上海交通大學

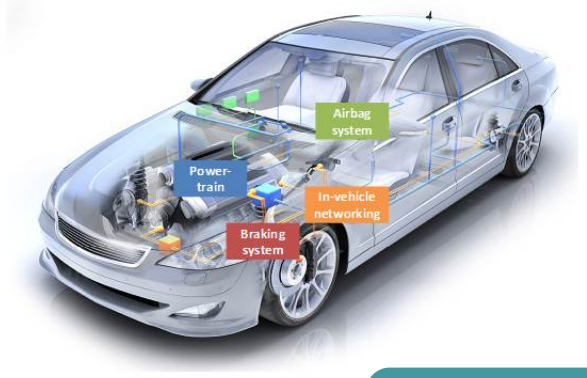
The Cortex-M3/M4 Processor Basics

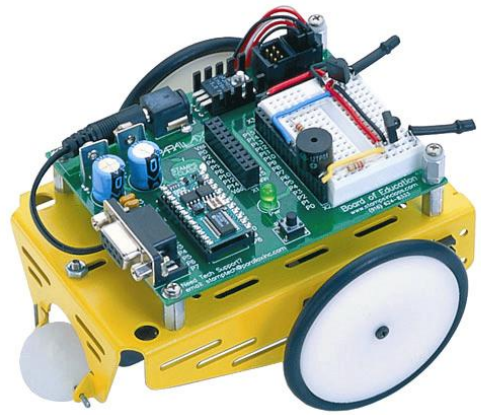
Refer to Chapter 1, 3, and 4 in the reference book

"The Definitive Guide to ARM Cortex-M3 and Cortex-M4 Processors"

Embedded System?

An **embedded system** is a computer system designed to perform one or a few dedicated functions often with constraints, e.g., in real time, with low power or limited memory, …





Embedded systems	General-purpose computers
A few dedicated functions	Flexible to meet a wide range of end-user needs
Under constrained conditions	No such rigid constraints

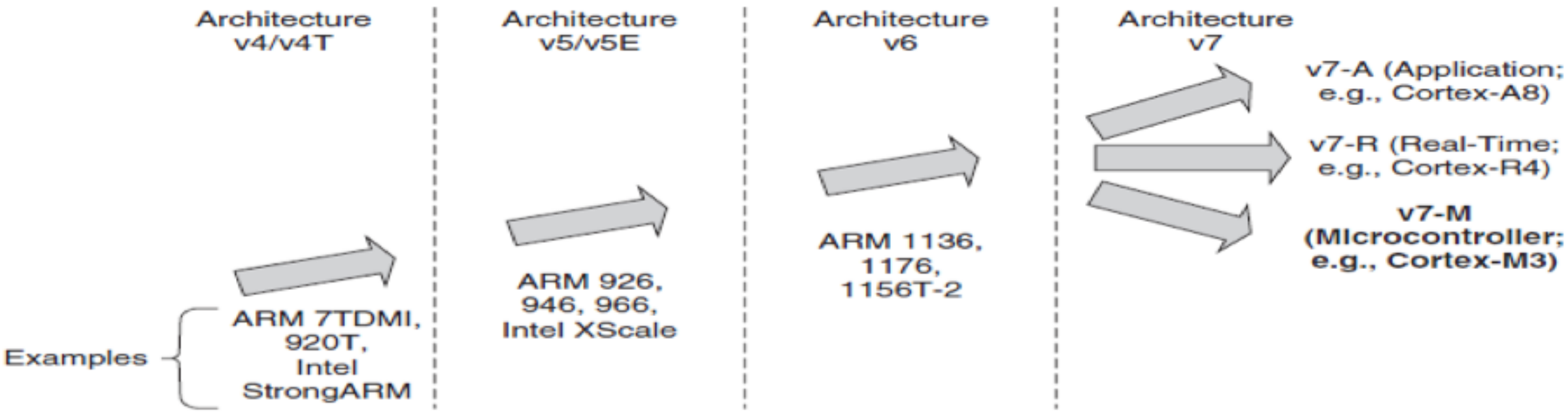
History about ARM Processers

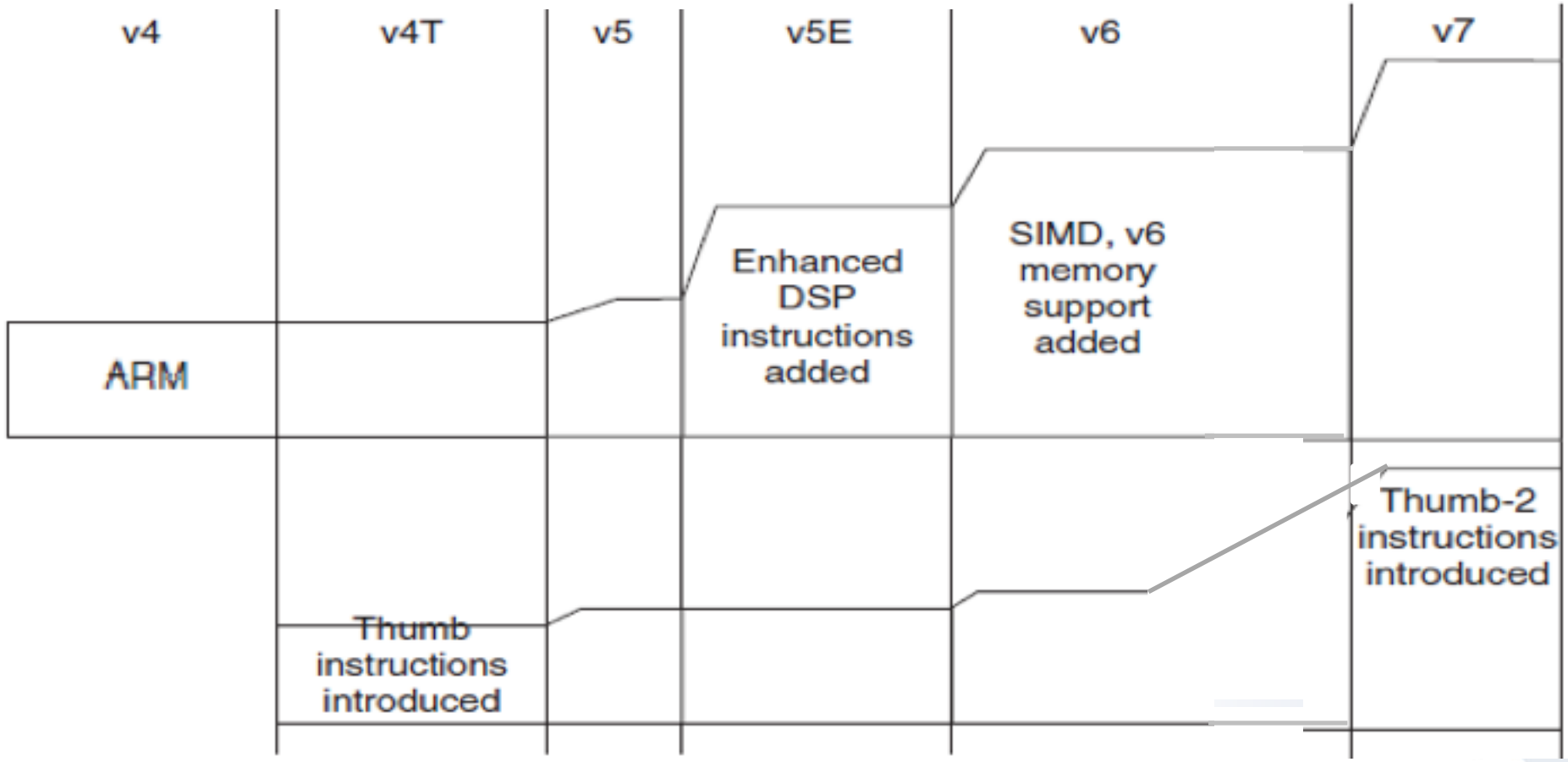
ARM company is known as the *Advanced RISC Machine*, born in 1990

- ARM does not manufacture processors or sell chips directly. They license their processor design to semiconductor companies, known as **intellectual property (IP) licensing**.

- Make over 2 billion ARM processors being shipped each year possible
- As of 2009, ARM processors account for approximately 90% of all embedded 32-bit RISC processors

