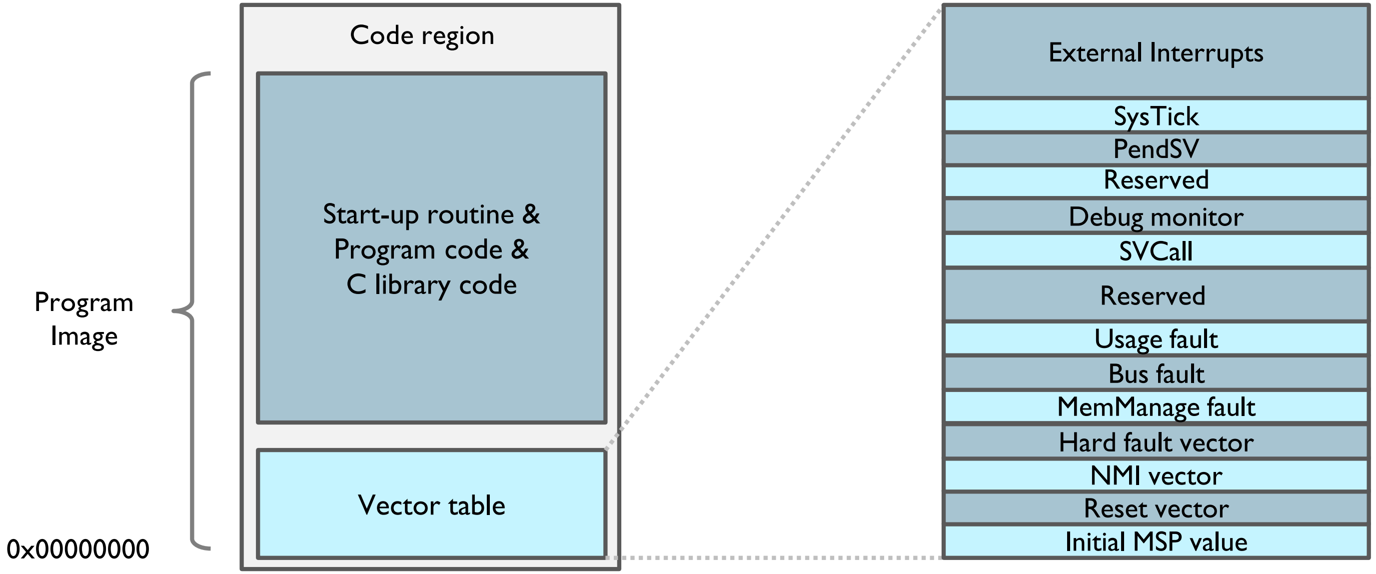
The ARM Cortex-M4 Processor Architecture

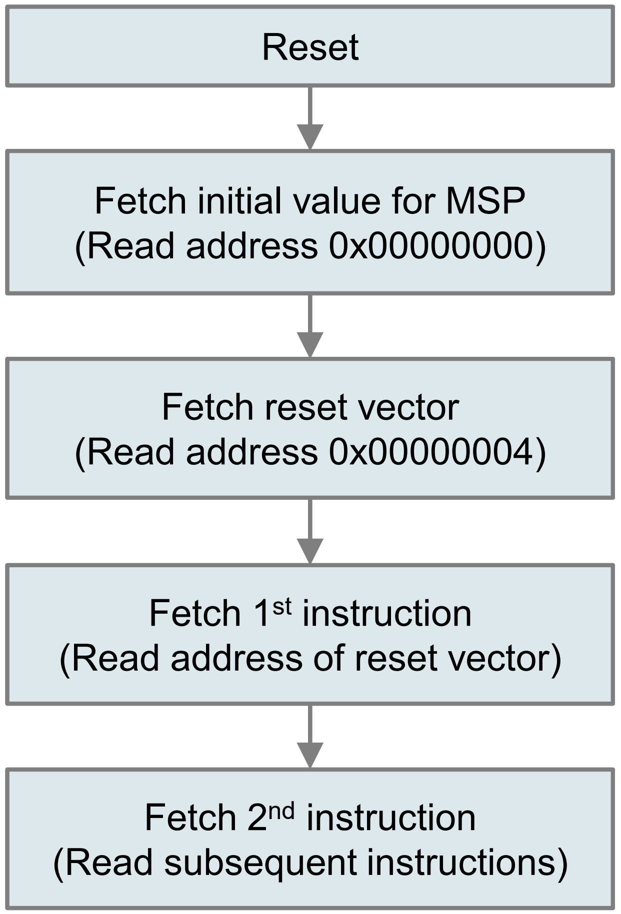
Cortex-M4 program image

- The program image in Cortex-M4 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-M4 program image

- After reset, the processor:
  - First reads the initial MSP value
  - Then reads the reset vector
  - Then branches to the start of the program execution address (reset handler)
  - Subsequently executes program instructions



Cortex-M4 endianness

- Endian refers to the order of bytes stored in memory
  - Little 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-M4 supports both little endian and big endian
- However, endianness only exists in the hardware level

Address	[31:24]	[23:16]	[15:8]	[7:0]	
0x00000008	Byte3	Byte2	Byte1	Byte0	Word 3
0x00000004	Byte3	Byte2	Byte1	Byte0	Word 2
0x00000000	Byte3	Byte2	Byte1	Byte0	Word 1

Little endian 32-bit memory

[31:24]	[23:16]	[15:8]	[7:0]	
Byte0	Byte1	Byte2	Byte3	Word 3
Byte0	Byte1	Byte2	Byte3	Word 2
Byte0	Byte1	Byte2	Byte3	Word 1

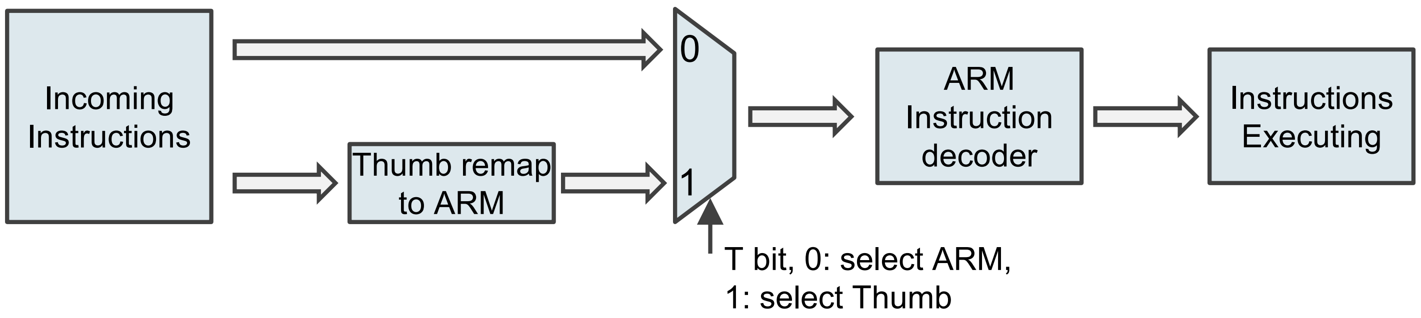
Big endian 32-bit memory

ARM Cortex-M4 processor instruction set 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 subset of ARM instructions, with 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%, with similar performance
  - Capable of handling all processing requirements in one operation state

Cortex-M4 instruction set

- Cortex-M4 processor
  - ARMv7-M architecture
  - Supports 32-bit Thumb-2 instructions
  - Can handle all processing requirements in one operation state (Thumb state)
  - Compared with traditional ARM processors (which 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 optimized efficiency and performance

