Jádro mikrokontroléru

a jeho typické vlastnosti

na příkladu současného jádra ARM Cortex M0+

Mikroprocesorové a vestavěné systémy (IMP)

Richard Růžička

Fakulta informačních technologií VUT v Brně

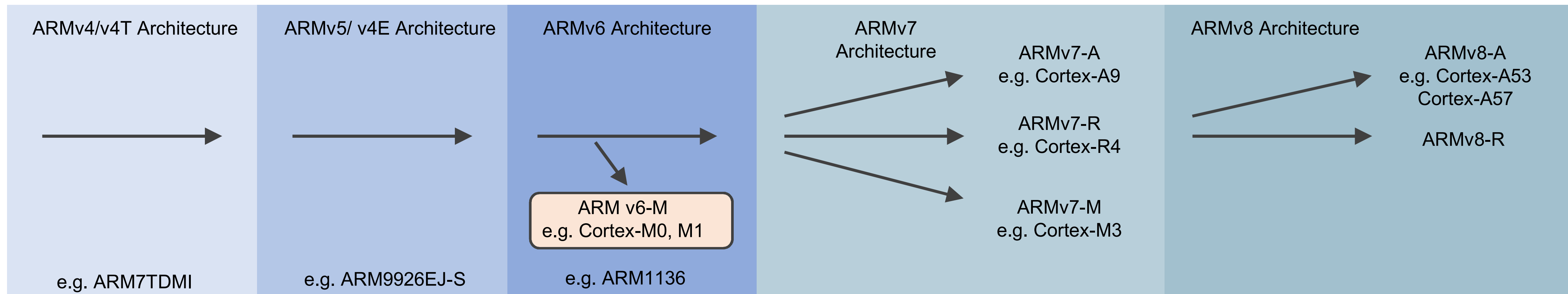
What is ARM Architecture

- ARM architecture is a family of RISC-based processor architectures
  - Well-known for its power efficiency;
  - Hence widely used in mobile devices, such as smartphones and tablets
  - Designed and licensed to a wide eco-system by ARM

- ARM Holdings
  - The company designs ARM-based processors;
  - Does not manufacture, but licenses designs to semiconductor partners who add their own Intellectual Property (IP) on top of ARM's IP, fabricate and sell to customers;
  - Also offer other IP apart from processors, such as physical IPs, interconnect IPs, graphics cores, and development tools.

ARM Processors vs. ARM Architectures

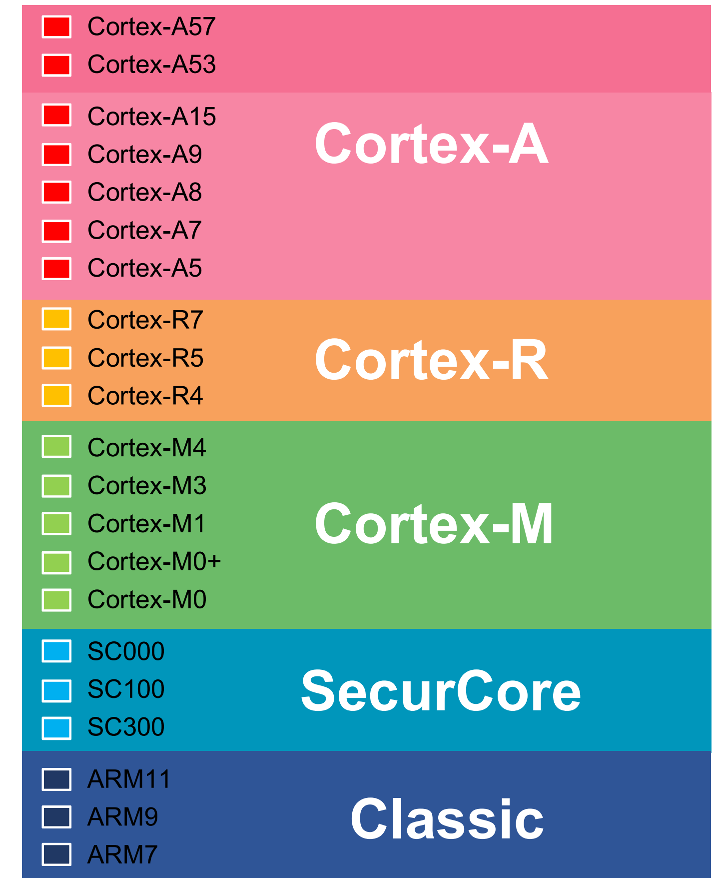
- ARM architecture
  - Describes the details of instruction set, programmer's model, exception model, and memory map
  - Documented in the Architecture Reference Manual
- ARM processor
  - Developed using one of the ARM architectures
  - More implementation details, such as timing information
  - Documented in processor's Technical Reference Manual



As of Dec 2013

ARM Processor Families

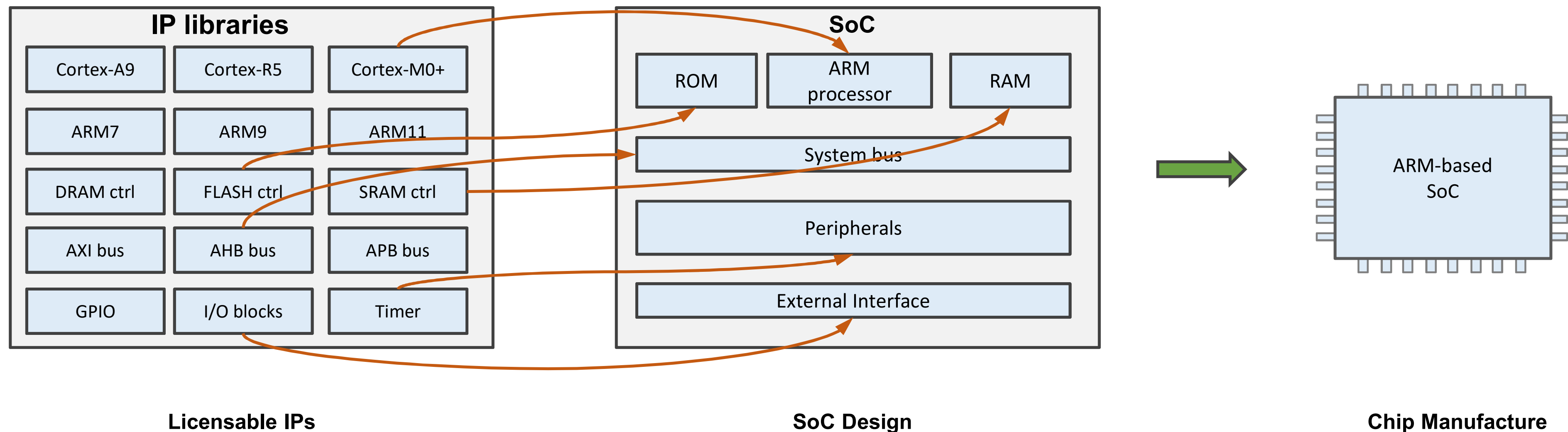
- Cortex-A series (Application)
  - High performance processors capable of full Operating System (OS) support;
  - Applications include smartphones, digital TV, smart books, home gateways etc.
- Cortex-R series (Real-time)
  - High performance for real-time applications;
  - High reliability
  - Applications include automotive braking system, powertrains etc.
- Cortex-M series (Microcontroller)
  - Cost-sensitive solutions for deterministic microcontroller applications;
  - Applications include microcontrollers, mixed signal devices, smart sensors, automotive body electronics and airbags;
- SecurCore series
  - High security applications.
- Previous classic processors
  - Include ARM7, ARM9, ARM11 families



As of Dec 2013

Design an ARM-based SoC

- Select a set of IP cores from ARM and/or other third-party IP vendors
- Integrate IP cores into a single chip design
- Give design to semiconductor foundries for chip fabrication



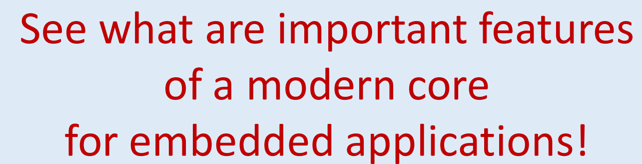
ARM Cortex-M Series

- Cortex-M series: Cortex-M0, M0+, M1, M3, M4.
- Energy-efficiency
  - Lower energy cost, longer battery life
- Smaller code
  - Lower silicon costs
- Ease of use
  - Faster software development and reuse
- Embedded applications
  - Smart metering, human interface devices, automotive and industrial control systems, white goods, consumer products and medical instrumentation

ARM Cortex-M Series Family

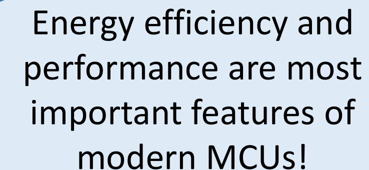
Processor	ARM Architecture	Core Architecture	Thumb®	Thumb®-2	Hardware Multiply	Hardware Divide	Saturated Math	DSP Extensions	Floating Point
Cortex-M0	ARMv6-M	Von Neumann	Most	Subset	1 or 32 cycle	No	No	No	No
Cortex-M0+	ARMv6-M	Von Neumann	Most	Subset	1 or 32 cycle	No	No	No	No
Cortex-M1	ARMv6-M	Von Neumann	Most	Subset	3 or 33 cycle	No	No	No	No
Cortex-M3	ARMv7-M	Harvard	Entire	Entire	1 cycle	Yes	Yes	No	No
Cortex-M4	ARMv7E-M	Harvard	Entire	Entire	1 cycle	Yes	Yes	Yes	Optional

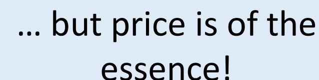
ARM Cortex-M0+ Processor Overview



Cortex-M0+ Processor Overview

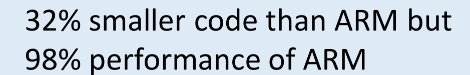
- Cortex-M0+ Processor
  - Entry-level 32-bit ARM Cortex processor designed for a broad range of embedded applications
  - An optimized superset of the Cortex-M0
- High Performance Efficiency
  - 11.21 µW/MHz dynamic power requirement
- Low Power Consumption
  - Longer battery life – especially critical in mobile products
- Enhanced Determinism
  - The critical tasks and interrupt routines can be served quickly in a known number of cycles
- Lower Cost
  - Reduced 32-bit-based system cost, close to those legacy 8-bit and 16-bit devices (e.g. can be priced at less than $1)





Cortex-M0+ Processor Features

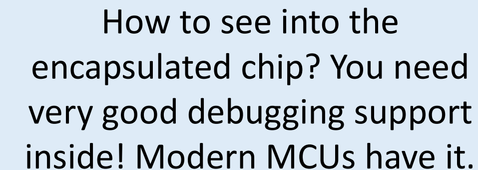
- 32-bit Reduced Instruction Set Computing (RISC) processor
- Von Neumann architecture
- Instruction set
  - Based on the 16-bit Thumb® instruction set and includes Thumb®-2 technology
  - High code density with 32-bit performance



