into a number of regions
- As a result all the Cortex-M devices have the same programming model for interrupt control and debug.



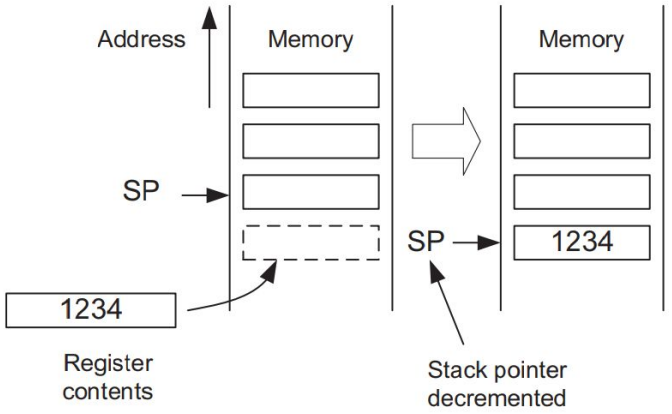
Stack Memory Operation

- **Stack memory** is a memory usage mechanism that allows the system memory to be used as **temporary** data storage that behaves as a **first-in-last-out** (FILO) buffer.
- Content can be **added** with **PUSH** instruction and can be **acquired** using **POP** instruction
- Usually located at the top of SRAM, and **grows downwards** - meaning if data is added it gets **filled toward lower addresses** (**full-descending**)
- **Stack Pointer** shows the top of the stack



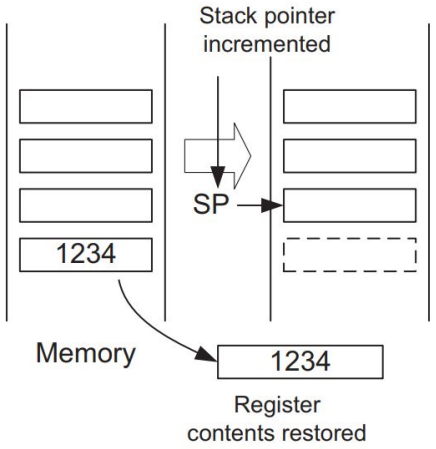
Stack Memory Operation – PUSH

- In case of a **PUSH** operation: (`PUSH {R1}`)
  - First Stack Pointer is **decremented by 4**
  - Then, the register data is written to the address that the Stack Pointer points to.



Stack Memory Operation – POP

- In case of a **POP** operation: (`POP {R2}`)
  - First, the memory content that the Stack Pointer points to is written to the Register in the instruction
  - Then, Stack pointer is **incremented by 4**.



Exceptions and Interrupts

- **Exceptions** are events that cause **changes to program control**
- the processor suspends the current executing task and executes a part of the program code called the **exception handler**
- after the handler is completed, the processor resumes the normal program execution
- **Interrupts** are a **subset** of exceptions commonly referred as **IRQ.**
  - Cortex-M0+ processors support up to 32 external interrupts
- The exception handlers for interrupt events are commonly called **Interrupt Service Routines (ISRs)**.

Exception numbers and priority

- Each exception has an **exception number**. This number is reflected in registers including **xPSR** and used to define the **exception vector address.**
- Exception numbers **1-16** are common in all M series, IRQ number varies based on manufacturer.
- Except **Reset**, **NMI** and **HardFault** - all other exceptions have a programmable **priority** level.

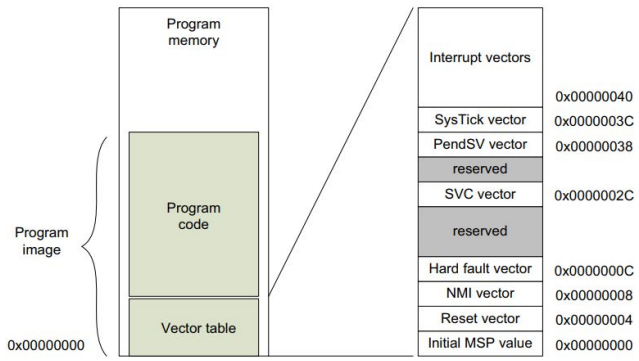
Exception type	Exception number	Description
Reset	1	Power on reset or system reset.
NMI	2	Non-Maskable interrupt—highest priority exception that cannot be disabled. For safety critical events.
HardFault	3	For fault handling—activated when a system error is detected.
SVCall	11	Supervisor call—activated when SVC instruction is executed. Primarily for OS applications.
PendSV	14	Pendable service (system) call—activate by writing to an interrupt control and status register. Primarily for OS applications.
SysTick	15	System Tick timer exception – typically used by an OS for a regular system tick exception. The system tick timer (SysTick) is an optional[a] timer unit inside the Cortex®-M processor.
IRQ0 to IRQ31[b]	16–47	Interrupts—can be from external sources or from on-chip peripherals.