Cortex-M4 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, e.g.:

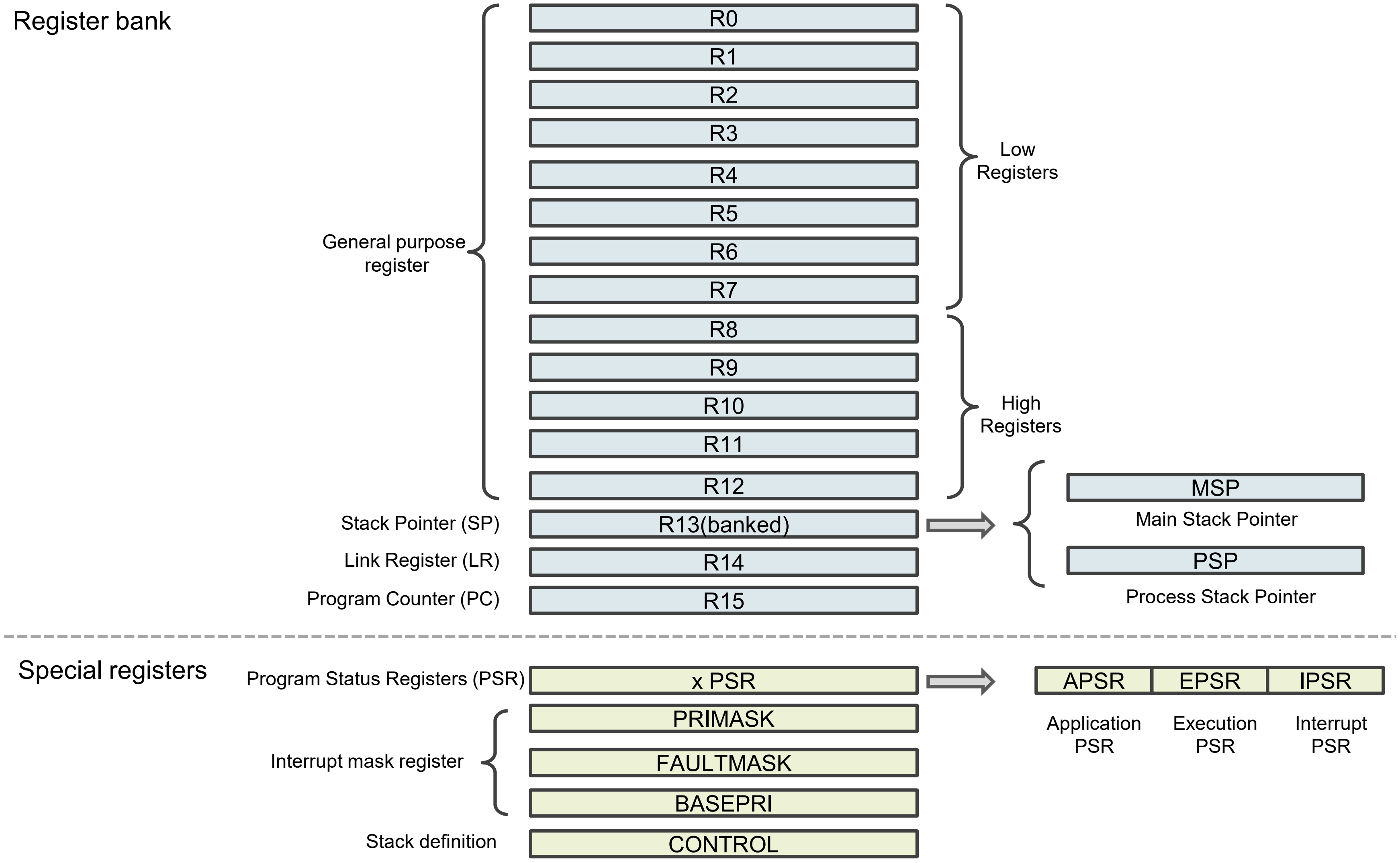
    MOVS R3, #0x11 ;Set register R3 to 0x11

  - 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 the GNU tool chain, the syntax for labels and comments is slightly different.

ARM Cortex-M4 processor registers

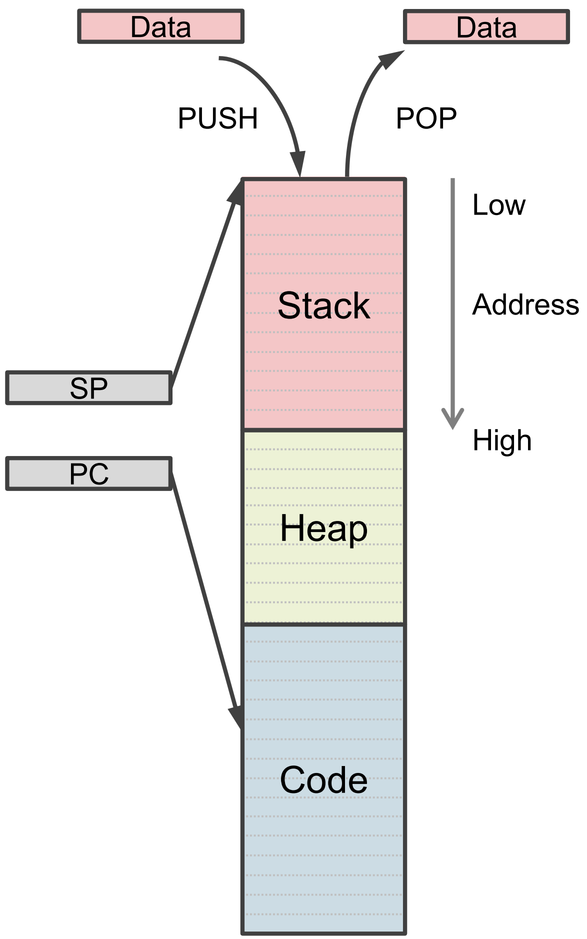
- Processor registers
  - The internal registers are used to store and process temporary data within the processor core
  - All registers are inside the processor core, hence they can be accessed quickly
  - Load-store architecture
    - To process memory data, they have to be first loaded from memory to registers, processed inside the processor core using register data only, and then written back to memory if needed
- Cortex-M4 registers
  - Register bank
    - Sixteen 32-bit registers (thirteen are used for general-purpose)
  - Special registers