- 2-stage pipeline
- Performance efficiency
  - 0.95-1.36 DMIPS/MHz (Dhrystone Million Instructions Per Second / MHz)
- Supported Interrupts
  - Non-maskable Interrupt (NMI) + 1 to 32 physical interrupts
  - 4 interrupt priority levels

Cortex-M0+ Processor Features

- Supports Sleep Modes
  - Integrated WFI and WFE Instructions and Sleep On Exit capability
  - Optional Retention Mode with ARM Power Management Kit
  - Sleep & Deep Sleep Signals
- Enhanced Instructions
  - Single-Cycle (32x32) Multiply
- Debug
  - JTAG or 2-pin Serial-Wire Debug (SWD) Ports
  - Up to 4 Breakpoints and up to 2 Watchpoints
- Memory Protection Unit (MPU)
  - Optional 8 region MPU with sub regions and background region
- Trace
  - Optional Micro Trace Buffer

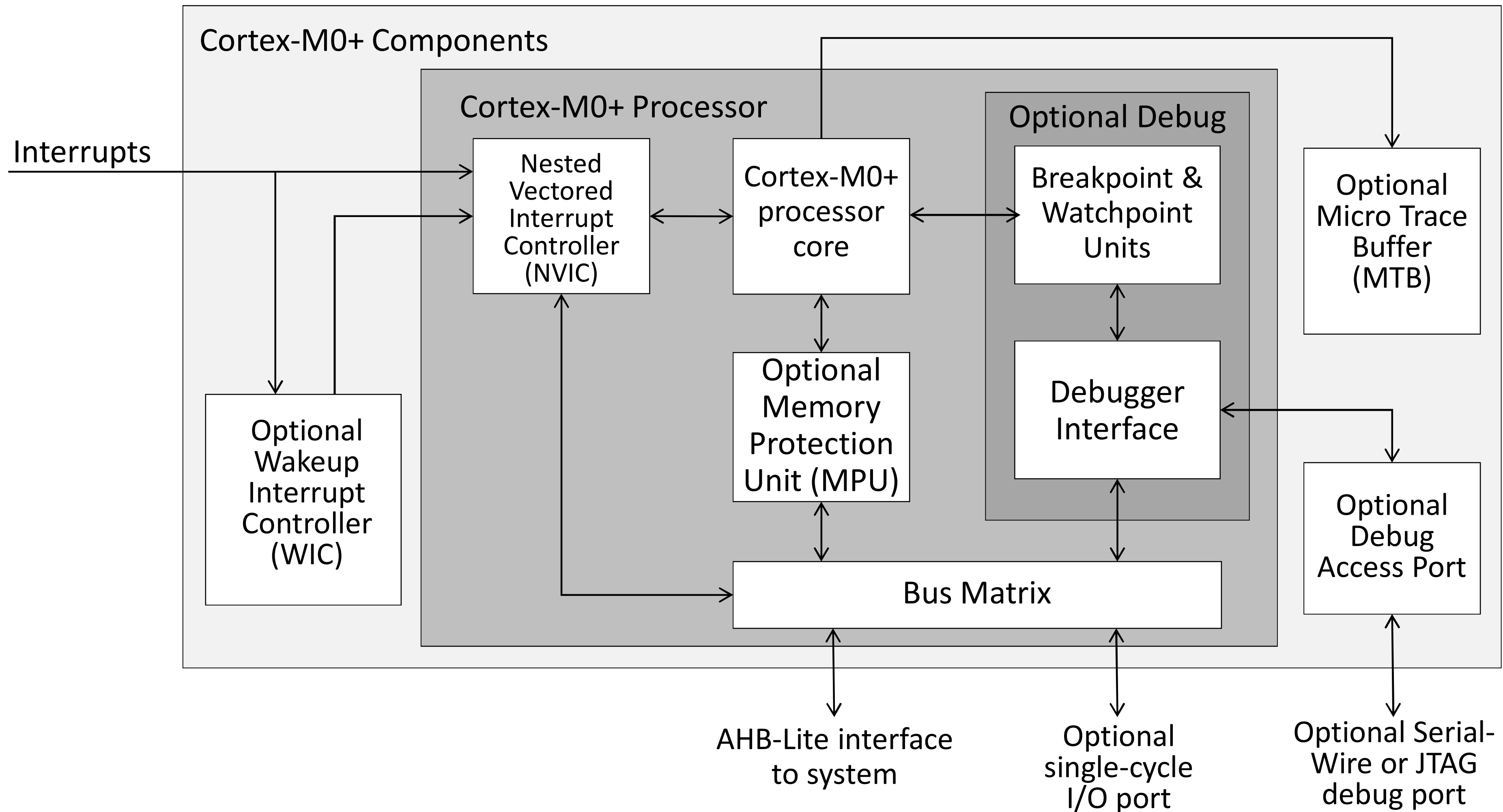


Cortex-M0+ Processor Features

- Cortex-M0+ processor is designed to meet the challenges of low dynamic power constraints while retaining light footprints
  - 180 nm ultra low power process –52 µW/MHz
  - 90 nm low power process – 9.8 µW/MHz
  - 40 nm G process – 3 µW/MHz

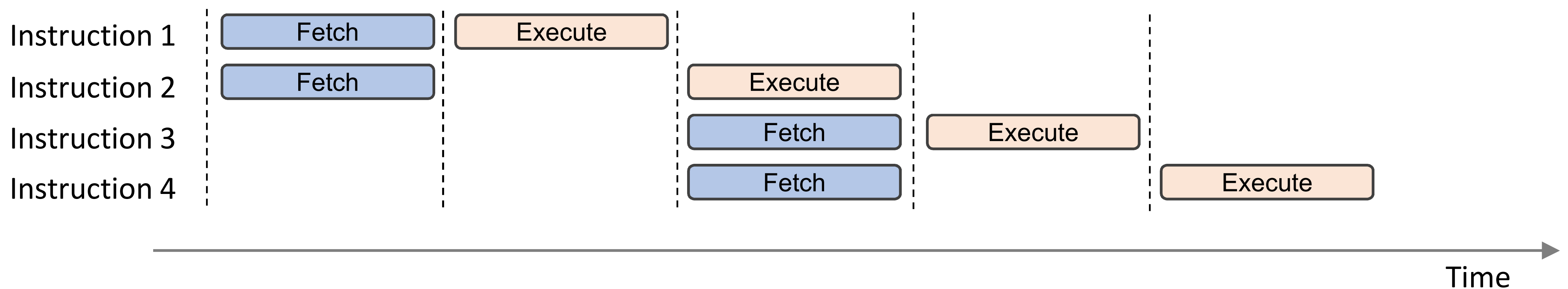
ARM Cortex-M0+ Implementation Data			
Process	180ULL (7-track, typical 1.8v, 25C)	90LP (7-track, typical 1.2v, 25C)	40G 9-track, typical 0.9v, 25C)
Dynamic Power	52 µW/MHz	9.8 µW/MHz (8.17 **µA/MHz**)	3 µW/MHz
Floorplanned Area	0.13 mm^2	0.035 mm^2	0.009 mm^2

Cortex-M0+ Block Diagram



Cortex-M0+ Block Diagram

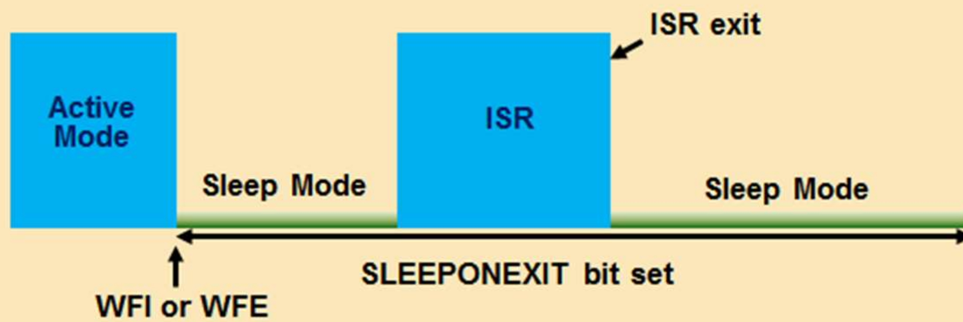
- Processor core
  - Contains internal registers, the ALU, data path, and some control logic
  - Registers include sixteen 32-bit registers for both general and special usage
- Processor pipeline stages
  - All loads and stores always complete in program order
  - All Strongly-ordered load/stores are automatically synchronized to the instruction stream
  - Device and Normal load/stores may be pipelined
  - Up to two instructions can be fetched in one transfer (16-bit instructions)

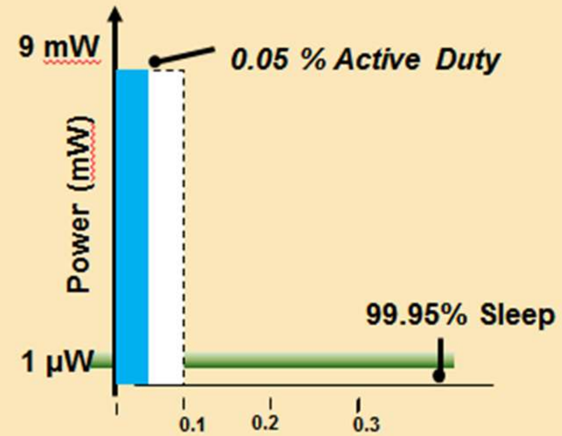


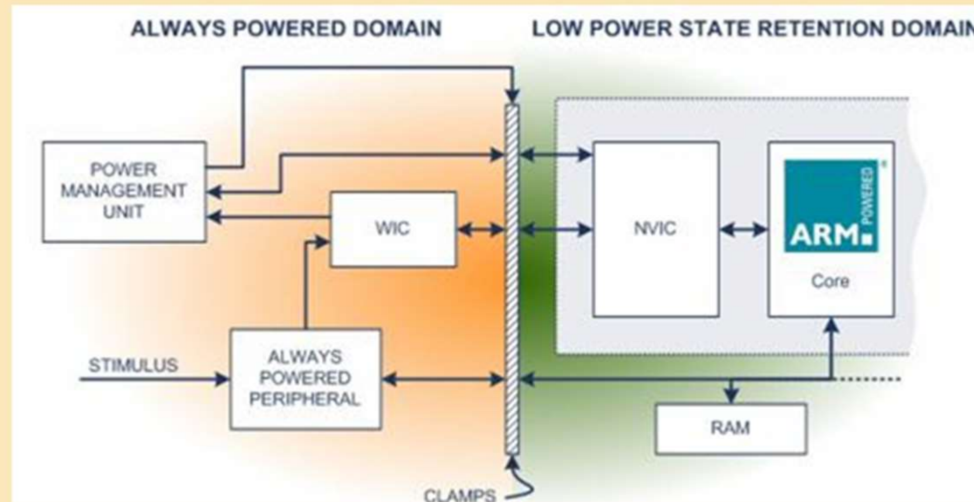
Cortex-M0+ Block Diagram

- Nested Vectored Interrupt Controller (NVIC)
  - Up to 32 external interrupt inputs, each with four levels of priority and a dedicated Non-maskable Interrupt (NMI) input
  - Automatically handles nested interrupts, such as comparing priorities between interrupt requests and the current priority level
- Wakeup Interrupt Controller (WIC)
  - For low-power applications, the microcontroller can enter sleep mode by shutting down most of the components
  - When an interrupt request is detected, the WIC can inform the power management unit to power up the system
- Memory Protection Unit (optional)
  - Used to protect memory content, e.g. make some memory regions read-only or preventing user applications from accessing privileged application data