Nested Vectored Interrupt Controller

- Used for **prioritizing** the interrupt requests and **handling** other exceptions
- The interrupt management function is controlled by registers in the **NVIC** located within the **System Control Space (SCS)**
- NVIC supports:
  - Flexible interrupt management
  - Nested interrupt support
  - Vectored exception entry
  - Interrupt masking

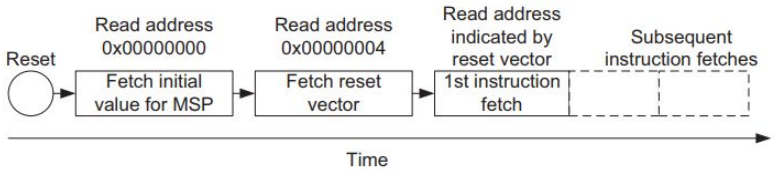
Vector Table

- The beginning of the program image has the **vector table** which contains the starting address of exceptions.
- The size of the vector table depends on the number of interrupts implemented.
- First two addresses are important for reset operation. The next two are important for faulty recovery.

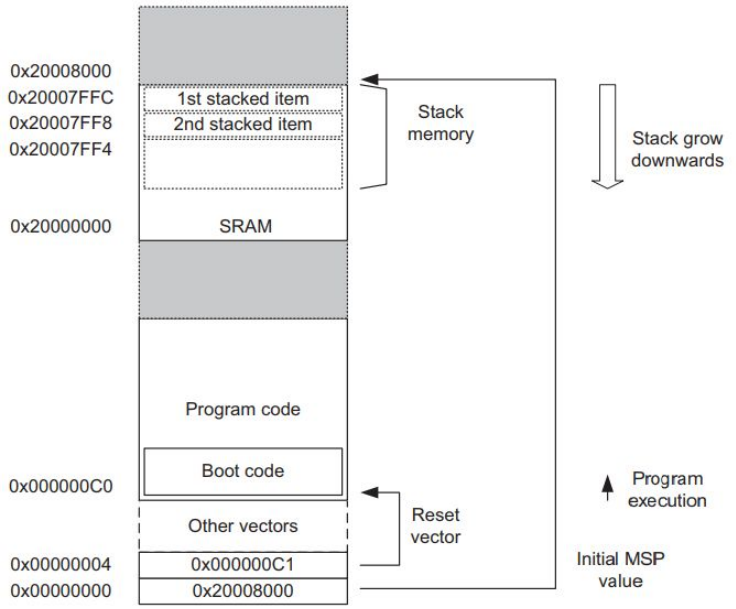


Reset Sequence

- We have already seen what happens after reset in terms of software routines.
- First address in the vector table indicates the stack pointer value. (**SP** is set)
- Second address in the vector table is the Reset Handler. (**PC** is set)
- Then PC makes the jump to the pointing address.



Example Stack Pointer and PC init

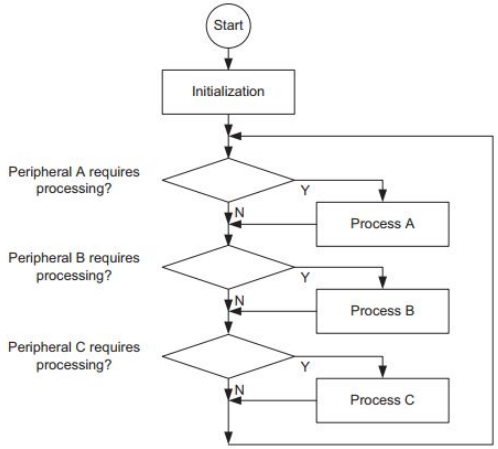


Let's talk about software design...