Cortex-M4 registers



Cortex-M4 registers

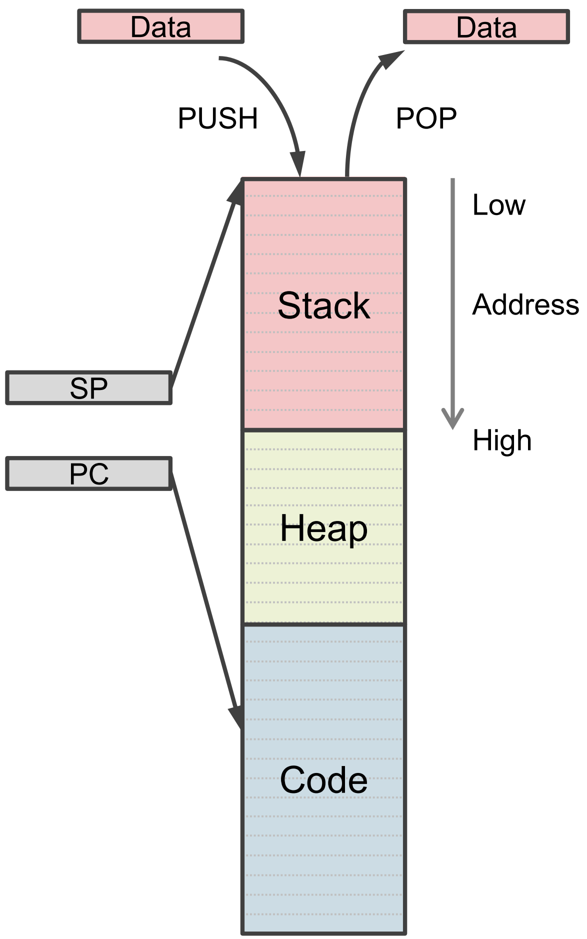
- R0 – R12: general purpose registers
  - Low registers (R0 – R7) can be accessed by any instruction
  - High registers (R8 – R12) sometimes cannot be accessed e.g. by some Thumb (16-bit) instructions
- R13: Stack Pointer (SP)
  - Records the current address of the stack
  - Used for saving the context of a program while switching between tasks
  - Cortex-M4 has two SPs: Main SP, used in applications that require privileged access e.g. OS kernel, and Process SP, used in base-level application code (when not running an exception handler)



ARM

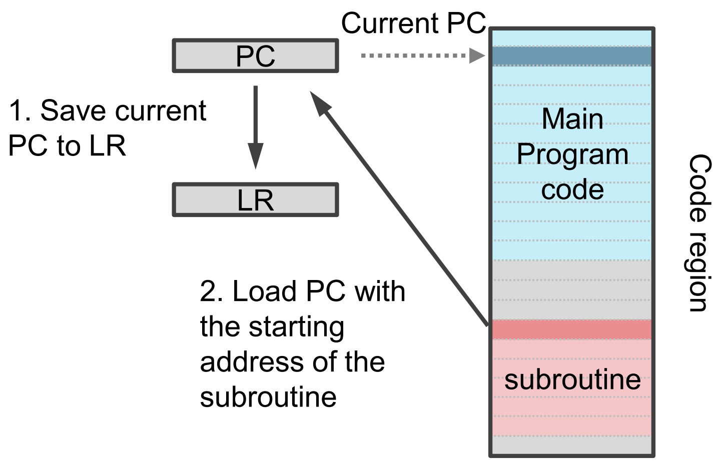
Cortex-M4 registers

- Program Counter (PC)
  - Records the address of the current instruction code
  - Automatically incremented by four 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, while saving the current PC to the Link Register (LR)

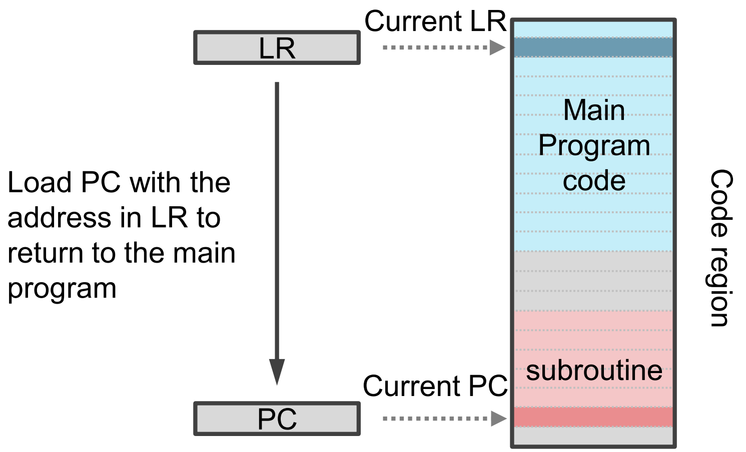


Cortex-M4 registers

- 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



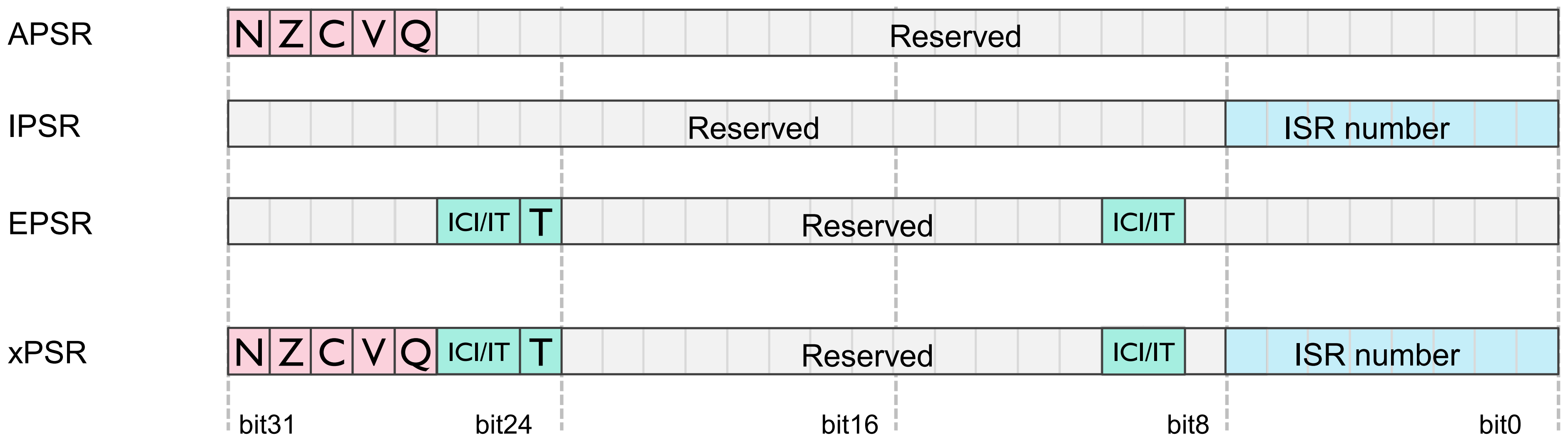
Call a subroutine



Return from subroutine to main program

Cortex-M4 registers

- xPSR, combined Program Status Register
  - Provides information about program execution and ALU flags
  - Application PSR (APSR)
  - Interrupt PSR (IPSR)
  - Execution PSR (EPSR)



Cortex-M4 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
  - Q: sticky saturation flag – set to one if saturation has occurred in saturating arithmetic instructions, or overflow has occurred in certain multiply instructions
- IPSR
  - ISR number – current executing interrupt service routine number
- EPSR
  - T: Thumb state – always one since Cortex-M4 only supports the Thumb state (more on processor states in the next module)
  - IC/IT: Interrupt-Continuable Instruction (ICI) bit, IF-THEN instruction status bit

Cortex-M4 registers

- Exception mask registers
  - 1-bit PRIMASK
    - If set to one, will block all the interrupts apart from non-maskable interrupt (NMI) and the hard fault exception
  - 1-bit FAULTMASK
    - If set to one, will block all the interrupts apart from NMI
  - BASEPRI
    - If set to one, will block all interrupts of the same or lower urgency (only allowing for interrupts with higher urgencies)
    - lower priority value, higher urgency
- CONTROL: special register
  - 1-bit stack definition
    - Set to one to use the process stack pointer (PSP)
    - Clear to zero to use the main stack pointer (MSP)

Cortex-M4 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, e.g.:

    MOVS R3, #0x11 ;Set register R3 to 0x11

  - 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 the GNU tool chain, the syntax for labels and comments is slightly different.