History about ARM Processers



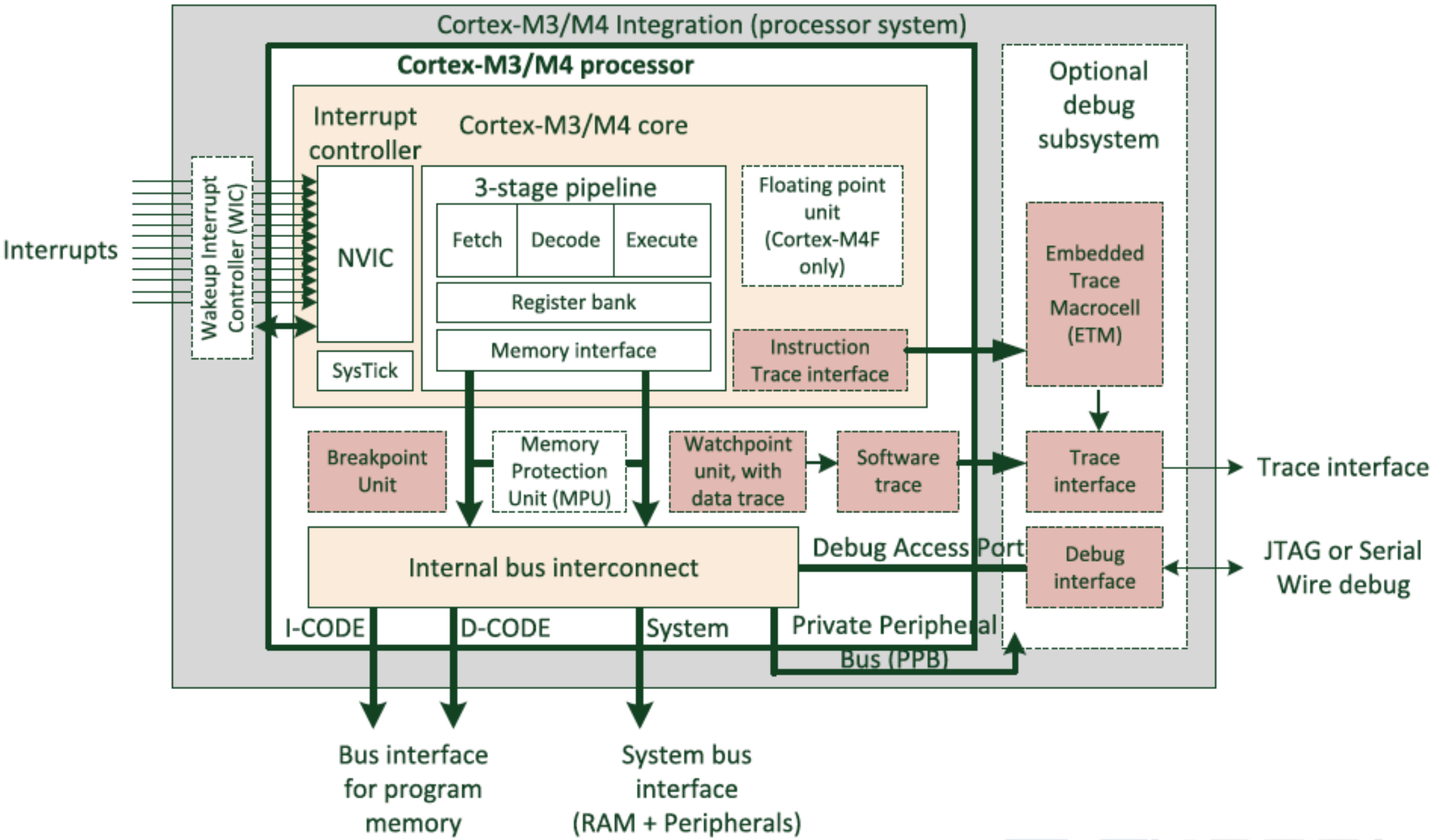


ARM instructions are 32-bit

Thumb instructions are 16-bit

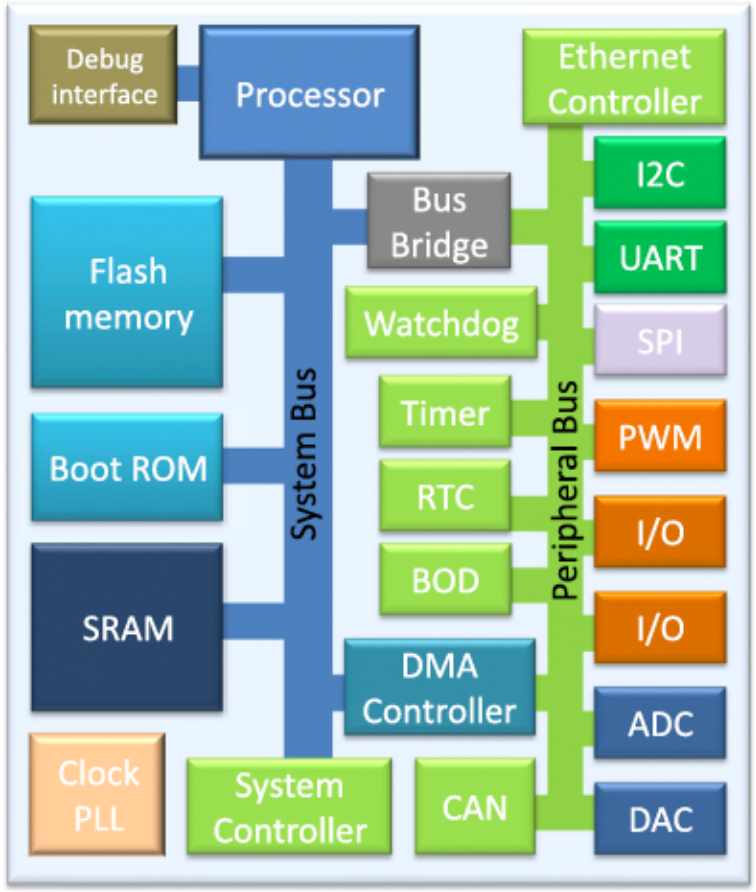
Thumb-2 instructions are 32-bit

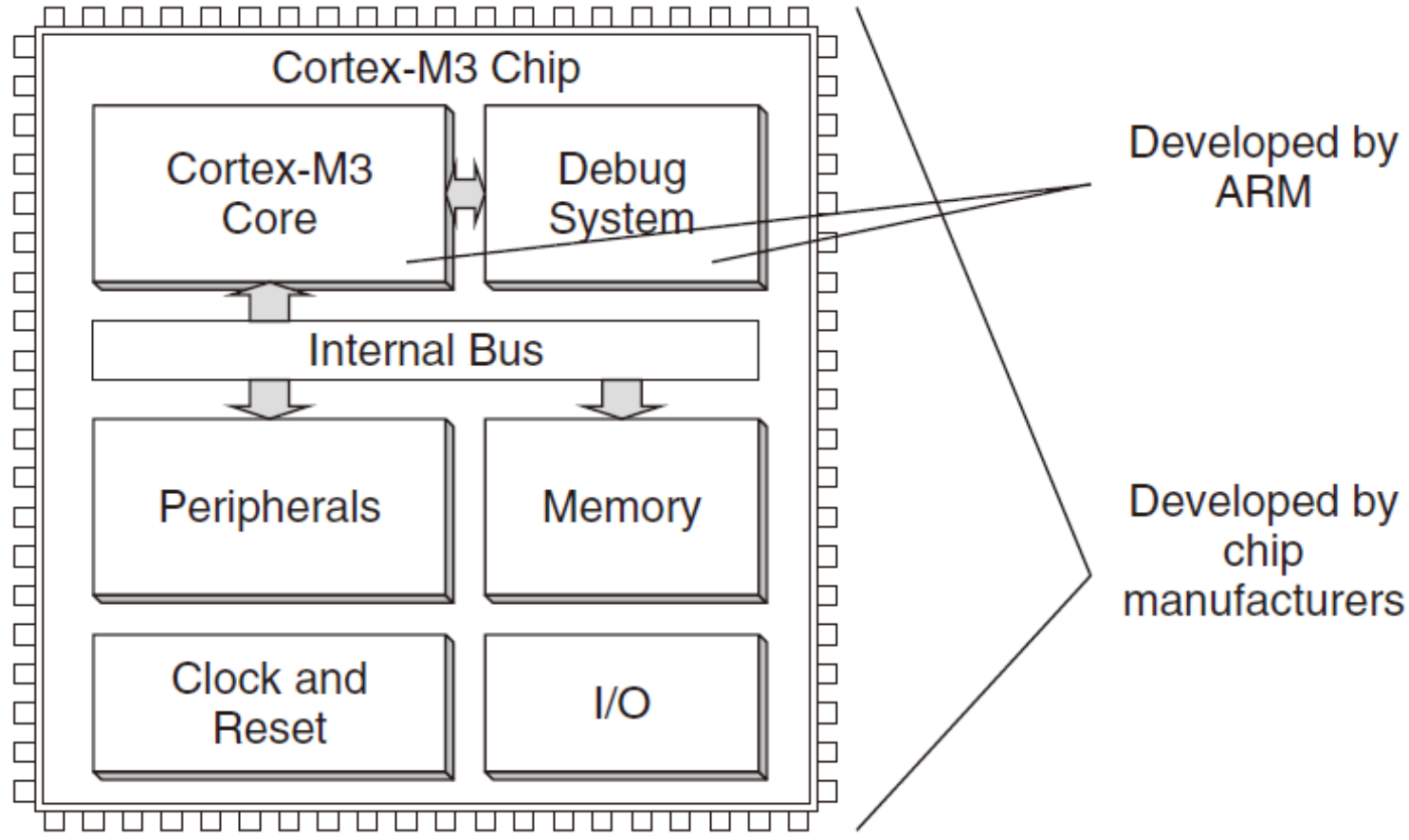
Overview of Cortex-M3/M4



Cortex-M3/M4 Processors vs MCUs

- The Cortex-M3/M4 processor is the central processing unit (CPU) of a microcontroller chip
- After chip manufacturers license the Cortex-M3/M4 processor, they can put the Cortex-M3/M4 processor in their silicon designs, adding memory, peripherals, input/output (I/O), and other features.
- Cortex-M3/M4 processor based chips from different manufacturers will have different memory sizes, types, peripherals, and features