Smart Power Management



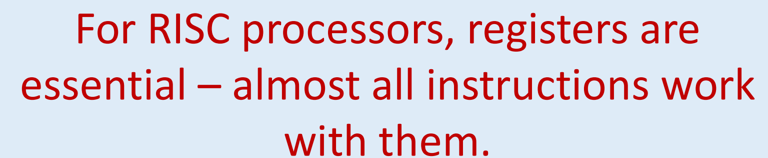




Cortex-M0+ Block Diagram

- Bus interconnect
  - Single 32-bit AMBA-3 AHB-Lite system interface that provides simple integration to all system peripherals and memory
  - Optional single 32-bit single-cycle I/O port
  - Optional single 32-bit slave port that supports the DAP
- Debug subsystem
  - Handles debug control, program breakpoints, and data watchpoints
  - When a debug event occurs, it can put the processor core in a halted state, where developers can analyse the status of the processor at that point, such as register values and flags
  - Support for unlimited software breakpoints using BKPT instruction
  - Non-intrusive access to core peripherals and zero-waitstate system slaves through a compact bus matrix. A debugger can access these devices, including memory, even when the processor is running

ARM Cortex-M0+ Processor Registers

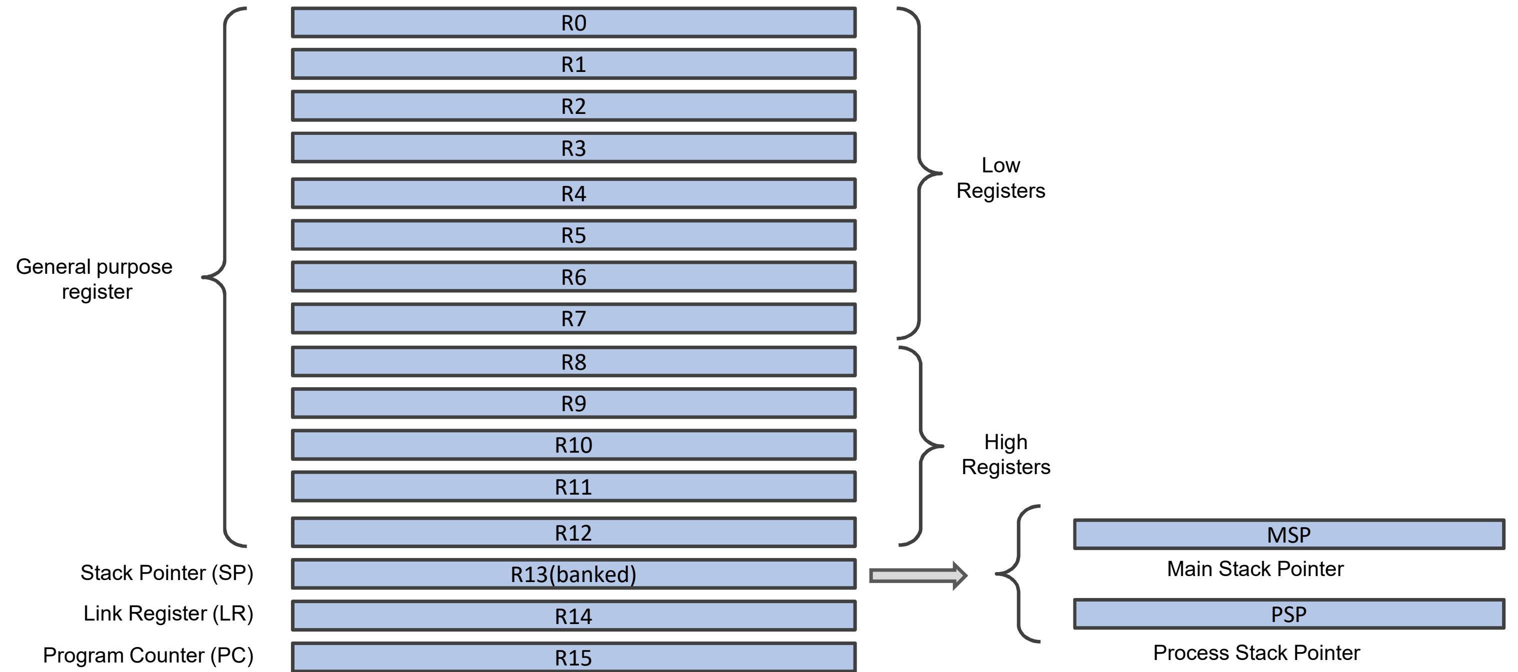


Cortex-M0+ Registers

- Processor registers
  - The internal registers are used to store and process temporary data within the processor core
  - All registers are inside the processor core, hence they can be accessed quickly
  - Load-store architecture
    - To process memory data, they have to be first loaded from memory to registers, processed inside the processor core using register data only, and then written back to memory if needed
- Cortex-M0+ registers
  - Register bank
    - Sixteen 32-bit registers (thirteen are used for general-purpose);
  - Special registers

Cortex-M0+ Registers

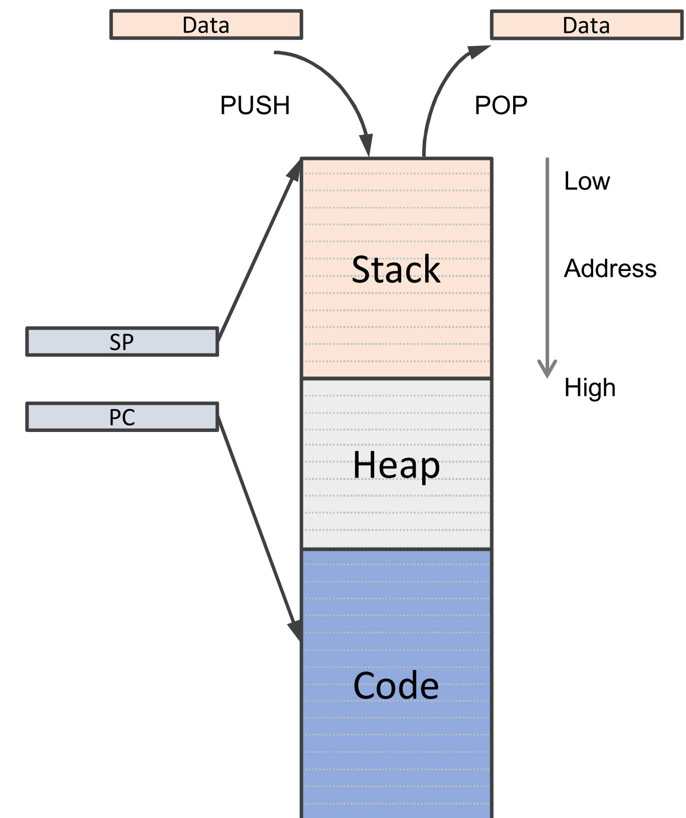
Register bank



Special registers

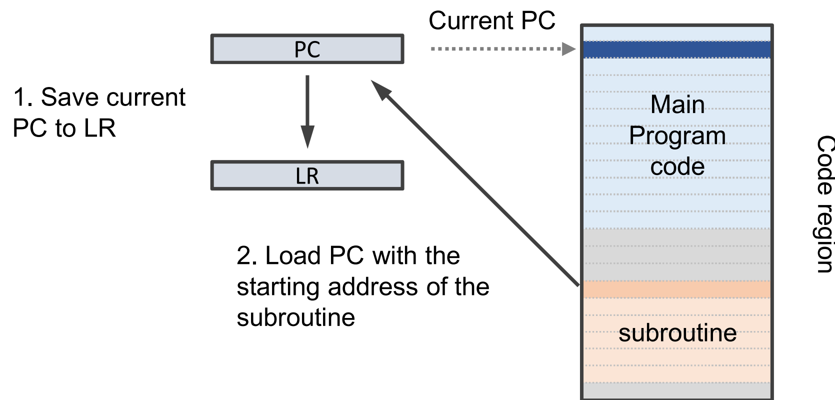
Cortex-M0+ Registers

- R0 – R12: general purpose registers
  - Low registers (R0 – R7) can be accessed by any instruction
  - High registers (R8 – R12) sometimes cannot be accessed e.g. by some Thumb (16-bit) instructions
- R13: Stack Pointer (SP)
  - Records the current address of the stack
  - Used for saving the context of a program while switching between tasks
  - Cortex-M0+ has two SPs: *Main SP*, used in applications that require privileged access e.g. OS kernel, and exception handlers, and *Process SP*, used in base-level application code (when not running an exception handler)
- R15: Program Counter (PC)
  - Records the address of the current instruction code
  - Automatically incremented by 4 at each operation (for 32-bit instruction code), except branching operations
  - A branching operation, such as function calls, will change the PC to a specific address, meanwhile it saves the current PC to the Link Register (LR)
  - On reset processor loads PC with the value of the reset vector, that is at address 0x00000004. Bit[0] of the value is loaded into the EPSR T-bit at reset and must be 1.