ARM Cortex M4 Instruction Set - Overview

Instructions supported by the Cortex-M4 processor can be grouped as follows:

- Memory access instructions
- General data processing instructions
- Multiply and divide instructions
- Saturating instructions
- Packing and unpacking instructions
- Bitfield instructions
- Branch and control instructions
- Miscellaneous instructions
- Floating-point instructions

Cortex-M4 instruction set

Mnemonic	Operands	Brief description	Flags
ADC, ADCS	{Rd,} Rn, Op2	Add with Carry	N,Z,C,V
ADD, ADDS	{Rd,} Rn, Op2	Add	N,Z,C,V
ADD, ADDW	{Rd,} Rn, #imm12	Add	N,Z,C,V
ADR	Rd, label	Load PC-relative Address	
AND, ANDS	{Rd,} Rn, Op2	Logical AND	N,Z,C
ASR, ASRS	Rd, Rm, <Rs|#n>	Arithmetic Shift Right	N,Z,C
B	label	Branch	
BFC	Rd, #lsb, #width	Bit Field Clear	
BFI	Rd, Rn, #lsb, #width	Bit Field Insert	
BIC, BICS	{Rd,} Rn, Op2	Bit Clear	N,Z,C
BKPT	#imm	Breakpoint	
BL	label	Branch with Link	
BLX	Rm	Branch indirect with Link	
BX	Rm	Branch indirect	

Cortex-M4 instruction set

Mnemonic	Operands	Brief description	Flags
CBNZ	Rn, label	Compare and Branch if Non Zero	
CBZ	Rn, label	Compare and Branch if Zero	
CLREX		Clear Exclusive	
CLZ	Rd, Rm	Count Leading Zeros	
CMN	Rn, Op2	Compare Negative	N,Z,C,V
CMP	Rn, Op2	Compare	N,Z,C,V
CPSID	i	Change Processor State, Disable Interrupts	
CPSIE	i	Change Processor State, Enable Interrupts	
DMB		Data Memory Barrier	
DSB		Data Synchronization Barrier	
EOR, EORS	{Rd,} Rn, Op2	Exclusive OR	N,Z,C
ISB	-	Instruction Synchronization Barrier	

Cortex-M4 instruction set

Mnemonic	Operands	Brief description	Flags
IT		If-Then condition block	
LDM	Rn{!}, reglist	Load Multiple registers, increment after	
LDMDB, LDMEA	Rn{!}, reglist	Load Multiple registers, decrement before	
LDMFD, LDMIA	Rn{!}, reglist	Load Multiple registers, increment after	
LDR	Rt, [Rn, #offset]	Load Register with word	
LDRB, LDRBT	Rt, [Rn, #offset]	Load Register with byte	
LDRD	Rt, Rt2, [Rn, #offset]	Load Register with two bytes	
LDREX	Rt, [Rn, #offset]	Load Register Exclusive	
LDREXB	Rt, [Rn]	Load Register Exclusive with Byte	
LDREXH	Rt, [Rn]	Load Register Exclusive with Halfword	
LDRH, LDRHT	Rt, [Rn, #offset]	Load Register with Halfword	

Cortex-M4 instruction set

Mnemonic	Operands	Brief description	Flags
LDRSB, LDRSBT	Rt, [Rn, #offset]	Load Register with Signed Byte	
LDRSH, LDRSHT	Rt, [Rn, #offset]	Load Register with Signed Halfword	
LDRT	Rt, [Rn, #offset]	Load Register with word	
LSL, LSLS	Rd, Rm, <Rs|#n>	Logical Shift Left	N,Z,C
LSR, LSRS	Rd, Rm, <Rs|#n>	Logical Shift Right	N,Z,C
MLA	Rd, Rn, Rm, Ra	Multiply with Accumulate, 32-bit result	
MLS	Rd, Rn, Rm, Ra	Multiply and Subtract, 32-bit result	
MOV, MOVS	Rd, Op2	Move	N,Z,C
MOVT	Rd, #imm16	Move Top	
MOVW, MOV	Rd, #imm16	Move 16-bit constant	N,Z,C
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

Cortex-M4 instruction set

Mnemonic	Operands	Brief description	Flags
MUL, MULS	{Rd,} Rn, Rm	Multiply, 32-bit result	N,Z
MVN, MVNS	Rd, Op2	Move NOT	N,Z,C
NOP		No Operation	
ORN, ORNS	{Rd,} Rn, Op2	Logical OR NOT	N,Z,C
ORR, ORRS	{Rd,} Rn, Op2	Logical OR	N,Z,C
PKHTB, PKHBT	{Rd, } Rn, Rm, Op2	Pack Halfword	
POP	reglist	Pop registers from stack	
PUSH	reglist	Push registers onto stack	
QADD	{Rd, } Rn, Rm	Saturating double and Add	Q
QADD16	{Rd, } Rn, Rm	Saturating Add 16	
QADD8	{Rd, } Rn, Rm	Saturating Add 8	

Cortex-M4 instruction set

Mnemonic	Operands	Brief description	Flags
QASX	{Rd,} Rn, Rm	Saturating Add and Subtract with Exchange	
QDADD	{Rd,} Rn, Rm	Saturating Add	Q
QDSUB	{Rd,} Rn, Rm	Saturating double and Subtract	Q
QSAX	{Rd,} Rn, Rm	Saturating Subtract and Add with Exchange	
QSUB	{Rd,} Rn, Rm	Saturating Subtract	Q
QSUB16	{Rd,} Rn, Rm	Saturating Subtract 16	
QSUB8	{Rd,} Rn, Rm	Saturating Subtract 8	
RBIT	Rd, Rn	Reverse Bits	
REV	Rd, Rn	Reverse byte order in a word	
REV16	Rd, Rn	Reverse byte order in each halfword	
REVSH	Rd, Rn	Reverse byte order in bottom halfword and sign extend	
ROR, RORS	Rd, Rm, <Rs|#n>	Rotate Right	N,Z,C

Cortex-M4 instruction set

Mnemonic	Operands	Brief description	Flags
RRX, RRXS	Rd, Rm	Rotate Right with Extend	N,Z,C
RSB, RSBS	{Rd,} Rn, Op2	Reverse Subtract	N,Z,C,V
SADD16	{Rd, } Rn, Rm	Signed Add 16	GE
SADD8	{Rd, } Rn, Rm	Signed Add 8	GE
SASX	{Rd, } Rn, Rm	Signed Add and Subtract with Exchange	GE
SBC, SBCS	{Rd,} Rn, Op2	Subtract with Carry	N,Z,C,V
SBFX	Rd, Rn, #lsb, #width	Signed Bit Field Extract	
SDIV	{Rd,} Rn, Rm	Signed Divide	
SEV		Send Event	
SHADD16	{Rd,} Rn, Rm	Signed Halving Add 16	
SHADD8	{Rd,} Rn, Rm	Signed Halving Add 8	
SHASX	{Rd,} Rn, Rm	Signed Halving Add and Subtract with Exchange	