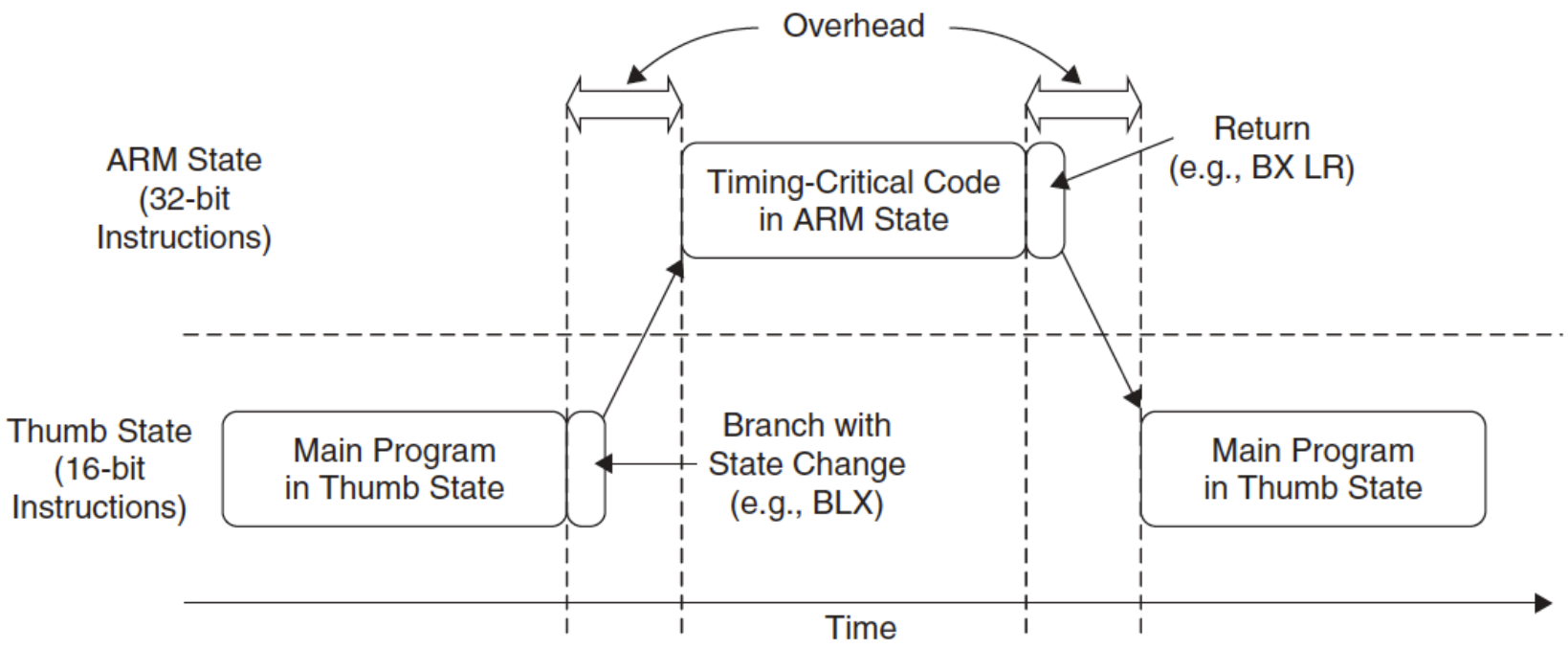




上海交通大學ARM Processors: Instruction State Switches

- In the ARM state, the instructions are 32-bit and can execute all supported instructions with very high performance
- In the Thumb state, the instructions are 16-bit, so there is a much higher instruction code density

Can we combine them to achieve the best of both worlds?

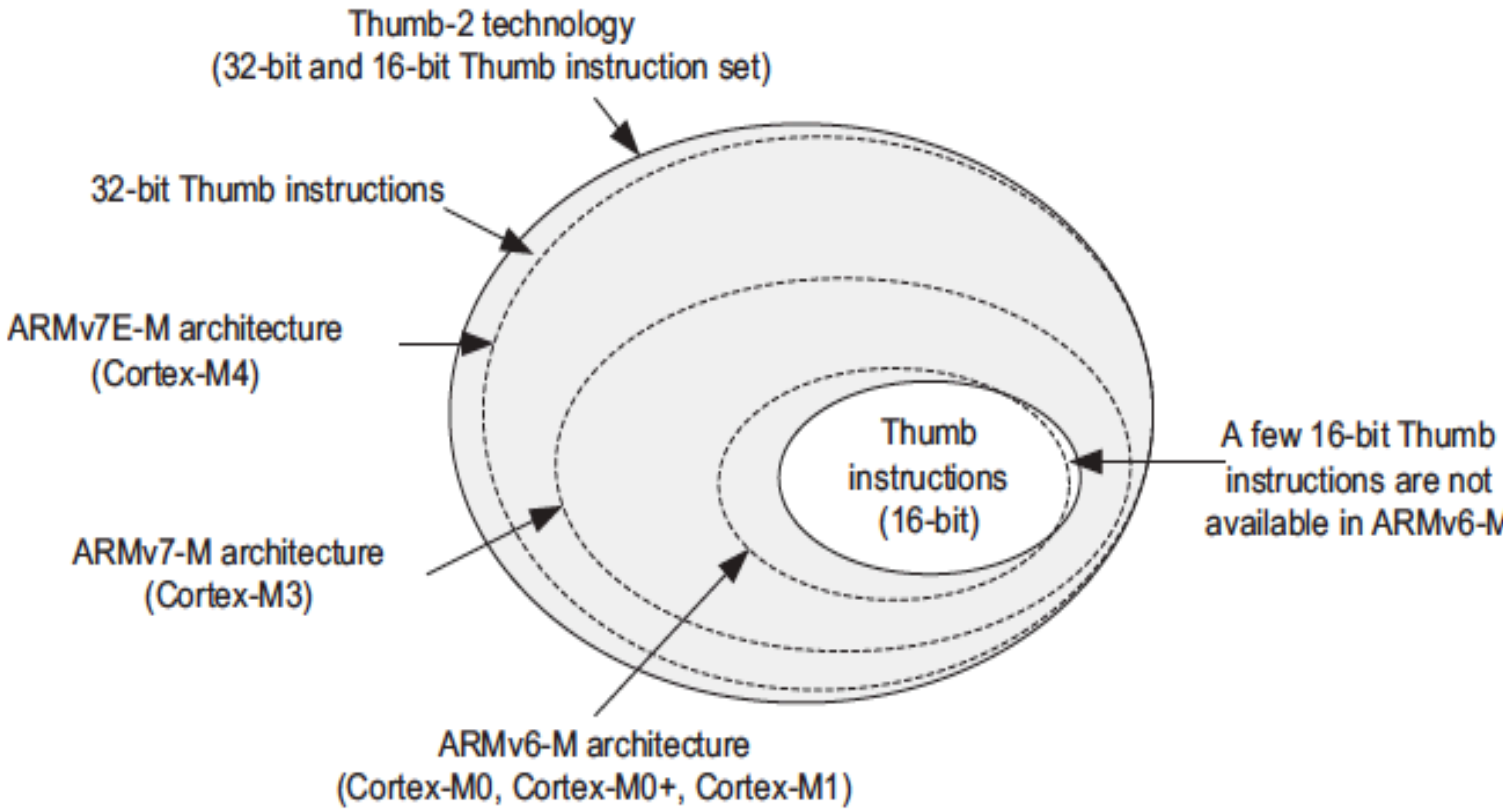


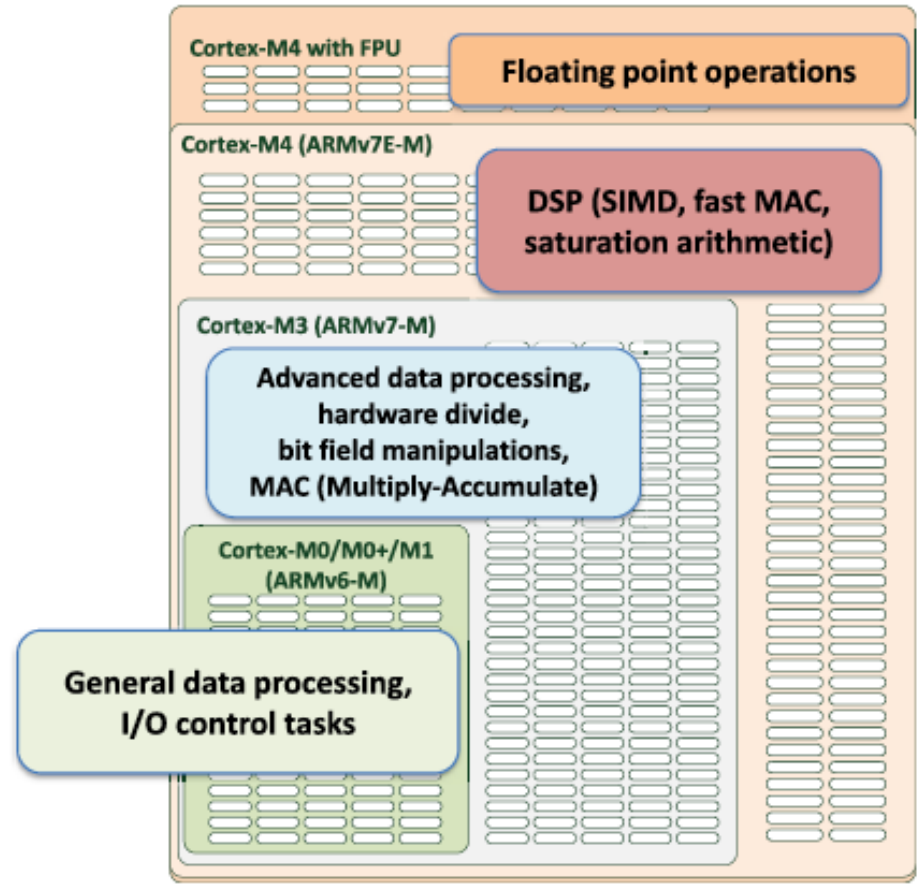
Any cost?