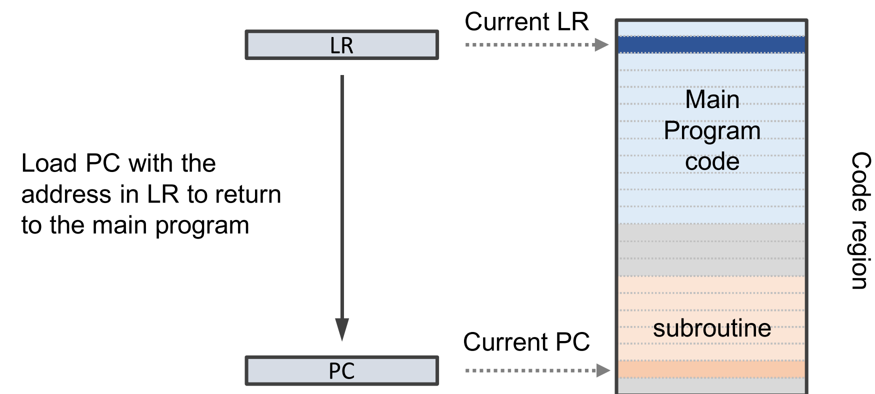


Cortex-M0+ Registers

- R14: Link Register (LR)
  - The LR is used to store the return address of a subroutine or a function call
  - The program counter (PC) will load the value from LR after a function is finished



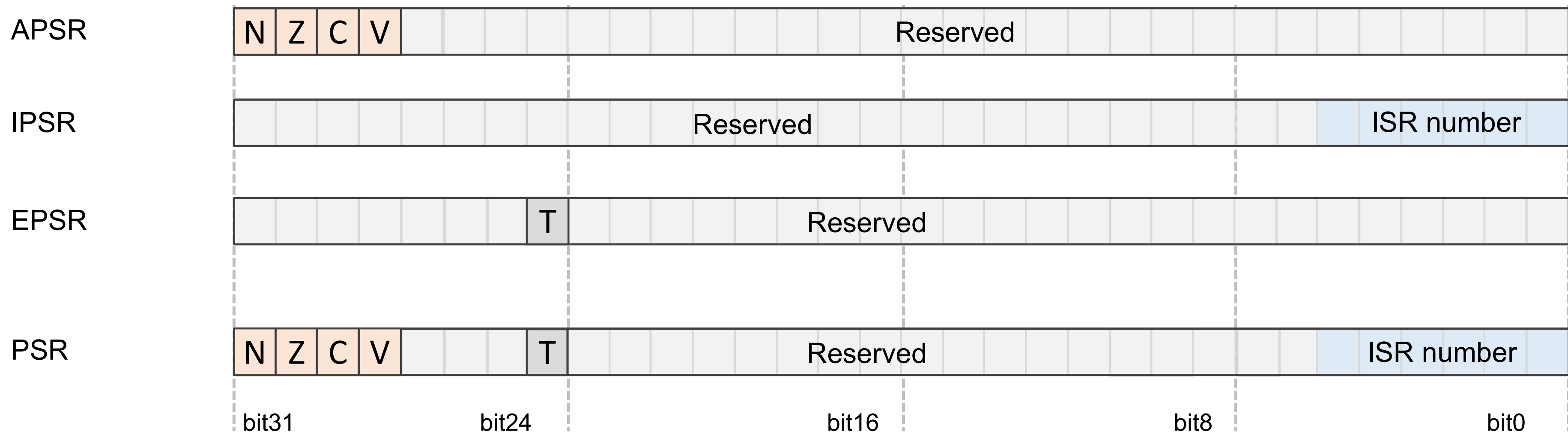
Call a subroutine



Return from a subroutine to the main program

Cortex-M0+ Registers

- Program Status Register (PSR)
  - Provides information about program execution and ALU flags
    - Application PSR (APSR) – condition code flag bit Negative, Zero, oVerflow, Carry
    - Interrupt PSR (IPSR) – holds exception number of currently executing ISR
    - Execution PSR (EPSR) – Thumb state



Cortex-M0+ Registers

- APSR
  - N: negative flag – set to one if the result from ALU is negative
  - Z: zero flag – set to one if the result from ALU is zero
  - C: carry flag – set to one if an unsigned overflow occurs
  - V: overflow flag – set to one if a signed overflow occurs
- IPSR
  - ISR number – current executing interrupt service routine number
- EPSR
  - T: Thumb state – always one since Cortex-M0+ only supports the Thumb state

Cortex-M0+ Registers

- Interrupt mask registers
  - Bit 0: PM Flag

    Set to one to prevent activation of all exceptions with configurable priority
  - Access using CPS, MSR and MSR instructions

    Use to prevent data race conditions with code needing atomicity
- CONTROL: special register
  - Bit 1: SPSEL Flag

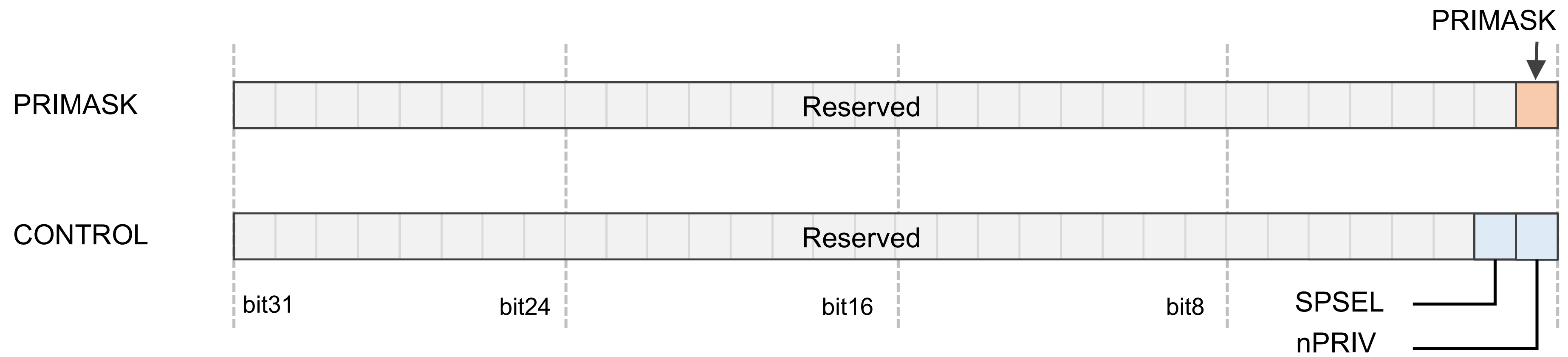
    Selects SP when in thread mode: MSP (0) or PSP (1)
    - Bit 0: nPRIV Flag

    Defines whether thread mode is privileged (0) or unprivileged (1)
    - With OS environment,

    Threads use PSP

    OS and exception handlers (ISRs) use MSP

Cortex-M0+ Registers



Useful Resources

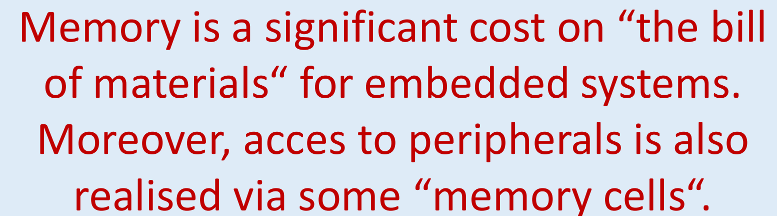
- Reference 1
  - Cortex-M0+ Technical Reference Manual:

  http://infocenter.arm.com/help/topic/com.arm.doc.ddi0484c/DDI0484C_cortex_m0p_r0p1_trm.pdf
- Reference 2
  - Cortex-M0+ Devices Generic User Guide:

  http://infocenter.arm.com/help/topic/com.arm.doc.dui0662b/DUI0662B_cortex_m0p_r0p1_dgug.pdf
- Reference 3
  - Cortex-M0+ Processor Overview:

  http://www.arm.com/products/processors/cortex-m/cortex-m0plus.php

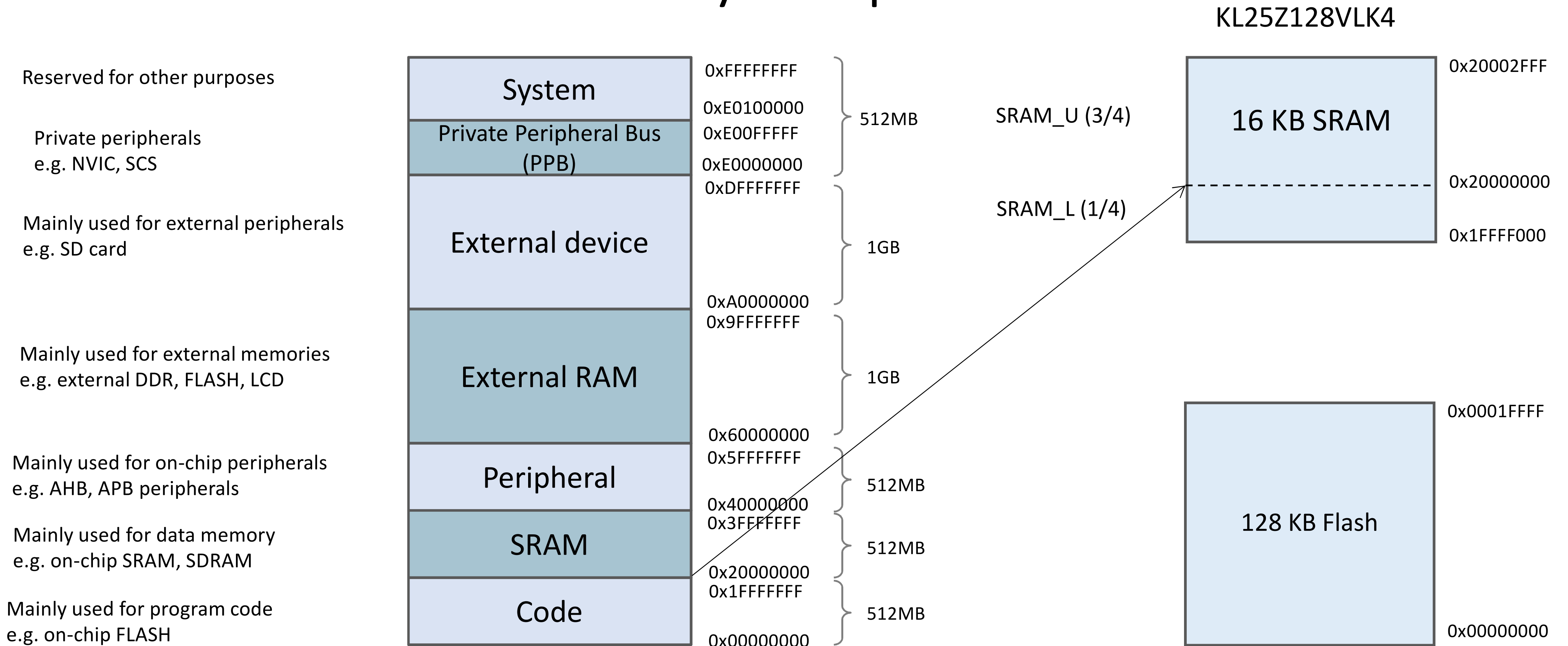
ARM Cortex-M0+ Processor Memory Map



Cortex-M0+ Memory Map

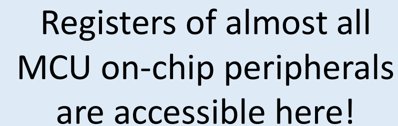
- The Cortex-M0+ processor has 4 GB of memory address space
  - Support for bit-band operation (detailed later)
- The 4GB memory space is architecturally defined as a number of regions
  - Each region is given for recommended usage
  - Easy for software programmer to port between different devices
- Nevertheless, despite of the default memory map, the actual usage of the memory map can also be flexibly defined by the user, except some fixed memory addresses, such as internal private peripheral bus