Cortex-M4 instruction set

Mnemonic	Operands	Brief description	Flags
SHSAX	{Rd,} Rn, Rm	Signed Halving Subtract and Add with Exchange	
SHSUB16	{Rd,} Rn, Rm	Signed Halving Subtract 16	
SHSUB8	{Rd,} Rn, Rm	Signed Halving Subtract 8	
SMLABB, SMLABT, SMLATB, SMLATT	Rd, Rn, Rm, Ra	Signed Multiply Accumulate Long (halfwords)	Q
SMLAD, SMLADX	Rd, Rn, Rm, Ra	Signed Multiply Accumulate Dual	Q
SMLAL	RdLo, RdHi, Rn, Rm	Signed Multiply with Accumulate (32 x 32 + 64), 64-bit result	
SMLALBB, SMLALBT, SMLALTB, SMLALTT	RdLo, RdHi, Rn, Rm	Signed Multiply Accumulate Long, halfwords	
SMLALD, SMLALDX	RdLo, RdHi, Rn, Rm	Signed Multiply Accumulate Long Dual	
SMLAWB, SMLAWT	Rd, Rn, Rm, Ra	Signed Multiply Accumulate, word by halfword	Q
SMLSD	Rd, Rn, Rm, Ra	Signed Multiply Subtract Dual	Q
SMLSLD	RdLo, RdHi, Rn, Rm	Signed Multiply Subtract Long Dual	
SMMLA	Rd, Rn, Rm, Ra	Signed Most significant word Multiply Accumulate	

Cortex-M4 instruction set

Mnemonic	Operands	Brief description	Flags
SMMLS, SMMLR	Rd, Rn, Rm, Ra	Signed Most significant word Multiply Subtract	
SMMUL, SMMULR	{Rd,} Rn, Rm	Signed Most significant word Multiply	
SMUAD	{Rd,} Rn, Rm	Signed dual Multiply Add	Q
SMULBB, SMULBT SMULTB, SMULTT	{Rd,} Rn, Rm	Signed Multiply (halfwords)	
SMULL	RdLo, RdHi, Rn, Rm	Signed Multiply (32 x 32), 64-bit result	
SMULWB, SMULWT	{Rd,} Rn, Rm	Signed Multiply word by halfword	
SMUSD, SMUSDX	{Rd,} Rn, Rm	Signed dual Multiply Subtract	
SSAT	Rd, #n, Rm {,shift #s}	Signed Saturate	Q
SSAT16	Rd, #n, Rm	Signed Saturate 16	Q
SSAX	{Rd,} Rn, Rm	Signed Subtract and Add with Exchange	GE
SSUB16	{Rd,} Rn, Rm	Signed Subtract 16	
SSUB8	{Rd,} Rn, Rm	Signed Subtract 8	

Cortex-M4 instruction set

Mnemonic	Operands	Brief description	Flags
STM	Rn{!}, reglist	Store Multiple registers, increment after	
STMDB, STMEA	Rn{!}, reglist	Store Multiple registers, decrement before	
STMFD, STMIA	Rn{!}, reglist	Store Multiple registers, increment after	
STR	Rt, [Rn, #offset]	Store Register word	
STRB, STRBT	Rt, [Rn, #offset]	Store Register byte	
STRD	Rt, Rt2, [Rn, #offset]	Store Register two words	
STREX	Rd, Rt, [Rn, #offset]	Store Register Exclusive	
STREXB	Rd, Rt, [Rn]	Store Register Exclusive Byte	
STREXH	Rd, Rt, [Rn]	Store Register Exclusive Halfword	
STRH, STRHT	Rt, [Rn, #offset]	Store Register Halfword	
STRT	Rt, [Rn, #offset]	Store Register word	
SUB, SUBS	{Rd,} Rn, Op2	Subtract	N,Z,C,V

Cortex-M4 instruction set

Mnemonic	Operands	Brief description	Flags
SUB, SUBW	{Rd,} Rn, #imm12	Subtract	N,Z,C,V
SVC	#imm	Supervisor Call	
SXTAB	{Rd,} Rn, Rm,{,ROR #}	Extend 8 bits to 32 and add	
SXTAB16	{Rd,} Rn, Rm,{,ROR #}	Dual extend 8 bits to 16 and add	
SXTAH	{Rd,} Rn, Rm,{,ROR #}	Extend 16 bits to 32 and add	
SXTB16	{Rd,} Rm {,ROR #n}	Signed Extend Byte 16	
SXTB	{Rd,} Rm {,ROR #n}	Sign extend a byte	
SXTH	{Rd,} Rm {,ROR #n}	Sign extend a halfword	
TBB	[Rn, Rm]	Table Branch Byte	
TBH	[Rn, Rm, LSL #1]	Table Branch Halfword	
TEQ	Rn, Op2	Test Equivalence	N,Z,C
TST	Rn, Op2	Test	N,Z,C

Cortex-M4 instruction set

Mnemonic	Operands	Brief description	Flags
UADD16	{Rd,} Rn, Rm	Unsigned Add 16	GE
UADD8	{Rd,} Rn, Rm	Unsigned Add 8	GE
USAX	{Rd,} Rn, Rm	Unsigned Subtract and Add with Exchange	GE
UHADD16	{Rd,} Rn, Rm	Unsigned Halving Add 16	
UHADD8	{Rd,} Rn, Rm	Unsigned Halving Add 8	
UHASX	{Rd,} Rn, Rm	Unsigned Halving Add and Subtract with Exchange	
UHSAX	{Rd,} Rn, Rm	Unsigned Halving Subtract and Add with Exchange	
UHSUB16	{Rd,} Rn, Rm	Unsigned Halving Subtract 16	
UHSUB8	{Rd,} Rn, Rm	Unsigned Halving Subtract 8	
UBFX	Rd, Rn, #lsb, #width	Unsigned Bit Field Extract	
UDIV	{Rd,} Rn, Rm	Unsigned Divide	
UMAAL	RdLo, RdHi, Rn, Rm	Unsigned Multiply Accumulate Accumulate Long (32 x 32 + 32 +32), 64-bit result	

Cortex-M4 instruction set

Mnemonic	Operands	Brief description	Flags
UMLAL	RdLo, RdHi, Rn, Rm	Unsigned Multiply with Accumulate (32 x 32 + 64), 64-bit result	
UMULL	RdLo, RdHi, Rn, Rm	Unsigned Multiply (32 x 32), 64-bit result	
UQADD16	{Rd,} Rn, Rm	Unsigned Saturating Add 16	
UQADD8	{Rd,} Rn, Rm	Unsigned Saturating Add 8	
UQASX	{Rd,} Rn, Rm	Unsigned Saturating Add and Subtract with Exchange	
UQSAX	{Rd,} Rn, Rm	Unsigned Saturating Subtract and Add with Exchange	
UQSUB16	{Rd,} Rn, Rm	Unsigned Saturating Subtract 16	
UQSUB8	{Rd,} Rn, Rm	Unsigned Saturating Subtract 8	
USAD8	{Rd,} Rn, Rm	Unsigned Sum of Absolute Differences	
USADA8	{Rd,} Rn, Rm, Ra	Unsigned Sum of Absolute Differences and Accumulate	
USAT	Rd, #n, Rm {,shift #s}	Unsigned Saturate	Q
USAT16	Rd, #n, Rm	Unsigned Saturate 16	Q