1) overhead of state switches
2) complex software development

Instruction Set of Cortex-M3/M4

- Cortex-M3/M4 supports only the **Thumb-2** (including the traditional Thumb) instruction set



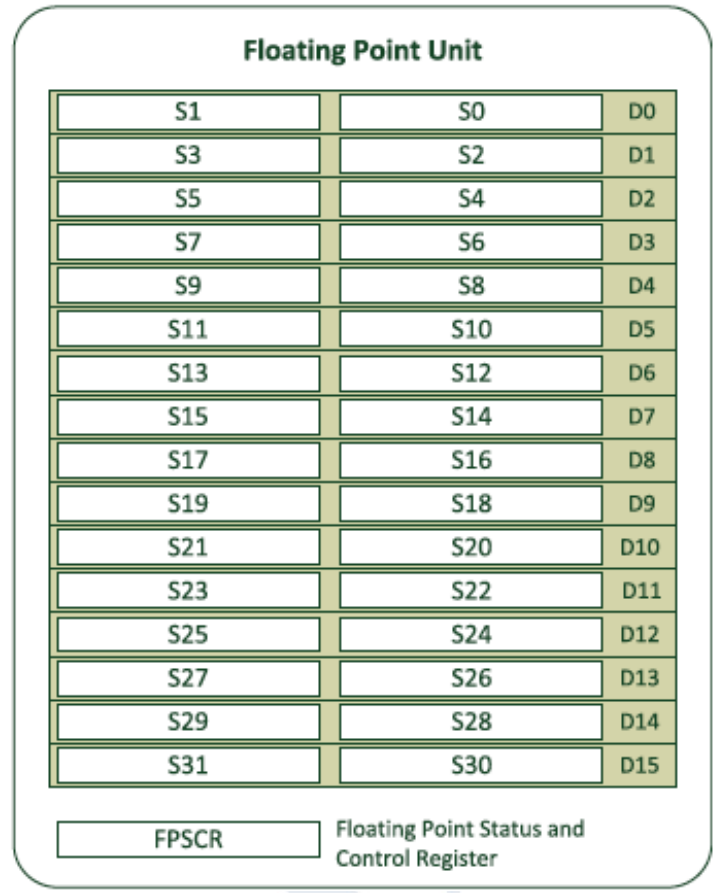


Advantages:

1) No state switching overhead, saving both execution time and instruction space
2) No need to separate ARM code and Thumb code source files, making software development and maintenance easier

Cortex-M3/M4 Basics: Registers

Cortex-M4 processor only



Cortex-M3 Basics: Registers

General-Purpose Registers

- **R0~R7** (low registers)
  - Can be accessed by all 16-bit Thumb instructions and all 32-bit Thumb-2 instructions
  - reset value is unpredictable
- **R8~R12** (high registers)
  - Accessible by all Thumb-2 instructions but not by all 16-bit Thumb instructions
  - reset value is unpredictable

Stack Pointer R13

- Two stack pointers are banked so that only one is visible at a time.
  - Main Stack Pointer (**MSP**) : This is the default stack pointer, used by the OS kernel, exception handlers, and privileged-mode programs
  - Process Stack Pointer (**PSP**) : Used by the user application code

Cortex-M3 Basics: Registers

Link Register R14

- **R14** is the link register (LR), used to store the return program counter when a subroutine or function is called.

e.g., when using the **BL** (Branch with Link) instruction:

```
main    ; Main program
        …
        BL  function1 ; Call function1 using Branch with Link
                      ; instruction.
                      ; PC = function1 and
                      ; LR = the next instruction in main
        …
function1
        …             ; Program code for function 1
        BX  LR        ; Return
```

Cortex-M3 Basics: Registers

Program Counter R15

- When you read this register you will find that the value is different than the location of the executing instruction by 4, due to the pipelining of the processor

Example:

```
0x1000 : MOV   R0, PC      ; R0 = 0x1004
```

- When reading the PC, the LSB (bit 0) is always 0.

  Why?

- When writing to the PC, it will cause a branch. The LSB must be set to 1 to indicate the Thumb state operations (setting to 0 implies to switch to the ARM state, which will result in a fault exception in Cortex-M3)

  Can you write to the PC in 8086? Why do we need to set LSB of R15 (true for R14) since all instructions are half-word or word aligned?

Cortex-M3 Basics: Registers

Special Registers

- The special registers in the Cortex-M3 processor include:
  - Program Status Registers (PSRs)
  - Interrupt Mask Registers (PRIMASK, FAULTMASK, and BASEPRI)
  - Control Register (CONTROL)

- Can only be accessed via MSR and MRS instructions

```
MRS <reg>, <special_reg> ; Read special register
MSR <special_reg>, <reg> ; write to special register
```

Note: MSR and MRS cannot have memory addresses, only registers are allowed

Special Registers

Program Status Registers (PSRs)

- The program status registers are subdivided into three status registers:

1. Application PSR (**APSR**), 2. Interrupt PSR (**IPSR**), 3. Execution PSR (**EPSR**)

	31	30	29	28	27	26:25	24	23:20	19:16	15:10	9	8	7	6	5	4:0
APSR	N	Z	C	V	Q											
IPSR												Exception Number				
EPSR						ICI/IT	T			ICI/IT						

- When they are accessed as a collective item, the name xPSR is used (**PSR** used in program codes).

	31	30	29	28	27	26:25	24	23:20	19:16	15:10	9	8	7	6	5	4:0
xPSR	N	Z	C	V	Q	ICI/IT	T			ICI/IT		Exception Number				

Cortex-M3 Interrupt Mask Registers

PRIMASK, FAULTMASK and BASEPRI Registers

- The PRIMASK, FAULTMASK, and BASEPRI registers are used to disable exceptions.

Register Name	Description
PRIMASK	A 1-bit register. When this is set, it allows NMI and the hard fault exception; all other interrupts and exceptions are masked; default is 0 (no masking)
FAULTMASK	A 1-bit register. When this is set, it allows only the NMI, and all interrupts and fault handling exceptions are disabled; default is 0
BASEPRI	A register of up to 9 bits. It defines the masking priority level. When this is set, it disables all interrupts of the same or lower level (larger priority value); default is 0

Special Registers

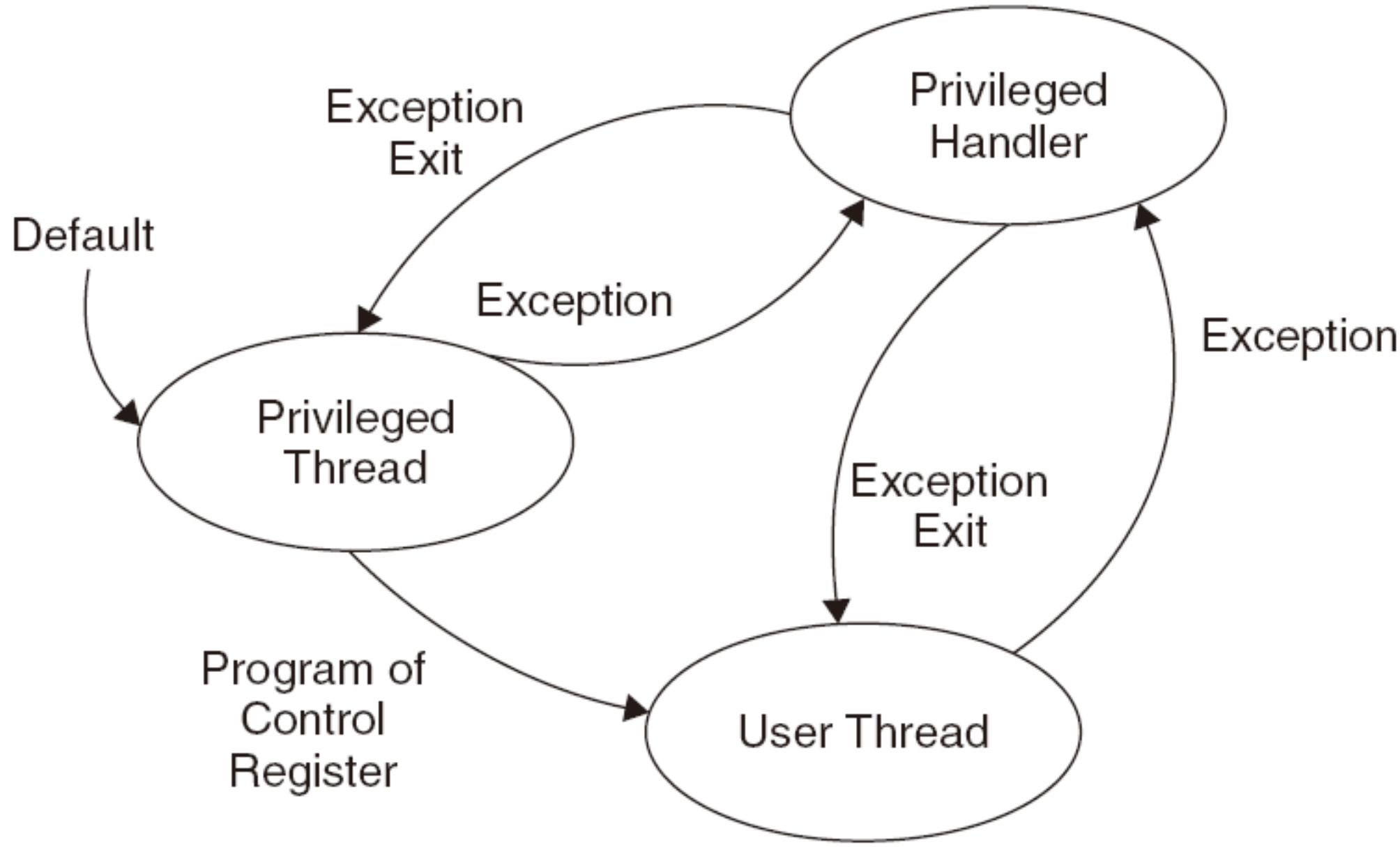
The Control Register

- The Control register is used to define the privilege level and the stack pointer selection. This register has two bits.