Cortex-M0+ Memory Map



Cortex-M0+ Memory Map

- Code Region
  - Primarily used to store program code
  - Can also be used for data memory (von Neumann!)
  - On-chip memory, such as on-chip FLASH
- SRAM Region
  - Primarily used to store data, such as heaps and stacks
  - Can also be used for program code (von Neumann!)
  - On-chip memory; despite its name "SRAM", the actual device could be SRAM, SDRAM or other types
- Peripheral Region
  - Primarily used for peripherals, such as Advanced High-performance Bus (AHB) or Advanced Peripheral Bus (APB) peripherals
  - On-chip peripherals



Cortex-M0+ Memory Map

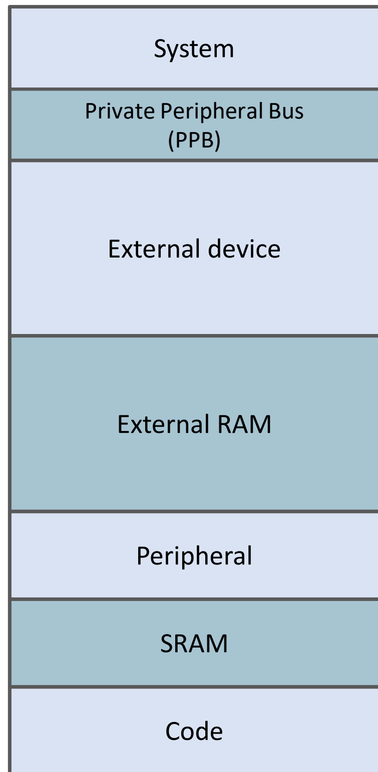
- External RAM Region
  - Primarily used to store large data blocks, or memory caches
  - Off-chip memory, slower than on-chip SRAM region

- External Device Region
  - Primarily used to map to external devices
  - Off-chip devices, such as SD card

- Internal Private Peripheral Bus (PPB)
  - Used inside the processor core for internal control
  - Within PPB, a special range of memory is defined as System Control Space (SCS)
  - The Nested Vectored Interrupt Controller (NVIC) is part of SCS

Memory Configuration Example

Memory Map:



Silicon Chip

Cortex-M0+

PPB

SCS

NVIC

Debug Ctrl

AHB bus

On-chip FLASH (Code Region)

On-chip SRAM (SRAM Region)

Timer

UART

GPIO

Peripheral Region

External memory interface (External RAM Region)

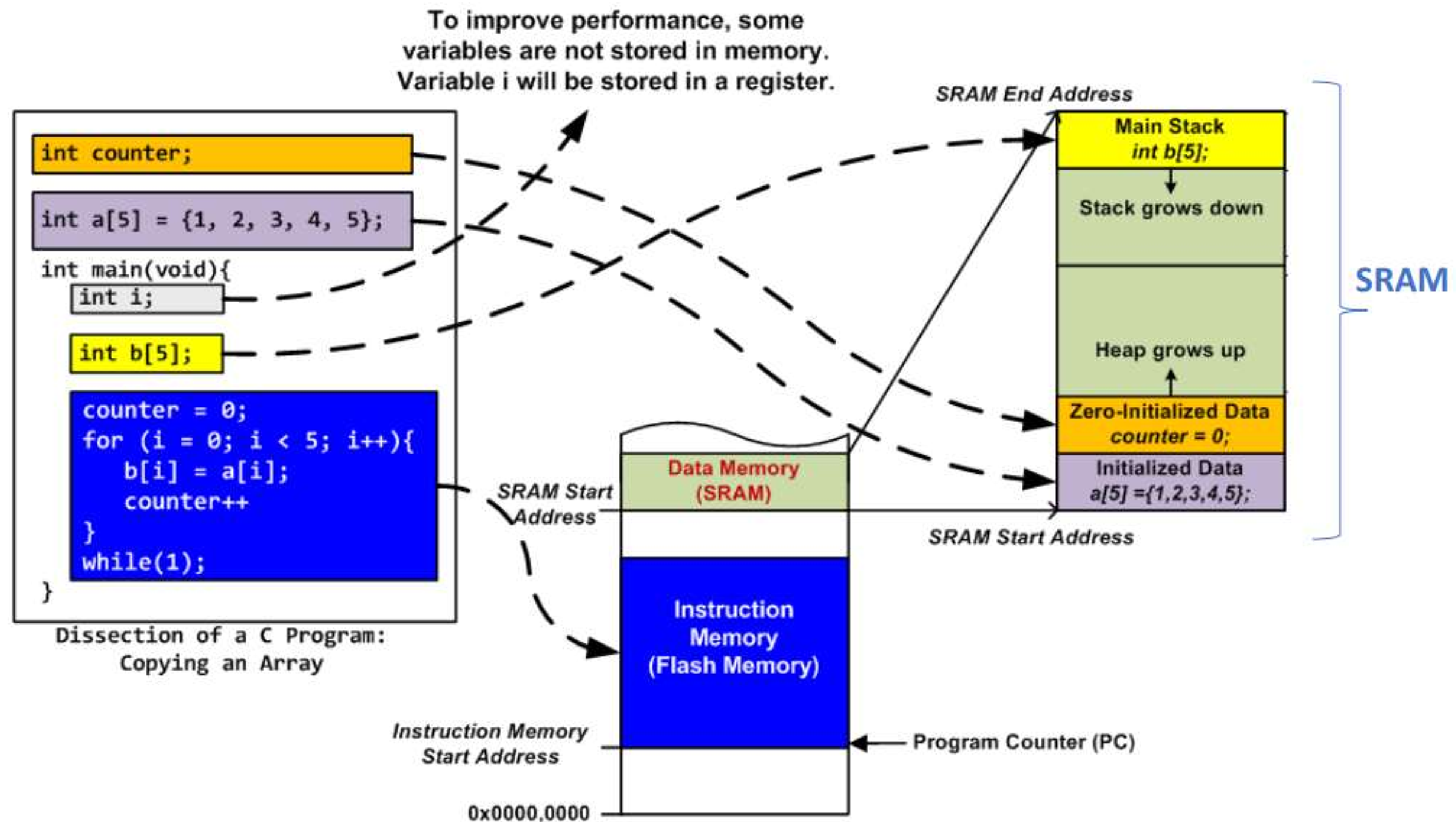
External device interface (External Device Region)

External SRAM, FLASH

External LCD

SD card

How the memory is typically used

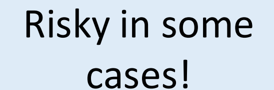


NXP Kinetis KL05 Memory Map:

System 32-bit Address Range	Destination Slave	Access
0x0000_0000–0x07FF_FFFF[1]	Program flash and read-only data (Includes exception vectors in first 196 bytes)	All masters
0x0800_0000–0x1FFF_FBFF	Reserved	—
0x1FFF_FC00-0x1FFF_FFFF[2]	SRAM_L: Lower SRAM	All masters
0x2000_0000-0x2000_0BFF[2]	SRAM_U: Upper SRAM	All masters
0x2000_0C00–0x3FFF_FFFF	Reserved	–
0x4000_0000–0x4007_FFFF	AIPS Peripherals	Cortex-M0+ core & DMA
0x4008_0000–0x400F_EFFF	Reserved	–
0x400F_F000–0x400F_FFFF	General purpose input/output (GPIO)	Cortex-M0+ core & DMA
0x4010_0000–0x43FF_FFFF	Reserved	–
0x4400_0000–0x5FFF_FFFF	Bit Manipulation Engine (BME) access to AIPS Peripherals for slots 0-127[3]	Cortex-M0+ core
0x6000_0000–0xDFFF_FFFF	Reserved	–
0xE000_0000–0xE00F_FFFF	Private Peripherals	Cortex-M0+ core
0xE010_0000–0xEFFF_FFFF	Reserved	–
0xF000_0000–0xF000_0FFF	Micro Trace Buffer (MTB) registers	Cortex-M0+ core
0xF000_1000–0xF000_1FFF	MTB Data Watchpoint and Trace (MTBDWT) registers	Cortex-M0+ core
0xF000_2000–0xF000_2FFF	ROM table	Cortex-M0+ core
0xF000_3000–0xF000_3FFF	Miscellaneous Control Module (MCM)	Cortex-M0+ core
0xF000_4000–0xF7FF_FFFF	Reserved	–
0xF800_0000–0xFFFF_FFFF	IOPORT: GPIO (single cycle)	Cortex-M0+ core

Bit-band Operations

- Bit-band operation allows a single load/store operation to access a single bit in the memory, for example, to change a single bit of one 32-bit data:
  - Normal operation without bit-band (read-modify-write)
    - Read the value of 32-bit data
    - Modify a single bit of the 32-bit value (keep other bits unchanged)
    - Write the value back to the address
  - Bit-band operation
    - Directly write a single bit (0 or 1) to the "bit-band alias address" of the data
- Bit-band alias address
  - Each bit-band alias address is mapped to a real data address
  - When writing to the bit-band alias address, only a single bit of the data will be changed



Bit-band Operation Example

- For example, in order to set bit[3] in word data in address 0x20000000:

```
;Read-Modify-Write Operation

LDR     R1, =0x20000000   ;Setup address
LDR     R0, [R1]          ;Read
ORR.W   R0, #0x8          ;Modify bit
STR     R0, [R1]          ;Write back
```

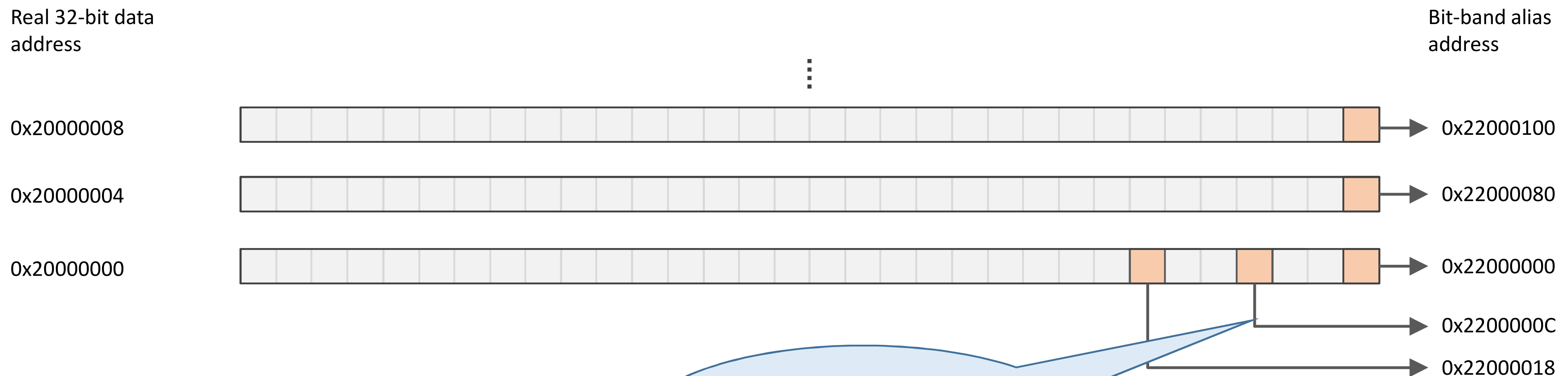
```
;Bit-band Operation

LDR     R1, =0x2200000C   ;Setup address
MOV     R0, #1            ;Load data
STR     R0, [R1]          ;Write
```