Cortex-M4 instruction set

Mnemonic	Operands	Brief description	Flags
UASX	{Rd,} Rn, Rm	Unsigned Add and Subtract with Exchange	GE
USUB16	{Rd,} Rn, Rm	Unsigned Subtract 16	GE
USUB8	{Rd,} Rn, Rm	Unsigned Subtract 8	GE
UXTAB	{Rd,} Rn, Rm,{,ROR #}	Rotate, extend 8 bits to 32 and Add	
UXTAB16	{Rd,} Rn, Rm,{,ROR #}	Rotate, dual extend 8 bits to 16 and Add	
UXTAH	{Rd,} Rn, Rm,{,ROR #}	Rotate, unsigned extend and Add Halfword	
UXTB	{Rd,} Rm {,ROR #n}	Zero extend a Byte	
UXTB16	{Rd,} Rm {,ROR #n}	Unsigned Extend Byte 16	
UXTH	{Rd,} Rm {,ROR #n}	Zero extend a Halfword	
VABS.F32	Sd, Sm	Floating-point Absolute	
VADD.F32	{Sd,} Sn, Sm	Floating-point Add	
VCMP.F32	Sd, <Sm | #0.0>	Compare two floating-point registers, or one floating-point register and zero	FPSCR

Cortex-M4 instruction set

Mnemonic	Operands	Brief description	Flags
VCMPE.F32	Sd, <Sm | #0.0>	Compare two floating-point registers, or one floating-point register and zero with Invalid Operation check	FPSCR
VCVT.S32.F32	Sd, Sm	Convert between floating-point and integer	
VCVT.S16.F32	Sd, Sd, #fbits	Convert between floating-point and fixed point	
VCVTR.S32.F32	Sd, Sm	Convert between floating-point and integer with rounding	
VCVT<B|H>.F32.F16	Sd, Sm	Converts half-precision value to single-precision	
VCVTT<B|T>.F32.F16	Sd, Sm	Converts single-precision register to half-precision	
VDIV.F32	{Sd,} Sn, Sm	Floating-point Divide	
VFMA.F32	{Sd,} Sn, Sm	Floating-point Fused Multiply Accumulate	
VFNMA.F32	{Sd,} Sn, Sm	Floating-point Fused Negate Multiply Accumulate	
VFMS.F32	{Sd,} Sn, Sm	Floating-point Fused Multiply Subtract	
VFNMS.F32	{Sd,} Sn, Sm	Floating-point Fused Negate Multiply Subtract	
VLDM.F<32|64>	Rn{!}, list	Load Multiple extension registers	

Cortex-M4 instruction set

Mnemonic	Operands	Brief description	Flags
VLDR.F<32|64>	<Dd|Sd>, [Rn]	Load an extension register from memory	
VLMA.F32	{Sd,} Sn, Sm	Floating-point Multiply Accumulate	
VLMS.F32	{Sd,} Sn, Sm	Floating-point Multiply Subtract	
VMOV.F32	Sd, #imm	Floating-point Move immediate	
VMOV	Sd, Sm	Floating-point Move register	
VMOV	Sn, Rt	Copy ARM core register to single precision	
VMOV	Sm, Sm1, Rt, Rt2	Copy 2 ARM core registers to 2 single precision	
VMOV	Dd[x], Rt	Copy ARM core register to scalar	
VMOV	Rt, Dn[x]	Copy scalar to ARM core register	
VMRS	Rt, FPSCR	Move FPSCR to ARM core register or APSR	N,Z,C,V
VMSR	FPSCR, Rt	Move to FPSCR from ARM Core register	FPSCR
VMUL.F32	{Sd,} Sn, Sm	Floating-point Multiply	

Cortex-M4 instruction set

Mnemonic	Operands	Brief description	Flags
VNEG.F32	Sd, Sm	Floating-point Negate	
VNMLA.F32	Sd, Sn, Sm	Floating-point Multiply and Add	
VNMLS.F32	Sd, Sn, Sm	Floating-point Multiply and Subtract	
VNMUL	{Sd,} Sn, Sm	Floating-point Multiply	
VPOP	list	Pop extension registers	
VPUSH	list	Push extension registers	
VSQRT.F32	Sd, Sm	Calculates floating-point Square Root	
VSTM	Rn{!}, list	Floating-point register Store Multiple	
VSTR.F<32|64>	Sd, [Rn]	Stores an extension register to memory	
VSUB.F<32|64>	{Sd,} Sn, Sm	Floating-point Subtract	
WFE		Wait For Event	
WFI		Wait For Interrupt	

Note: full explanation of each instruction can be found in Cortex-M4 Devices' Generic User Guide (Ref-4)

Cortex-M4 instruction set

- Cortex-M4 suffix
  - Some instructions can be followed by suffixes to update processor flags or execute the instruction on a certain condition

Suffix	Description	Example	Example explanation
S	Update APSR (flags)	ADDS R1, #0x21	Add 0x21 to R1 and update APSR
EQ, NE, CS, CC, MI, PL, VS, VC, HI, LS, GE, LT, GT, LE	Condition execution e.g. EQ= equal, NE= not equal, LT= less than	BNE label	Branch to the label if not equal

Data insertion and alignment

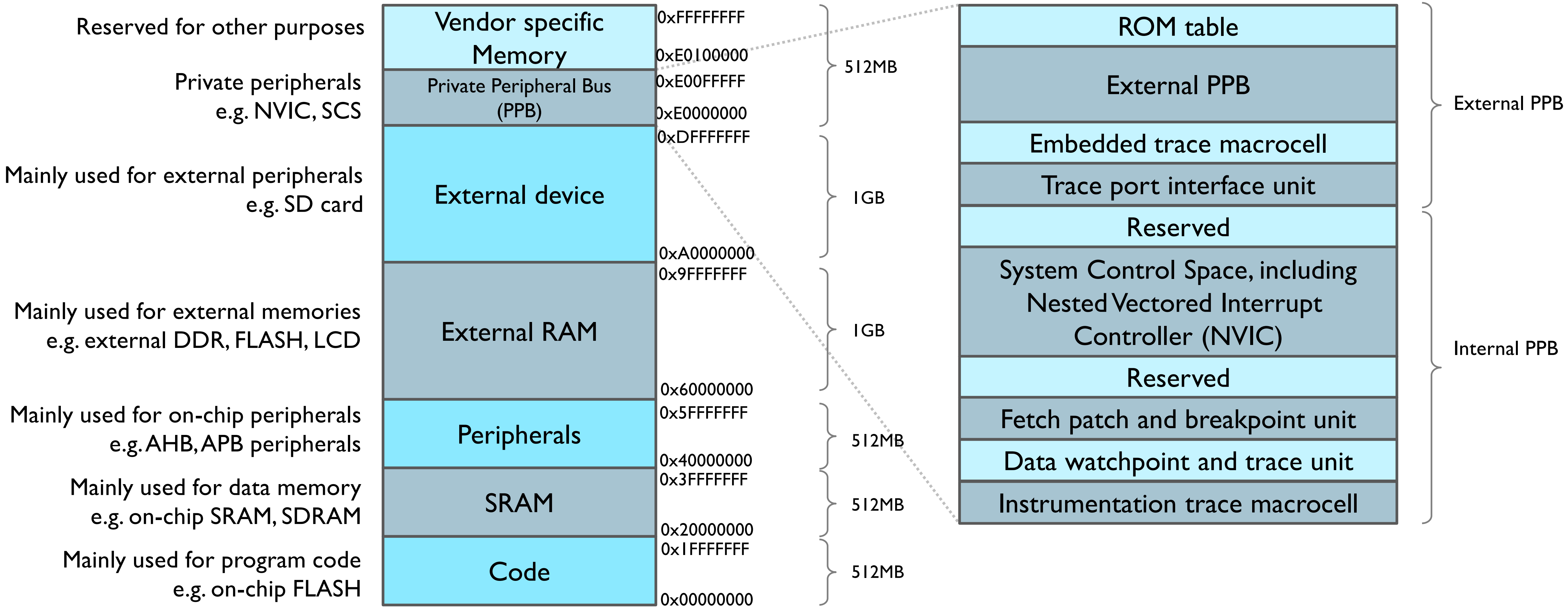
- Insert data inside programs
  - DCD: insert a word-size data
  - DCB: insert a byte-size data
  - ALIGN:
  - used before inserting a word-size data
  - Uses a number to determine the alignment size
- For example:

```
...
ALIGN           4                    ; Align to a word boundary
MY_DATA         DCD    0x12345678    ; Insert a word-size data
MY_STRINGDCB    "Hello",      0      ; Null terminated string
...
```