Bit	Function
CONTROL[1]	Stack status: 1 = Alternate stack is used 0 = Default stack (MSP) is used If it is in the Thread or base level, the alternate stack is the PSP. There is no alternate stack for handler mode, so this bit must be zero when the processor is in handler mode.
CONTROL[0]	0 = Privileged in Thread mode 1 = User state in Thread mode If in handler mode (not Thread mode), the processor operates in privileged mode.

Operation Modes in Cortex-M3/M4

- Two modes and two privilege levels



上海交通大學

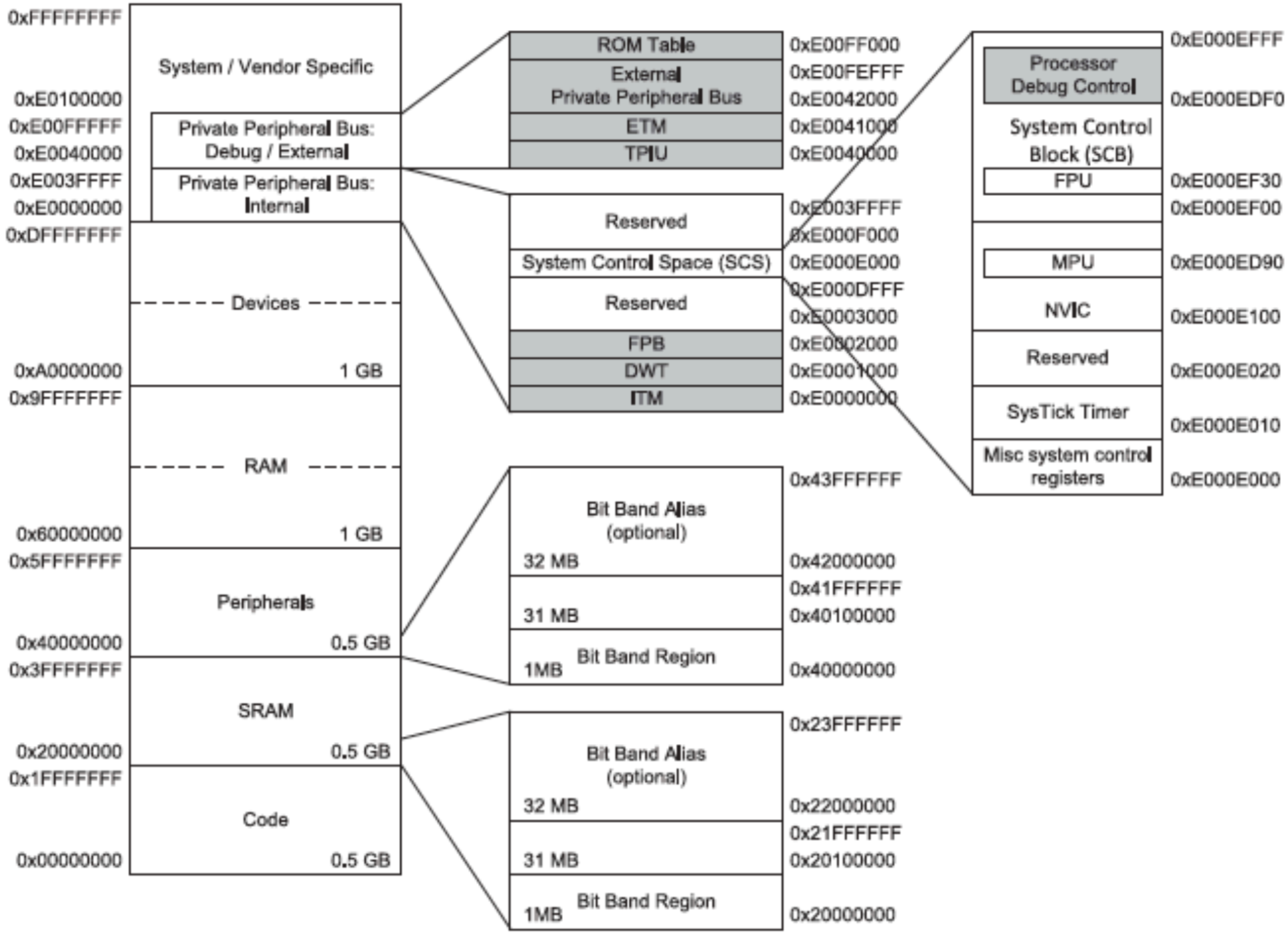
Cortex-M3/M4 Memory Systems

Refer to Chapter 6 in the reference book

“*The Definitive Guide to ARM Cortex-M3 and Cortex-M4 Processors*”

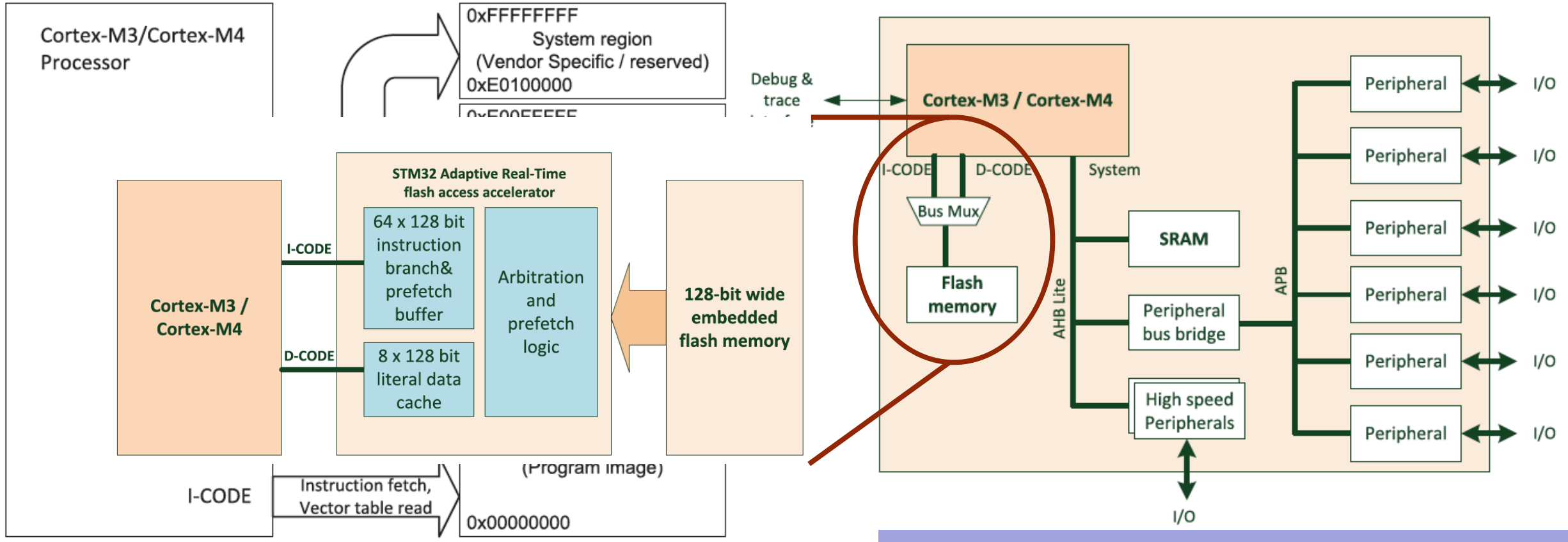
Predefined Memory Map

- The Cortex-M3/M4 processor has a ***unified*** and ***fixed*** memory map
- The Cortex-M3/M4 processor has a total of 4GB of address space



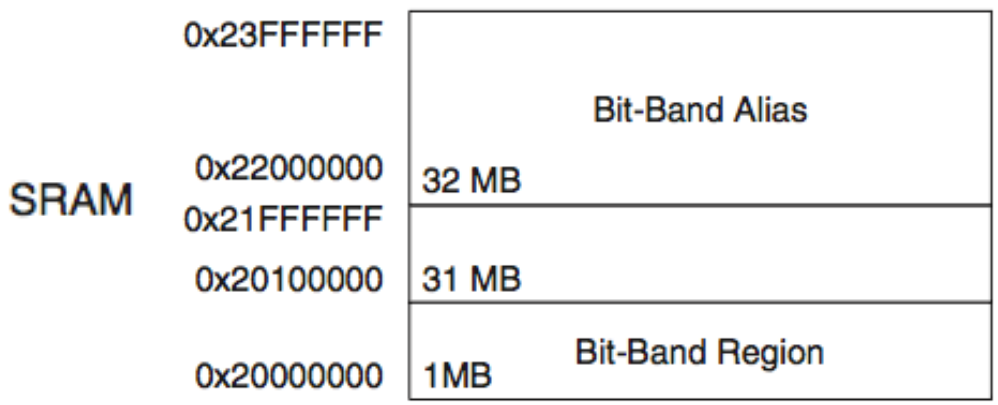
Connecting the Processor to Memory and Peripherals