- Read-Modify-Write operation
  - Read the real data address (0x20000000)
  - Modify the desired bit (retain other bits unchanged)
  - Write the modified data back
- Bit-band operation
  - Directly set the bit by writing '1' to address 0x2200000C, which is the alias address of the fourth bit of the 32-bit data at 0x20000000
    - In effect, this single instruction is mapped to 2 bus transfers: read data from 0x20000000 to the buffer, and then write to 0x20000000 from the buffer with bit [3] set

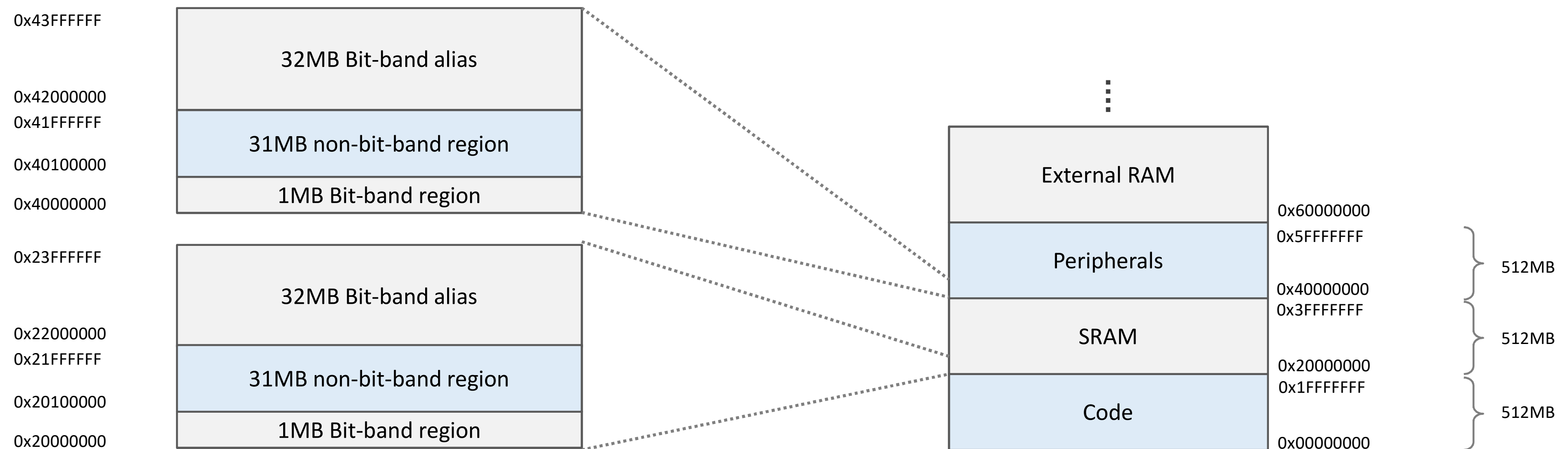
Bit-band Alias Address

- Each bit of the 32-bit data is one-to-one mapped to the bit-band alias address
  - For example, the fourth bit (bit [3]) of the data at 0x20000000 is mapped to the bit-band alias address at 0x2200000C
  - Hence, to set bit [3] of the data at 0x20000000, we only need to write '1' to address 0x2200000C
  - In Cortex-M0+, there are two pre-defined bit-band alias regions: one for SRAM region, and one for peripherals region



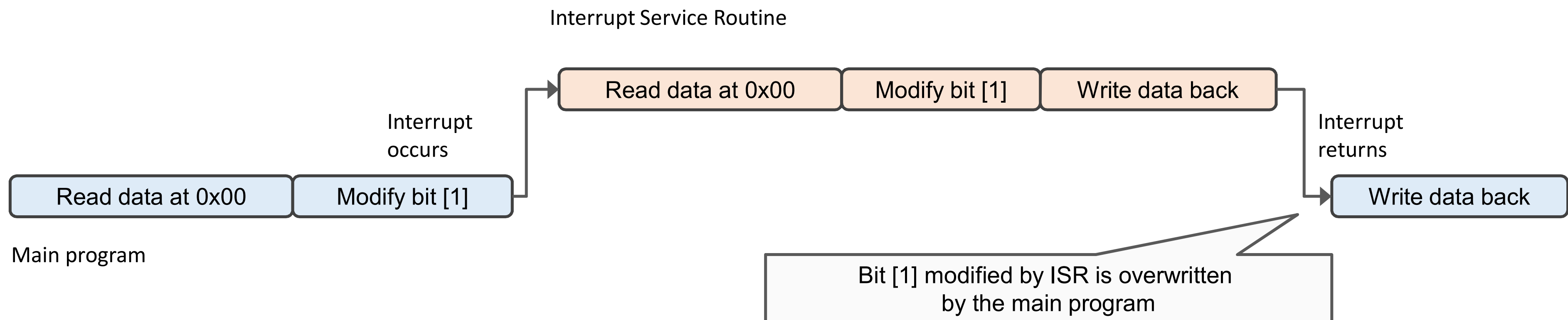
Bit-band Alias Address

- SRAM region
  - 32MB memory space (0x22000000 – 0x23FFFFFF) is used as the bit-band alias region for 1MB data (0x20000000 – 0x200FFFFF)
- Peripherals region
  - 32MB memory space (0x42000000 – 0x43FFFFFF) is used as the bit-band alias region for 1MB data (0x40000000 – 0x400FFFFF)



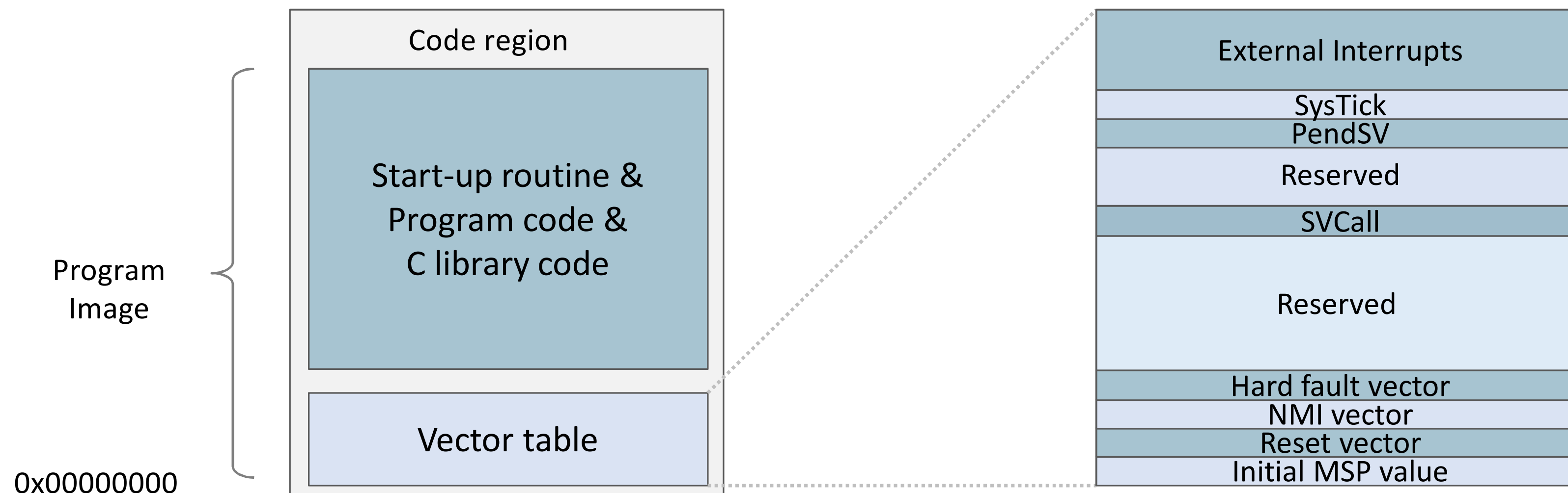
Benefits of Bit-Band Operations

- Faster bit operations
- Fewer instructions
- Atomic operation, avoid hazards
  - For example, if an interrupt is triggered and served during the Read-Modify-Write operations, and the interrupt service routine modifies the same data, a data conflict will occur



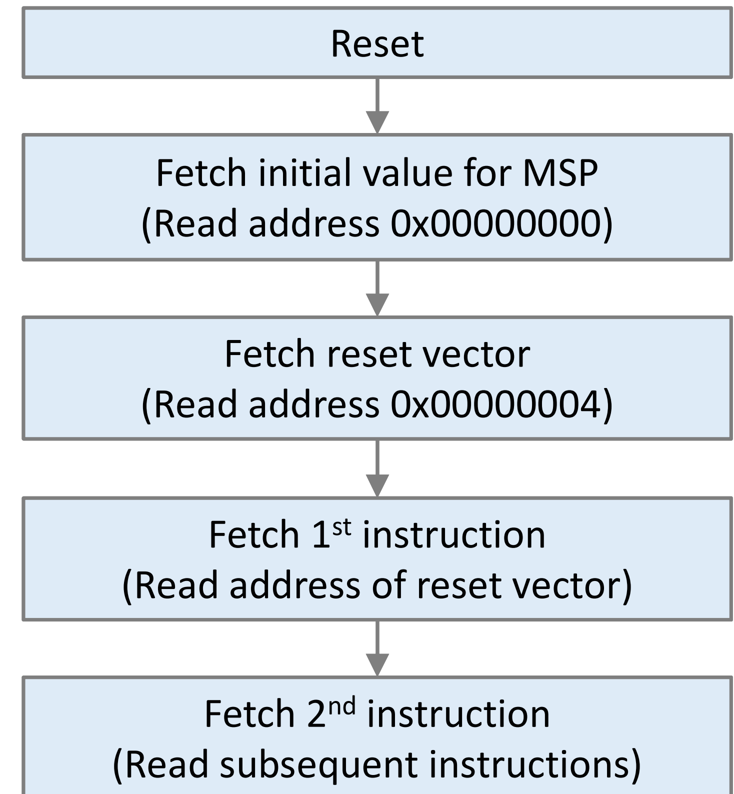
Cortex-M0+ Program Image

- The program image in Cortex-M0+ contains
  - Vector table -- includes the starting addresses of exceptions (vectors) and the value of the main stack point (MSP);
  - C start-up routine;
  - Program code – application code and data;
  - C library code – program codes for C library functions.