Software Program Flows - Polling

Polling (also called **super-loop**)

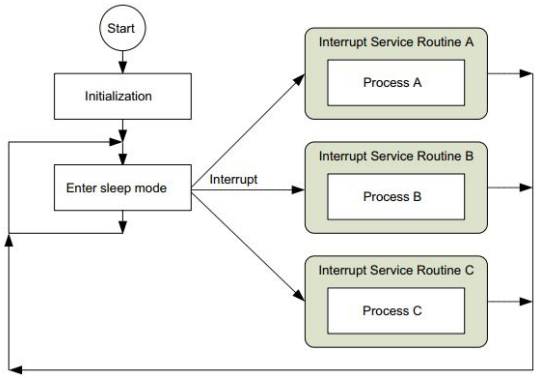
- uses one big loop, and completes tasks in order
- As tasks get bulkier or tasks increase in number, the execution frequency of tasks decreases



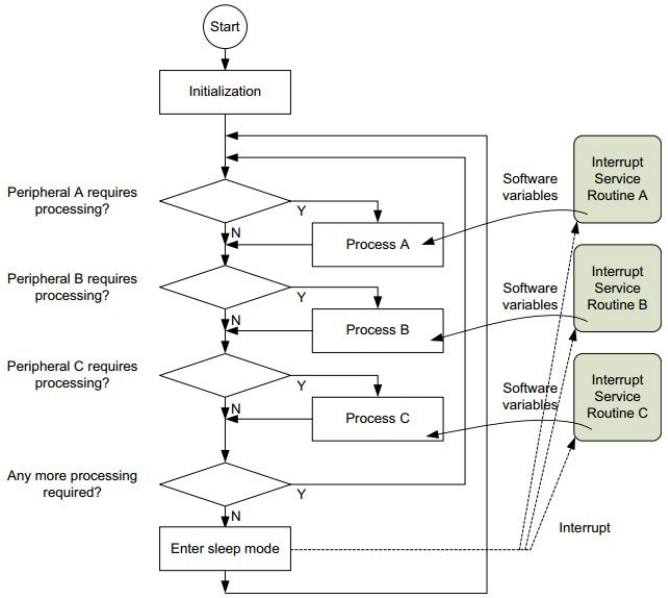
Interrupt Driven

Interrupt driven approach uses interrupts that will handle predefined requests, or incoming events

- Otherwise can sleep to reduce power consumption
- Different priorities can be set to interrupts ensuring higher priority routine gets executed no matter what.



Hybrid with Polling and Interrupt Driven



Handling Concurrent Processes

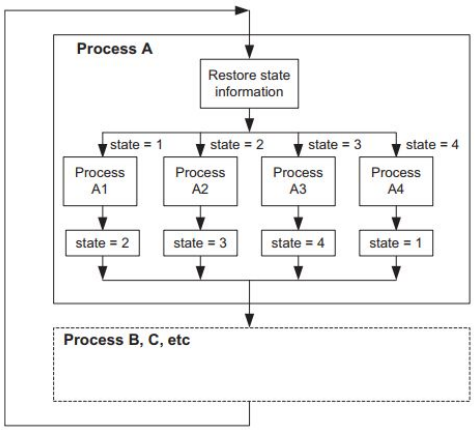
In some cases an application process could take a significant amount of time to complete

- e.g if A takes a long time, B and C will not be able to respond to requests fast enough, and even might not be able to run at all (**starve**).

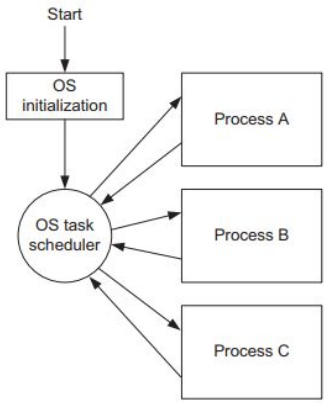
Two solutions:

- Break down a long processing task to a sequence of states managing with an FSM. **Each time** the process is processed, **only one state is executed**
- Use a Real-Time Operating System (**RTOS**) to manage multiple tasks.