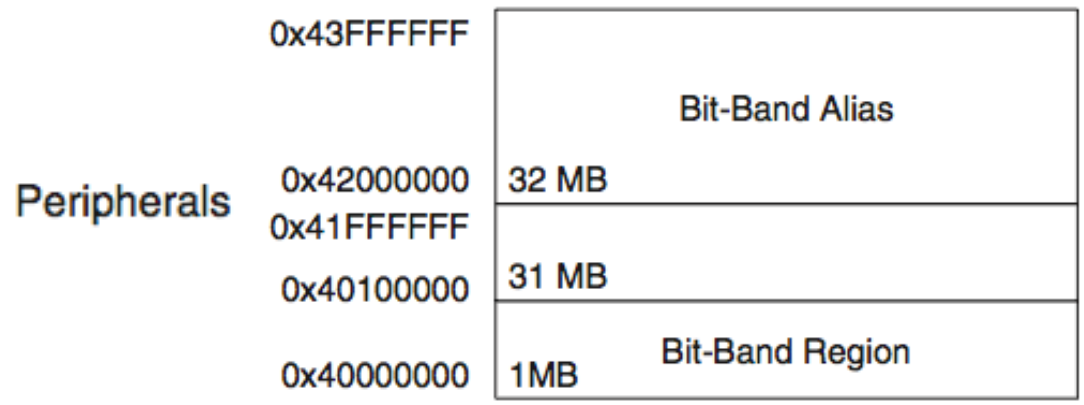
- the AHB (AMBA High-performance Bus) Lite protocol:
  - used for the main bus interfaces
- the APB protocol:
  - used for the Private Peripheral Bus (PPB), mainly used for debug components.
  - Additional bus segments based on APB can be added onto the system bus by using additional bus bridge components.



One simple microcontroller design

Bit-Band Region and Bit-Band Alias





Write to Bit-Band Alias

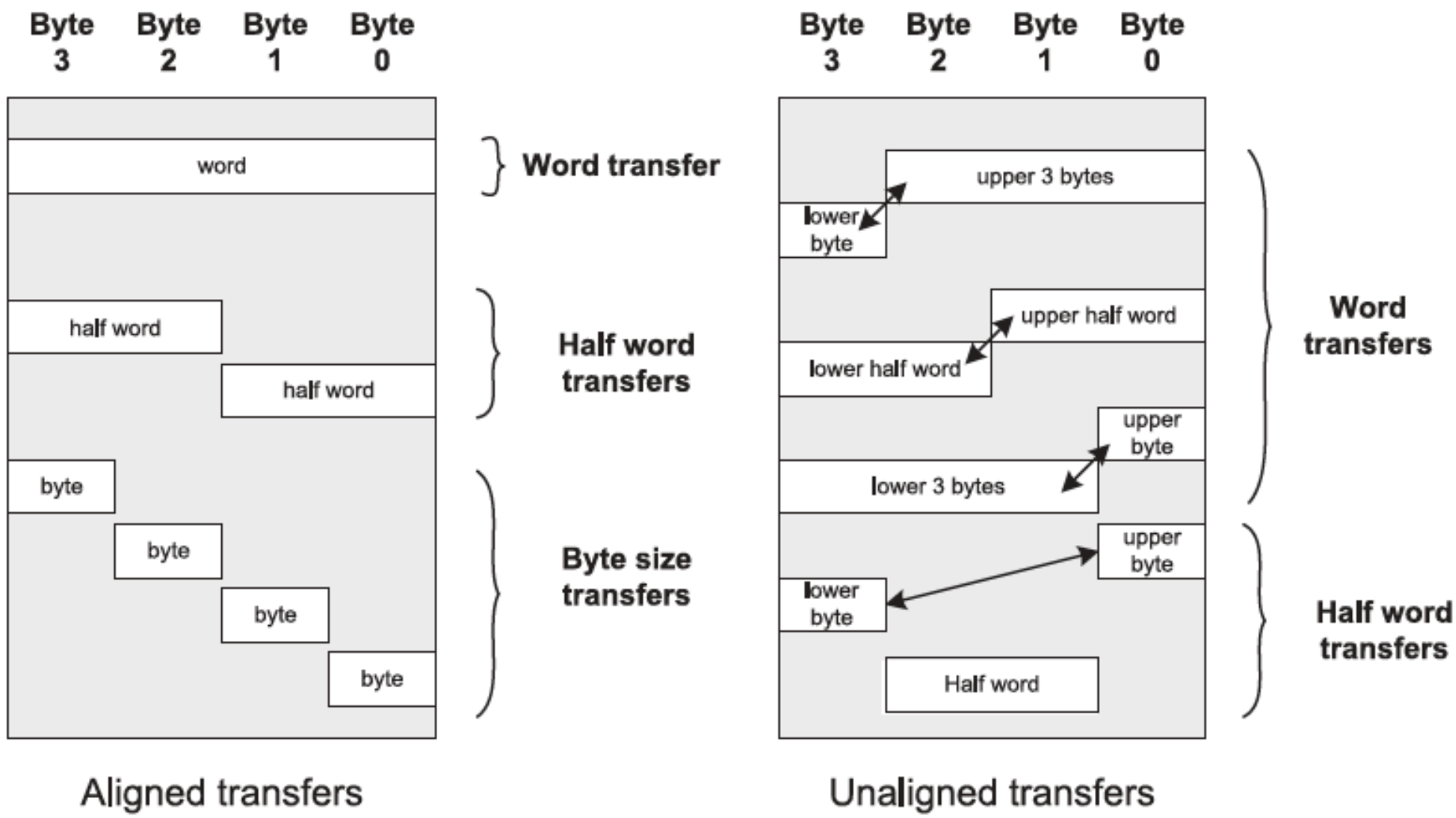
- A **hardware** READ-MODIFY-WRITE is performed
- For example, to set bit 2 in word data at address 0x20000000:

Read from Bit-Band Alias

- To read bit 2 in word data in address 0x20000000:

Data Alignment and Unaligned Transfers

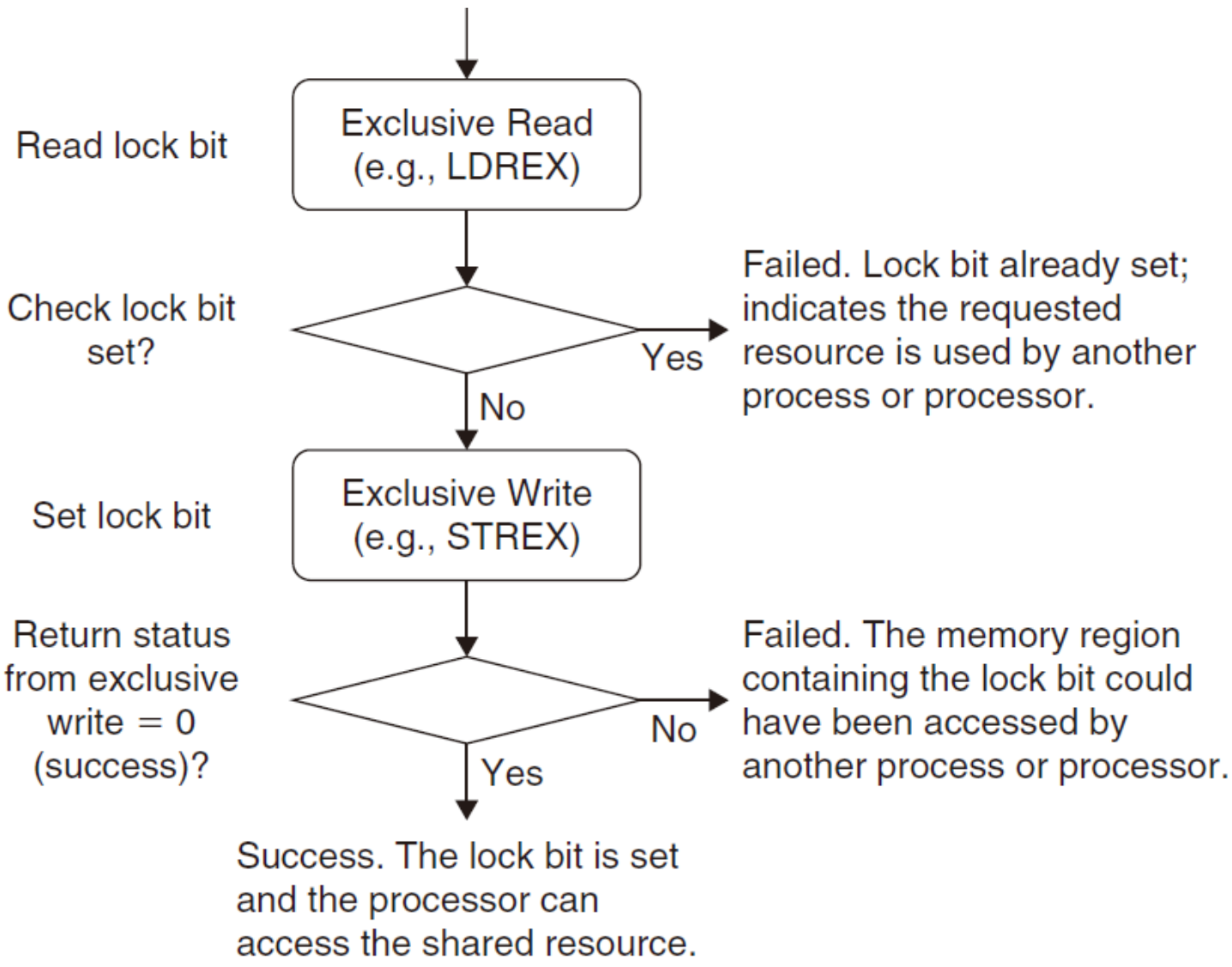
- Aligned transfers means the address value is a multiple of the size (in bytes).
- The Cortex-M3 supports unaligned transfers on single accesses.
  - When unaligned transfers are issued by the processor, they are actually converted into multiple aligned transfers by the processor's bus interface unit.