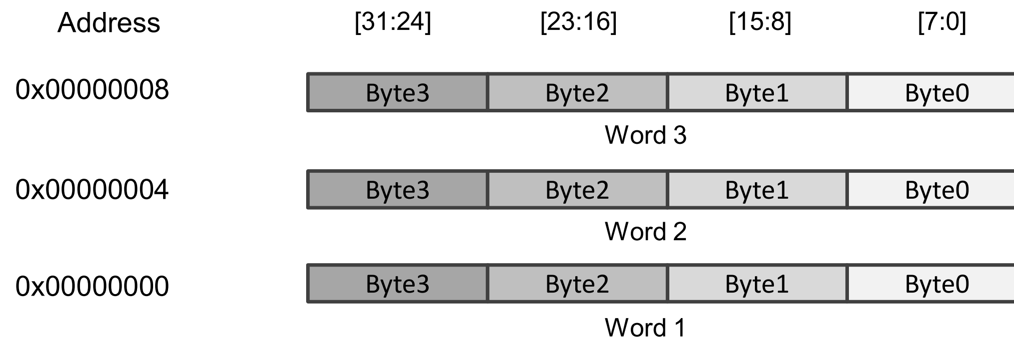
Cortex-M0+ Program Image

- After Reset, the processor:
    1. First reads the initial MSP value;
    2. Then reads the reset vector;
    3. Branches to the start of the programme execution address (reset handler);
    4. Subsequently executes program instructions

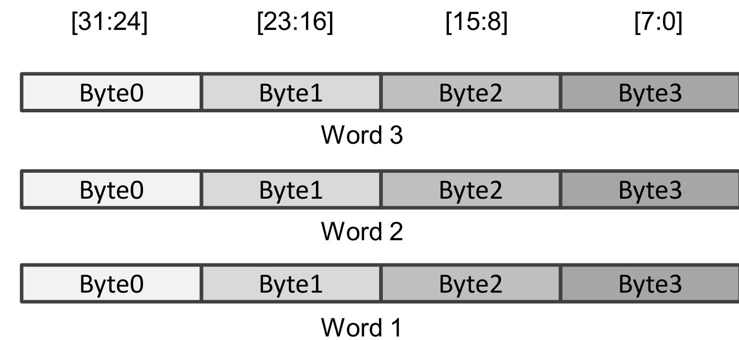


Cortex-M0+ Endianness

- Endian refers to the order of bytes stored in memory
  - Big endian: lowest byte of a word-size data is stored in bit 0 to bit 7
  - Big endian: lowest byte of a word-size data is stored in bit 24 to bit 31
- Cortex-M0+ supports both little endian and big endian
- However, Endianness only exists in the hardware level



Little endian 32-bit memory



Big endian 32-bit memory

Endianness

- In Little Endian forms, since lowest order byte is at offset 0 and is accessed first, instructions for accessing 1, 2, 4 or longer byte number proceed in exactly the same way for all formats. Also, because of 1:1 relationship between address offset and byte number (offset 0 is byte 0), multiple precision math routines are correspondingly easy to write.
- In Big Endian form, since the higher-ordered byte come first, it is easy to test whether the number is positive or negative by looking the byte at offset 0. Thus there is no need to receive the complete packet of bytes to know the sign information. The numbers are also stored in the order in which they are printed out, so binary to decimal routines are particularly efficient.

Endiannes doesn't matter on a single system. It matters only when two systems are trying to communicate (using network or shared memory). This is why modern MCU cores are Bi-Endian

ARM Cortex-M0+ Processor Instruction Set

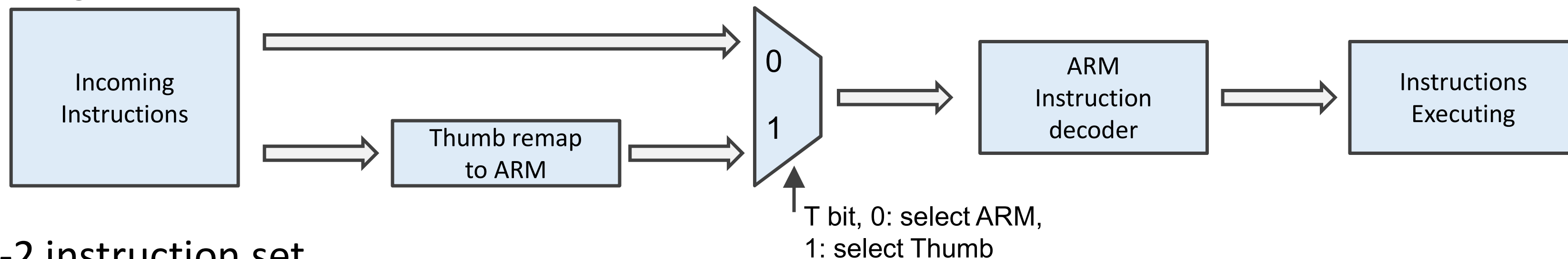
ARM and Thumb® Instruction Set

- Early ARM instruction set
  - 32-bit instruction set, called the ARM instructions
  - Powerful and good performance
  - Larger program memory compared to 8-bit and 16-bit processors
  - Larger power consumption

- Thumb-1 instruction set
  - 16-bit instruction set, first used in ARM7TDMI processor in 1995
  - Provides a subset of the ARM instructions, giving better code density compared to 32-bit RISC architecture
  - Code size is reduced by ~30%, but performance is also reduced by ~20%

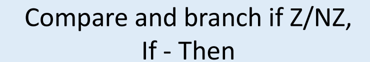
ARM and Thumb Instruction Set

- Mix of ARM and Thumb-1 Instruction sets
  - Benefit from both 32-bit ARM (high performance) and 16-bit Thumb-1 (high code density)
  - A multiplexer is used to switch between two states: ARM state (32-bit) and Thumb state (16-bit), which requires a switching overhead



- Thumb-2 instruction set
  - Consists of both 32-bit Thumb instructions and original 16-bit Thumb-1 instruction sets
  - Compared to 32-bit ARM instructions set, code size is reduced by ~26%, while keeping a similar performance
  - Capable of handling all processing requirements in one operation state

Cortex-M0+ Instruction Set



- ARMv6-M architecture profile
- Includes all of the 16-bit Thumb instruction from ARMv7-M, excluding CBZ, CBNZ and IT.
- The 32-bit Thumb instructions BL, DMB, DSB, ISB, MRS and MSR.
- Supports 32-bit Thumb-2 instructions
- Possible to handle all processing requirements in one operation state (Thumb state)
- Compared with traditional ARM processors (use state switching), advantages include:
  - No state switching overhead – both execution time and instruction space are saved
  - No need to separate ARM code and Thumb code source files, which makes the development and maintenance of software easier
  - Easier to get optimised efficiency and performance

Cortex-M0+ Instruction Set

- ARM assembly syntax:

  label

  mnemonic operand1, operand2, … ; Comments

  - Label is used as a reference to an address location;
  - Mnemonic is the name of the instruction;
  - Operand1 is the destination of the operation;
  - Operand2 is normally the source of the operation;
  - Comments are written after “ ; ”, which does not affect the program;
  - For example

    MOVS R3, #0x11 ;Set register R3 to 0x11
  - Note that the assembly code can be assembled by either ARM assembler (armasm) or assembly tools from a variety of vendors (e.g. GNU tool chain). When using GNU tool chain, the syntax for labels and comments is slightly different

Cortex-M0+ Instruction Set

Mnemonic	Operands	Brief description	Flags
ADCS	{Rd,} Rn, Rm	Add with Carry	N,Z,C,V
ADD{S}	{Rd,} Rn, <Rm|#imm>	Add	N,Z,C,V
ADR	Rd, label	PC-relative Address to Register	
ANDS	{Rd,} Rn, Rm	Bitwise AND	N,Z
ASRS	{Rd,} Rm, <Rs|#imm>	Arithmetic Shift Right	N,Z,C
B{cc}	Label	Branch {conditionally}	
BICS	{Rd,} Rn, Rm	Bit Clear	N,Z
BKPT	#imm	Breakpoint	
BL	label	Branch with Link	
BLX	Rm	Branch indirect with Link	
BX	Rm	Branch indirect	
CMN	Rn, Rm	Compare Negative	N,Z,C,V
CMP	Rn, <Rm| #imm>	Compare	N,Z,C,V
CPSID	i	Change Processor State, Disable Interrupts	

Cortex-M0+ Instruction Set

Mnemonic	Operands	Brief description	Flags
CPSIE	I	Change Processor State, Enable Interrupts	
DMB		Data Memory Barrier	
DSB		Data Synchronization Barrier	
EORS	{Rd,} Rn, Rm	Exclusive OR	N,Z
ISB		Instruction Synchronization Barrier	
LDM	Rn{!}, reglist	Load Multiple registers, increment after	
LDR	Rt, label	Load Register from PC-relative address	
LDR	Rt, [Rn, <Rm| #imm>]	Load Register with word	
LDRB	Rt, [Rn, <Rm| #imm>]	Load Register with Byte	
LDRH	Rt, [Rn, <Rm| #imm>]	Load Register with Halfword	
LDRSB	Rt, [Rn, <Rm| #imm>]	Load Register with Signed Byte	
LDRSH	Rt, [Rn, <Rm| #imm>]	Load Register with Signed Halfword	
LSLS	{Rd, } Rn, <Rs| #imm>	Logical Shift Left	N,Z,C
LSRS	{Rd, } Rn, <Rs| #imm>	Logical Shift Right	N,Z,C

Cortex-M0+ Instruction Set

Mnemonic	Operands	Brief description	Flags
MOV{S}	Rd, Rm	Move	N,Z
MRS	Rd, spec_reg	Move from Special Register to general register	
MSR	spec_reg, Rm	Move from general register to Special Register	N,Z,C,V
MULS	Rd, Rn, Rm	Multiply, 32-bit result	N,Z
MVNS	Rd, Rm	Bitwise NOT	N,Z
NOP		No Operation	
ORRS	{Rd,} Rn, Rm	Logical OR	N,Z
POP	reglist	Pop registers from stack	
PUSH	reglist	Push registers onto stack	
REV	Rd, Rm	Reverse byte order in a word	
REV16	Rd, Rm	Reverse byte order in each halfword	
REVSH	Rd, Rm	Reverse byte order in bottom halfword and sign extend	
RORS	{Rd, } Rn, Rs	Rotate Right	N,Z,C