Handling Concurrent Processes – Solutions



FSM



RTOS

Programming Language Decisions

Language	Pros and cons
C/C++	Pros: • easy to learn • portable • easy handling of complex data structures Cons: • limited/no direct access to core register and stack • no direct control over instruction sequence generate • no direct control over stack usage
Assembly	Pros: • allows direct control to each instruction step and all memory operations • allows direct access to instructions that cannot be generated with C Cons: • takes longer time to learn • difficult to manage data structure • less portable (syntax of assembly language in different tool chains can be different)

We will get ourselves familiarized with Assembly for the first couple weeks, then transition into C.

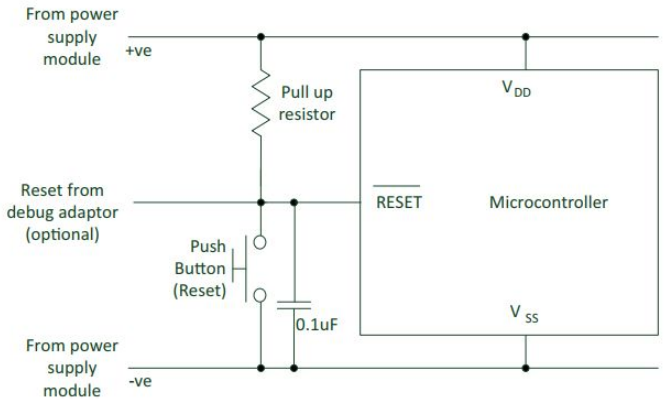
Inside a program image

When we compile code for the target platform, there are different components:

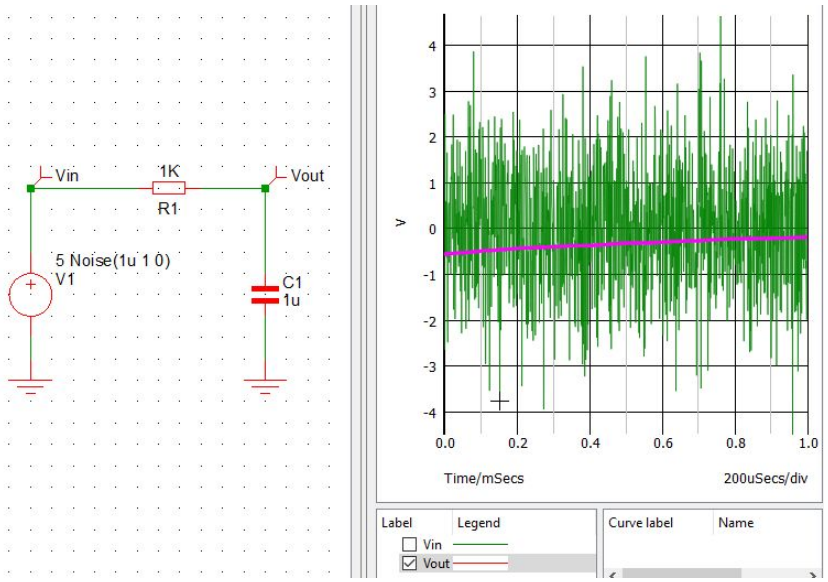
- **vector table** - contains the starting address of each **exception** and **interrupt**
- **reset handler** - some software and hardware initialization routines, clock initialization
- **startup code** - initialization of data and calling of **main()** function.
- **application code** - your code
- runtime library functions - any functions that you didn't implement
- **other data** - other global and static variables

Programming Microcontrollers

A simple reset circuitry looks like this.