ARM Cortex-M4 memory map

- The Cortex-M4 processor has 4 GB of memory address space
  - Support for bit-band operation (detailed later)
- The 4GB memory space is architecturally defined with a number of regions
  - Each region is designed for particular recommended uses
  - Easy for software programmer to port between different devices
- Note that, despite the default definitions, the actual usage of the memory map can also be flexibly defined by the user, apart from some fixed memory addresses such as the internal private peripheral bus.

ARM Cortex-M4 memory map



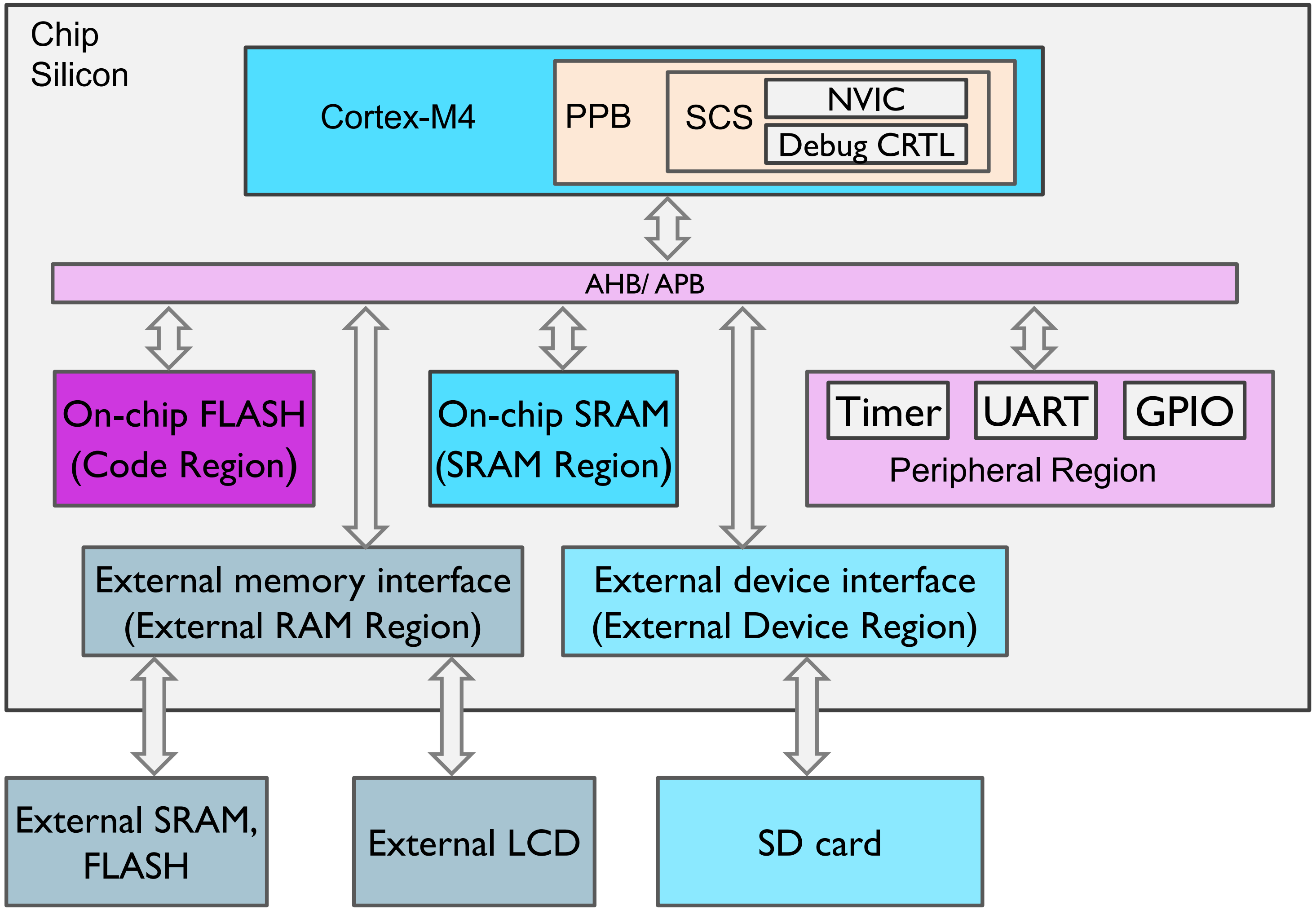
ARM Cortex-M4 memory map

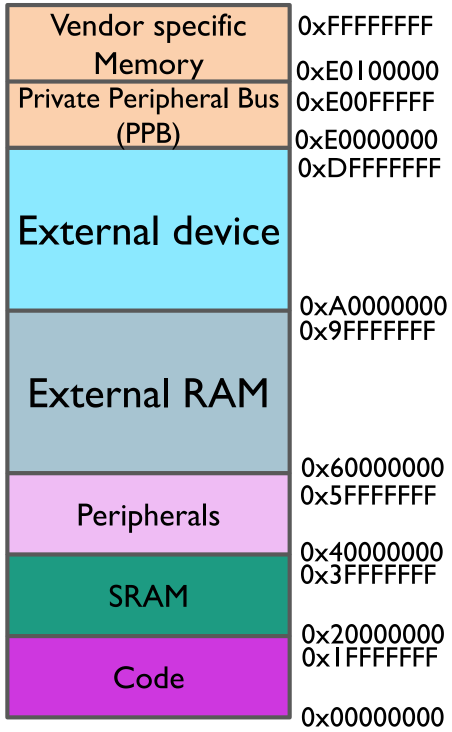
- Code region
  - Primarily used to store program code
  - Can also be used for data memory
  - On-chip memory, such as on-chip FLASH
- SRAM region
  - Primarily used to store data, such as heaps and stacks
  - Can also be used for program code
  - On-chip memory; despite its name "SRAM", the actual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

ARM Cortex-M4 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
- Private Peripheral Bus (PPB)
  - Provides access to internal and external processor resources

Cortex-M4 memory map example





ARM

Bit-band operations

- Bit-band operations allow 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:

```
...
LDR R1, =0x20000000 ;Setup address
STR R2,[R1]         ;Write back 0x79
...
```

Interrupt!

Return!

```
;Read-Modify-Write Operation

LDR   R1, =0x20000000 ;Setup address
LDR   R0, [R1]        ;Read 0x21


ORR.W R0, #0x8        ;Set bit[3]
STR   R0, [R1]        ;Write back 0x29
```

- Read-modify-write operation
  - Reads the data (0x21) from the address 0x20000000
  - The interrupt changes the data of 0x20000000 address to 0x79 and then returns
  - Writes back the modified old data back
  - 0x79 has been lost!