Cortex-M0+ Instruction Set

Mnemonic	Operands	Brief description	Flags
RSBS	{Rd,} Rn, #0	Reverse Subtract	N,Z,C,V
SBCS	{Rd,} Rn, Rm	Subtract with Carry	N,Z,C,V
SEV		Send Event	
STM	Rn!, reglist	Store Multiple registers, increment after	
STR	Rt, [Rn, <Rm| #imm>]	Store Register as word	
STRB	Rt, [Rn, <Rm| #imm>]	Store Register as byte	
STRH	Rt, [Rn, <Rm| #imm>]	Store Register as Halfword	
SUB{S}	{Rd, } Rn, <Rm| #imm>	Subtract	N,Z,C,V
SVC	#imm	Supervisor Call	
SXTB	Rd, Rm	Sign extend byte	
SXTH	Rd, Rm	Sign extend halfword	
TST	Rn, Rm	Test	N,Z
UXTB	Rd, Rm	Zero extend a Byte	
UXTH	Rd, Rm	Zero extend a Halfword	

Cortex-M0+ Instruction Set

Mnemonic	Operands	Brief description	Flags
WFE		Wait For Event	
WFI		Wait For Interrupt	

Note: full explanation of each instruction can be found in Cortex-M0+ Devices’ Generic User Guide (Ref-3)

Cortex-M0+ Instruction Set

- Condition codes
  - Append to branch instruction (B) to make a conditional branch
  - Full ARM instructions (not Thumb or Thumb-2) support conditional execution of arbitrary instructions
  - Note: Carry bit = not-borrow for compares and subtractions

Mnemonic Extension	Meaning	Condition flags	Mnemonic Extension	Meaning	Condition flags
EQ	Equal	Z == 1	HI	Unsigned Higher	C == 1 and Z == 0
NE	Not Equal	Z == 0	LS	Unsigned lower or same	C == 0 or Z == 1
CS	Carry Set	C == 1	GE	Signed greater than or equal	N == V
CC	Carry Clear	C == 0	LT	Signed less than	N != V
MI	Minus, negative	N == 1	GT	Signed greater than	Z == 0 or N != V
PL	Plus, positive or zero	N == 0	None (AL)	Always (unconditional)	Any
VS	Overflow	V == 1			
VC	No Overflow	V == 0			

Code Example

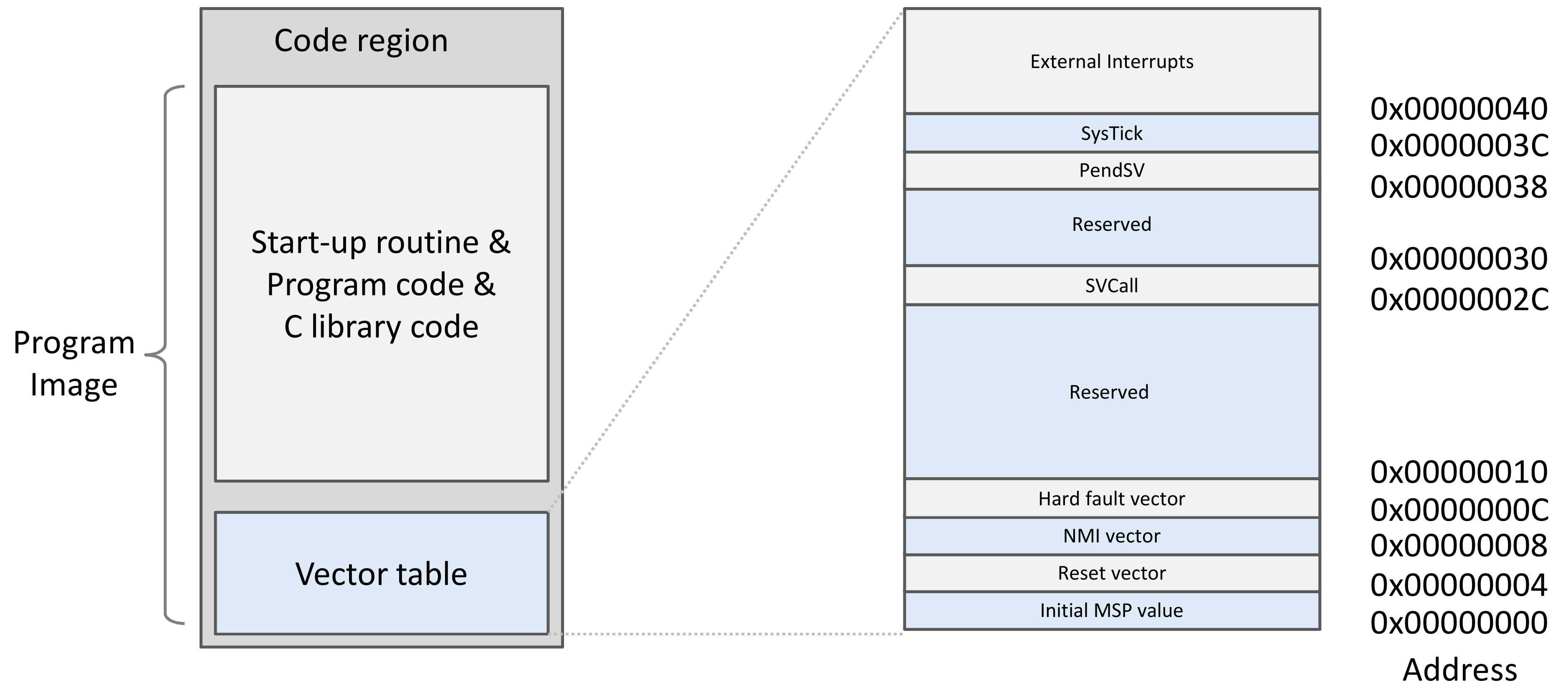
```
        AREA    subrout, CODE, READONLY     ; Name this block of code
        ENTRY                       ; Mark first instruction to execute
start   MOV     r0, #10             ; Set up parameters
        MOV     r1, #3
        BL      doadd               ; Call subroutine
stop    MOV     r0, #0x18           ; angel_SWIreason_ReportException
        LDR     r1, =0x20026        ; ADP_Stopped_ApplicationExit
        SVC     #0x123456           ; ARM semihosting (formerly SWI)
doadd   ADD     r0, r0, r1          ; Subroutine code
        BX      lr                  ; Return from subroutine
        END                         ; Mark end of file
```

C Language vs. Assembly Language

Language	Advantages	Disadvantage
C	Easy to learn	Limited or no direct access to core registers and stack
	Portable	No direct control over instruction sequence generation
	Easy handling of complex data structures	No direct control over stack usage
Assembly	Allow direct control to each instruction step and all memory	Take longer time to learn
	Allows direct access to instructions that cannot be generated with C	Difficult to manage data structure
		Less portable

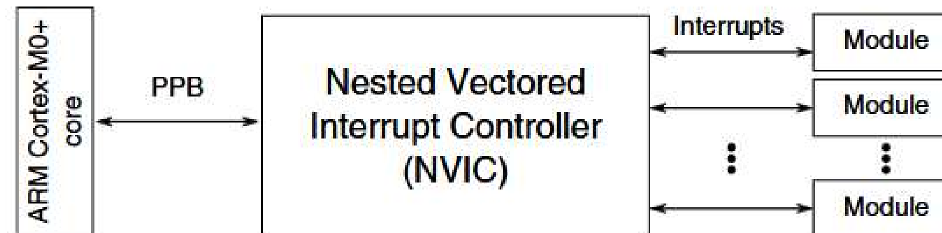
ARM Cortex-M0+ Interrupts

Cortex-M0+ Program Image



Address	Vector	IRQ[1]	NVIC IPR register number[2]	Source module	Source description
ARM Core System Handler Vectors					
0x0000_0000	0	—	—	ARM core	Initial Stack Pointer
0x0000_0004	1	—	—	ARM core	Initial Program Counter
0x0000_0008	2	—	—	ARM core	Non-maskable Interrupt (NMI)
0x0000_000C	3	—	—	ARM core	Hard Fault
0x0000_0010	4	—	—	—	—
0x0000_0014	5	—	—	—	—
0x0000_0018	6	—	—	—	—
0x0000_001C	7	—	—	—	—
0x0000_0020	8	—	—	—	—
0x0000_0024	9	—	—	—	—
0x0000_0028	10	—	—	—	—
0x0000_002C	11	—	—	ARM core	Supervisor call (SVCall)
0x0000_0030	12	—	—	—	—
0x0000_0034	13	—	—	—	—
0x0000_0038	14	—	—	ARM core	Pendable request for system service (PendableSrvReq)
0x0000_003C	15	—	—	ARM core	System tick timer (SysTick)

Peripheral Interrupts (non-core vectors)



3.3.2.1 Interrupt priority levels

This device supports 4 priority levels for interrupts. Therefore, in the NVIC each source in the IPR registers contains 2 bits. For example, IPR0 is shown below:

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	IRQ3		0	0	0	0	0	0	IRQ2		0	0	0	0	0	0	IRQ1		0	0	0	0	0	0	IRQ0		0	0	0	0	0	0
W																																

Non-Core Vectors					
0x0000_0040	16	0	0	DMA	DMA channel 0 transfer complete and error
0x0000_0044	17	1	0	DMA	DMA channel 1 transfer complete and error
0x0000_0048	18	2	0	DMA	DMA channel 2 transfer complete and error
0x0000_004C	19	3	0	DMA	DMA channel 3 transfer complete and error
0x0000_0050	20	4	1	—	—
0x0000_0054	21	5	1	FTFA	Command complete and read collision
0x0000_0058	22	6	1	PMC	Low-voltage detect, low-voltage warning
0x0000_005C	23	7	1	LLWU	Low Leakage Wakeup
0x0000_0060	24	8	2	I^2C0	
0x0000_0064	25	9	2	—	
0x0000_0068	26	10	2	SPI0	Single interrupt vector for all sources
0x0000_006C	27	11	2	—	
0x0000_0070	28	12	3	UART0	Status and error
0x0000_0074	29	13	3	—	
0x0000_0078	30	14	3	—	
0x0000_007C	31	15	3	ADC0	
0x0000_0080	32	16	4	CMP0	
0x0000_0084	33	17	4	TPM0	
0x0000_0088	34	18	4	TPM1	
0x0000_008C	35	19	4	—	
0x0000_0090	36	20	5	RTC	Alarm interrupt
0x0000_0094	37	21	5	RTC	Seconds interrupt

Address	Vector	IRQ[1]	NVIC IPR register number[2]	Source module	Source description
0x0000_0098	38	22	5	PIT	Single interrupt vector for all channels
0x0000_009C	39	23	5	—	—
0x0000_00A0	40	24	6	—	
0x0000_00A4	41	25	6	DAC0	
0x0000_00A8	42	26	6	TSI0	
0x0000_00AC	43	27	6	MCG	
0x0000_00B0	44	28	7	LPTMR0	
0x0000_00B4	45	29	7	—	
0x0000_00B8	46	30	7	Port control module	Pin detect (Port A)
0x0000_00BC	47	31	7	Port control module	Pin detect (Port B)

1. Indicates the NVIC's interrupt source number.
2. Indicates the NVIC's IPR register number used for this IRQ. The equation to calculate this value is: IRQ div 4