Exclusive Accesses

3. The difference between SWP instructions and exclusive access.

The concept of exclusive access operation is quite simple but different from SWP, which allows the possibility that the memory location for a semaphore could be accessed by multiple buses.



Endian Mode

- The Cortex-M3/M4 supports both little endian (recommended) and big endian modes.
- In the Cortex-M3/M4, the big endian scheme uses *byte-invariant big endian* (BE-8) byte lane usage.

Memory view

Table 6.3 Little Endian Memory View

Address	Bits 31 – 24	Bits 23 – 16	Bits 15 – 8	Bits 7 – 0
0x0003 – 0x0000	Byte – 0x3	Byte – 0x2	Byte – 0x1	Byte – 0x0
…				
0x1003 – 0x1000	Byte – 0x1003	Byte – 0x1002	Byte – 0x1001	Byte – 0x1000
0x1007 – 0x1004	Byte – 0x1007	Byte – 0x1006	Byte – 0x1005	Byte – 0x1004
…				
…	Byte – 4xN+3	Byte – 4xN+2	Byte – 4xN+1	Byte – 4xN

Table 6.4 Big Endian Memory View

Address	Bits 31 – 24	Bits 23 – 16	Bits 15 – 8	Bits 7 – 0
0x0003 – 0x0000	Byte – 0x0	Byte – 0x1	Byte – 0x2	Byte – 0x3
…				
0x1003 – 0x1000	Byte – 0x1000	Byte – 0x1001	Byte – 0x1002	Byte – 0x1003
0x1007 – 0x1004	Byte – 0x1004	Byte – 0x1005	Byte – 0x1006	Byte – 0x1007
…				
…	Byte – 4xN	Byte – 4xN+1	Byte – 4xN+2	Byte – 4xN+3

Byte lane usage on the bus

Table 6.5 The Cortex-M3 and Cortex-M4 (byte-invariant big-endian, BE-8) – Data on the AHB Bus

Address, Size	Bits 31 – 24	Bits 23 – 16	Bits 15 – 8	Bits 7 – 0
0x1000, word	Data bit[7:0]	Data bit [15:8]	Data bit [23:16]	Data bit [31:24]
0x1000, half word	-	-	Data bit [7:0]	Data bit [15:8]
0x1002, half word	Data bit [7:0]	Data bit [15:8]	-	-
0x1000, byte	-	-	-	Data bit [7:0]
0x1001, byte	-	-	Data bit [7:0]	-
0x1002, byte	-	Data bit [7:0]	-	-
0x1003, byte	Data bit [7:0]	-	-	-

Table 6.6 ARM7TDMI (word-invariant big-endian, BE-32) – Data on the AHB Bus

Address, Size	Bits 31 – 24	Bits 23 – 16	Bits 15 – 8	Bits 7 – 0
0x1000, word	Data bit [7:0]	Data bit [15:8]	Data bit [23:16]	Data bit [31:24]
0x1000, half word	Data bit [7:0]	Data bit [15:8]	-	-
0x1002, half word	-	-	Data bit [7:0]	Data bit [15:8]
0x1000, byte	Data bit [7:0]	-	-	-
0x1001, byte	-	Data bit [7:0]	-	-
0x1002, byte	-	-	Data bit [7:0]	-
0x1003, byte	-	-	-	Data bit [7:0]

Table 6.7 Little Endian – Data on the AHB Bus

Address, Size	Bits 31 – 24	Bits 23 – 16	Bits 15 – 8	Bits 7 – 0
0x1000, word	Data bit [31:24]	Data bit [23:16]	Data bit [15:8]	Data bit [7:0]
0x1000, half word	-	-	Data bit [15:8]	Data bit [7:0]
0x1002, half word	Data bit [15:8]	Data bit [7:0]	-	-
0x1000, byte	-	-	-	Data bit [7:0]
0x1001, byte	-	-	Data bit [7:0]	-
0x1002, byte	-	Data bit [7:0]	-	-
0x1003, byte	Data bit [7:0]	-	-	-

上海交通大學

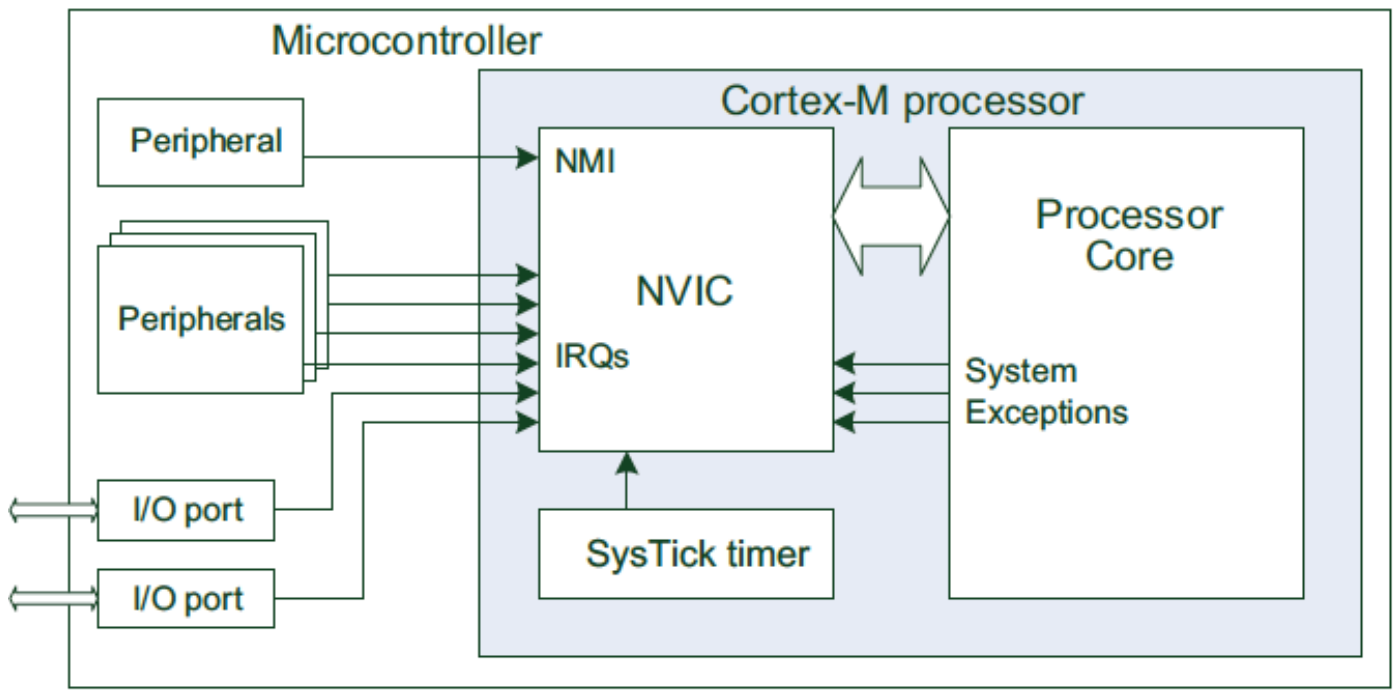
Cortex-M3/M4 Exceptions and Interrupts

Refer to Chapter 7, 8, 10, 12 in the reference book

"The Definitive Guide to ARM Cortex-M3 and Cortex-M4 Processors"

Exceptions

- Exceptions are numbered 1 to 15 for system exceptions and the rest 240 for external interrupt inputs.(Total 256 entries in vector table.)
- Most of the exceptions have programmable priority, and a few have fixed priority.
- The value of the current running exception is indicated by the special register **IPSR** or from the NVIC's **Interrupt Control State Register**.
- An enabled exception can be pended (which means it cannot be carried out immediately due to some reasons)



List of Exceptions

Table 7.1 List of System Exceptions

Exception Number	Exception Type	Priority	Description
1	Reset	−3 (Highest)	Reset
2	NMI	−2	Nonmaskable interrupt (external NMI input)
3	Hard Fault	−1	All fault conditions, if the corresponding fault handler is not enabled
4	MemManage Fault	Programmable	Memory management fault; MPU violation or access to illegal locations
5	Bus Fault	Programmable	Bus error; occurs when AHB interface receives an error response from a bus slave (also called *prefetch abort* if it is an instruction fetch or *data abort* if it is a data access)
6	Usage Fault	Programmable	Exceptions due to program error or trying to access coprocessor (the Cortex-M3 does not support a coprocessor)
7–10	Reserved	NA	–
11	SVCall	Programmable	System Service call
12	Debug Monitor	Programmable	Debug monitor (breakpoints, watchpoints, or external debug requests)
13	Reserved	NA	–
14	PendSV	Programmable	Pendable request for system device
15	SYSTICK	Programmable	System Tick Timer

Table 7.2 List of External Interrupts

Exception Number	Exception Type	Priority
16	External Interrupt #0	Programmable
17	External Interrupt #1	Programmable
...	...	...
255	External Interrupt #239	Programmable

Note that here the interrupt number (e.g., Interrupt #0) refers to the interrupt inputs to the Cortex-M3/M4 NVIC