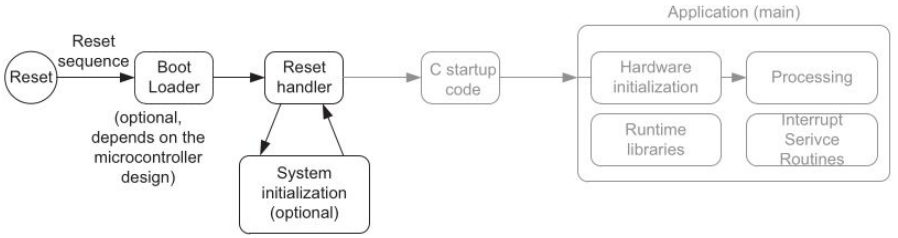


Role of decoupling capacitors



Microcontroller Startup Sequence

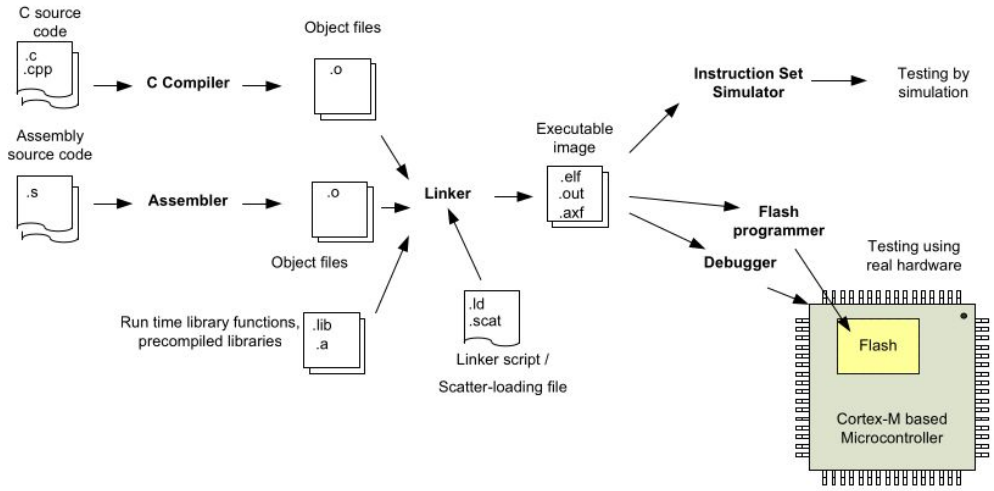
- Most modern microcontrollers hold compiled program in on-chip flash memory.
- After the processor is reset, it obtains the **initial stack pointer** value and **reset address** from **vector table**.
- Executes the **reset handler** in startup code.



Bootloader

- A piece of software burned at the beginning of memory that initializes any selected peripherals and boots up the actual code.
- Before booting up the actual code, it offers a way to re-flash the code with a new one without the need for a debugger.
- It can be through **serial port**, **usb** or any other communication method using **wired** or **wireless** channels.

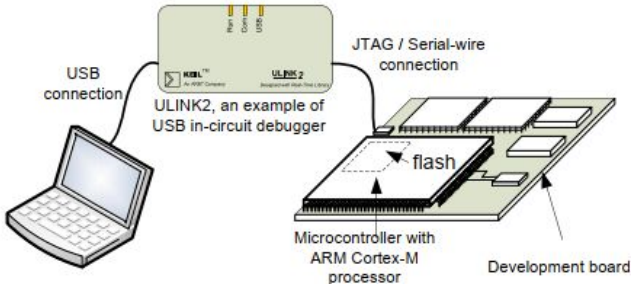
Typical program generation flow



Programming / Debugging

- After generating the binary (elf / bin) we can program the board using a **debugger** (or bootloader if present)
  - Usually are more expensive and has debugging capabilities.
- Program can be debugged using an **in-circuit debugger**.

Nucleo G031K8 board has built-in debugger on the back. No additional hardware required



This week

- Find / order hardware components
- Install VisUAL / CubeIDE + G0
  - import a project and compile the program. Will send out info later this week.
- Read Chapter 2, 3 and 4 from Yiu
- Assignment 2

Links

- https://en.wikipedia.org/wiki/ARM_Cortex-M
- https://www.st.com/en/microcontrollers-microprocessors/stm32-32-bit-arm-cortex-mcus.html
- https://www.arm.com/products/silicon-ip-cpu
- https://developer.arm.com/ip-products/processors/cortex-m
- https://developer.arm.com/documentation/ddi0419/c
- https://www.simetrix.co.uk/