Bit-band operation example

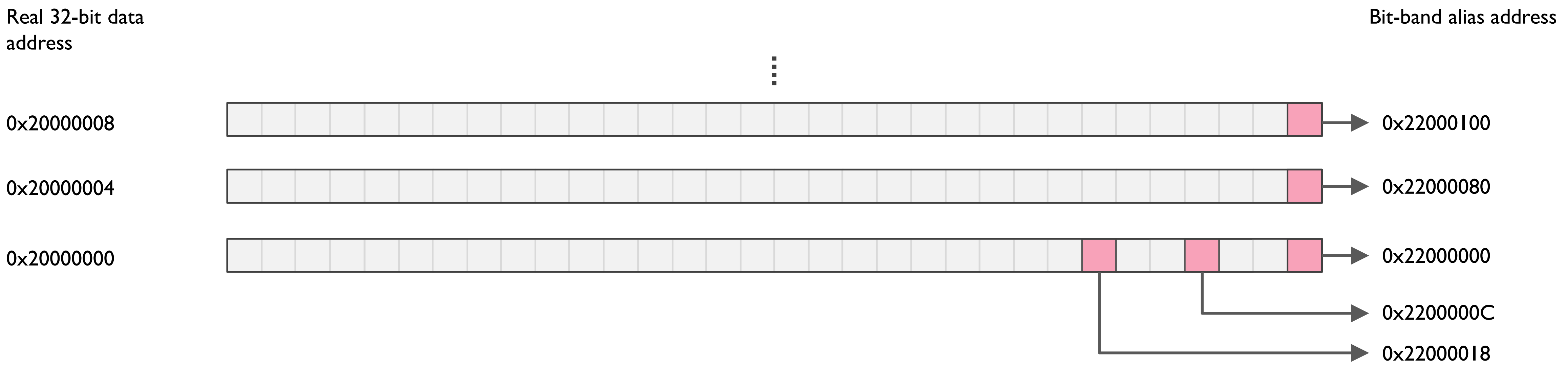
```
;Bit-band Operation

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

- 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

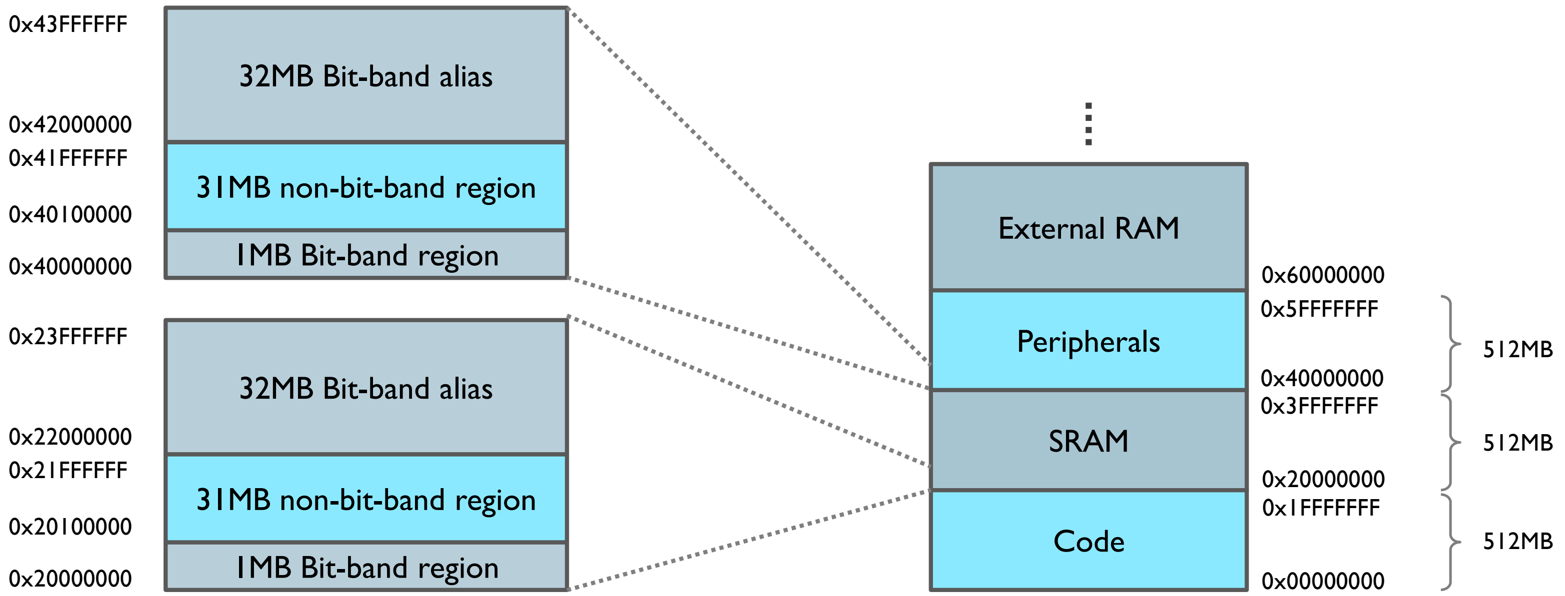
Bit-band alias address

- 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-M4, there are two pre-defined bit-band alias regions: one for SRAM region, and one for peripherals region



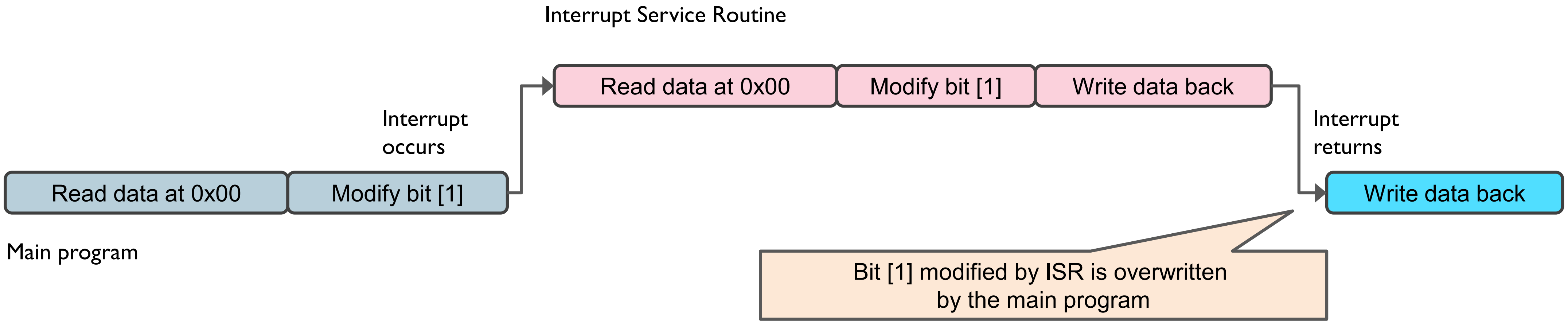
Bit-band alias address

- 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



Useful resources

- Architecture Reference Manual:

  http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0403c/index.html

- Cortex-M4 Technical Reference Manual:

  http://infocenter.arm.com/help/topic/com.arm.doc.ddi0439d/DDI0439D_cortex_m4_processor_r0p1_trm.pdf

- Cortex-M4 Devices Generic User Guide:

  http://infocenter.arm.com/help/topic/com.arm.doc.dui0553a/DUI0553A_cortex_m4_dgug.pdf