Vector Table

- The **vector table** is an array of word data, with each representing the starting address of the ISR for one exception/interrupt type.
- The base address of the vector table is re-locatable (set the relocation register in the NVIC); initially, the base address is 0x0.
- The word stored at address 0x00000000 is used as the starting value for the MSP.
- **Note that the LSB of all vectors in the table must be set to 1 (indicating the exception will be executed in Thumb state)**

Example:

The reset is exception type 1. The address of the reset vector is 1 times 4, which equals 0x00000004; and NMI vector (type 2) is located at 2 * 4 = 0x00000008

Reset Sequence

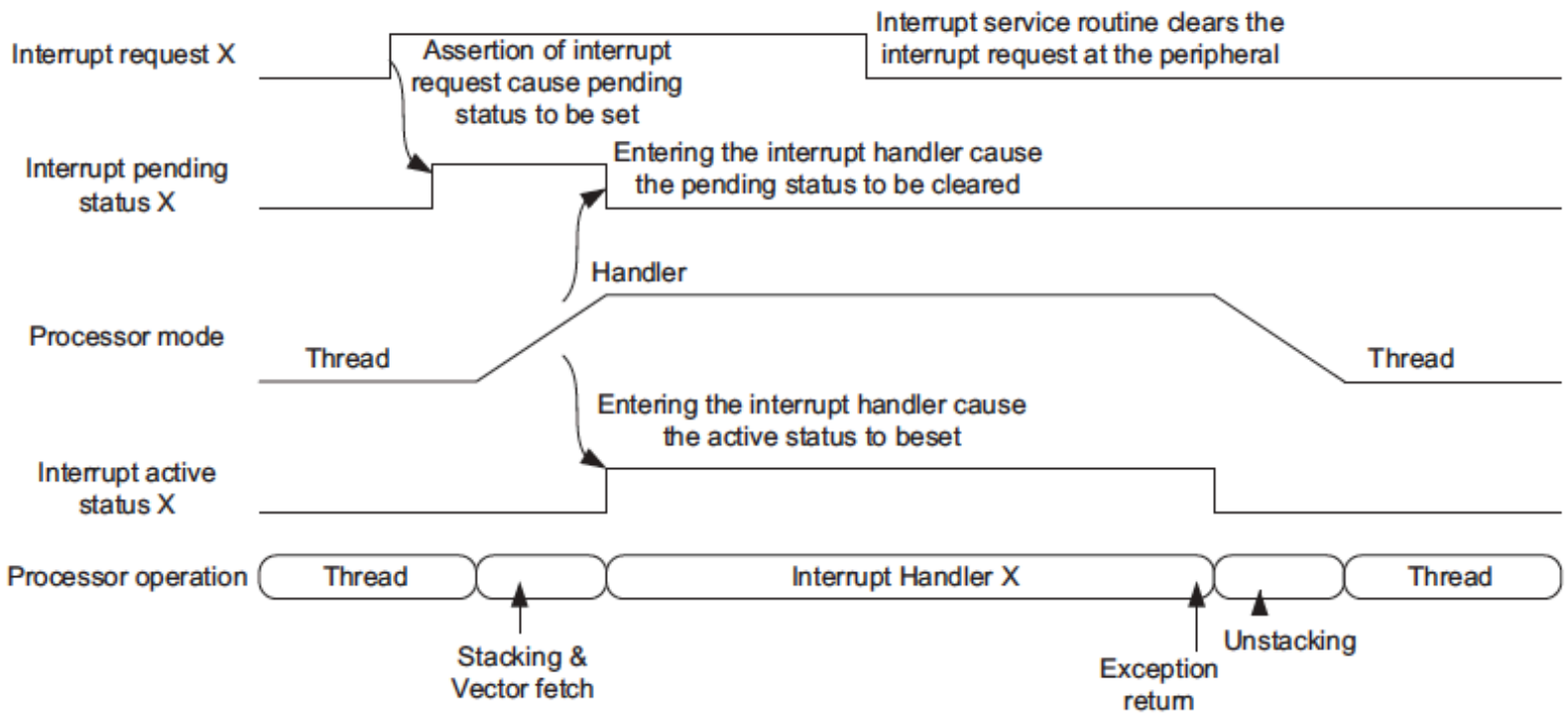
- After the processor exits reset, it will read two words from memory:
  - Address 0x00000000: default value of R13 (MSP)
  - Address 0x00000004: Reset vector (the starting address of startup program)

Exception Priority

- Whether and when an exception can be carried out can be affected by the priority of the exception
- A higher-priority (smaller number in priority level) exception can preempt (抢占) a lower-priority exception
- **Reset**, **NMI**, and **hard fault** have fixed highest-priority levels, i.e., -3, -2, and -1
- The Cortex-M3/M4 supports 256 levels of programmable priority
- The reduction of priority levels can be implemented by cutting out several ***lowest*** bits of the priority configuration register

Interrupt Inputs and Pending Behaviors

- An interrupt request can be accepted by the processor if:
  - *The pending status (hold by a register) of this interrupt is set*
  - *The interrupt is enabled (controlled by an interrupt masking register)*
  - *The priority of the interrupt is higher than the current level*
- Otherwise, the interrupt would be pended until the other interrupt handler is finished, or when the interrupt masking is cleared
- When the interrupt is being served, the pending status of the interrupt is cleared automatically and it is in the active state (hold by a register).



With the pending status of an interrupt, NVIC supports both pulsed interrupt requests and high level interrupt request

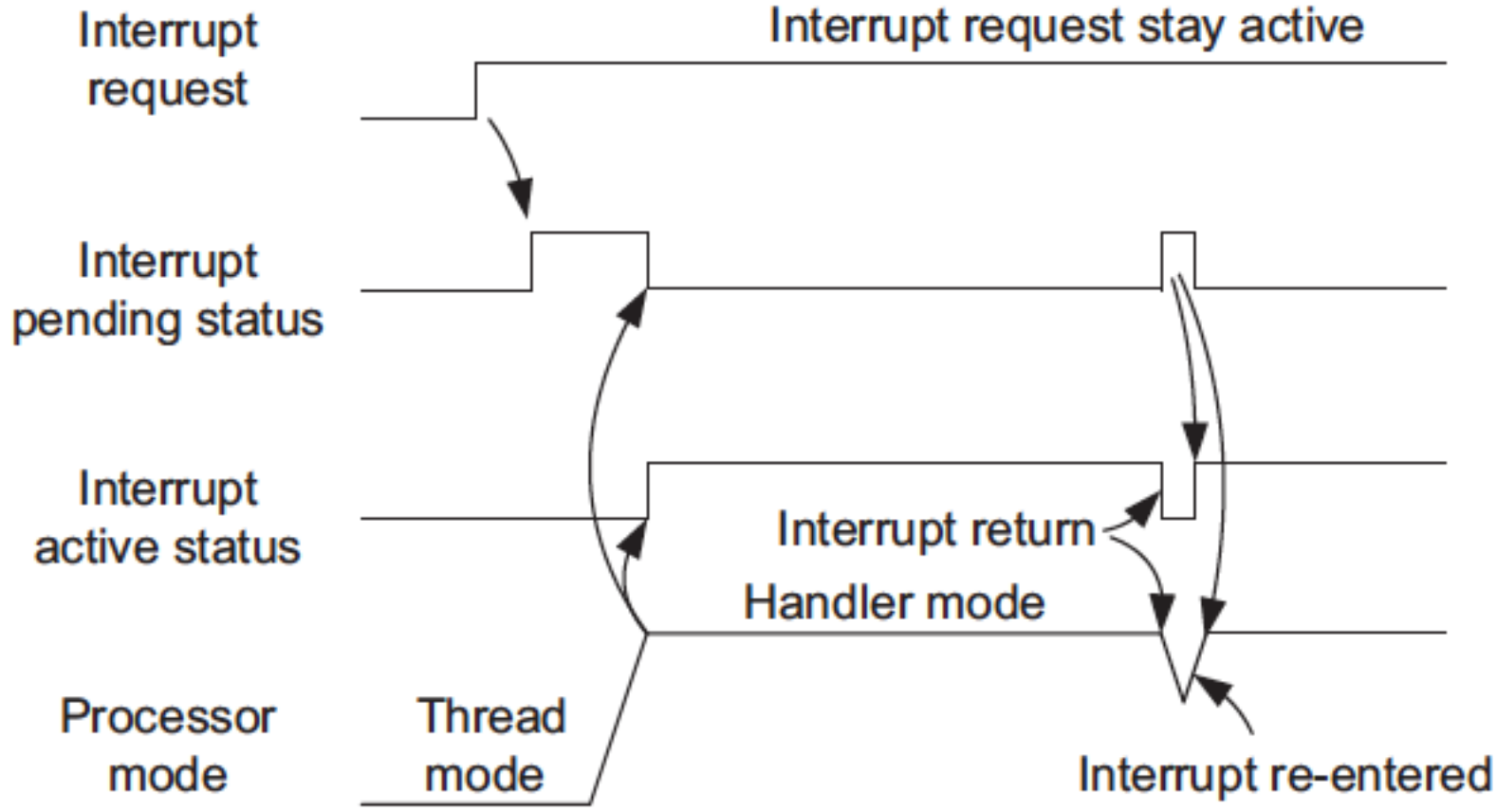
Interrupt Inputs and Pending Behaviors

- you can clear a pending interrupt or use software to pend a new interrupt by setting the pending register
- If the pending status is cleared (the pending status of the interrupt can be accessed in the NVIC and is writable) before the processor starts responding to the pended interrupt, the interrupt can be canceled

Interrupt Pending Cleared Before Processor Takes Action

Interrupt Inputs and Pending Behaviors

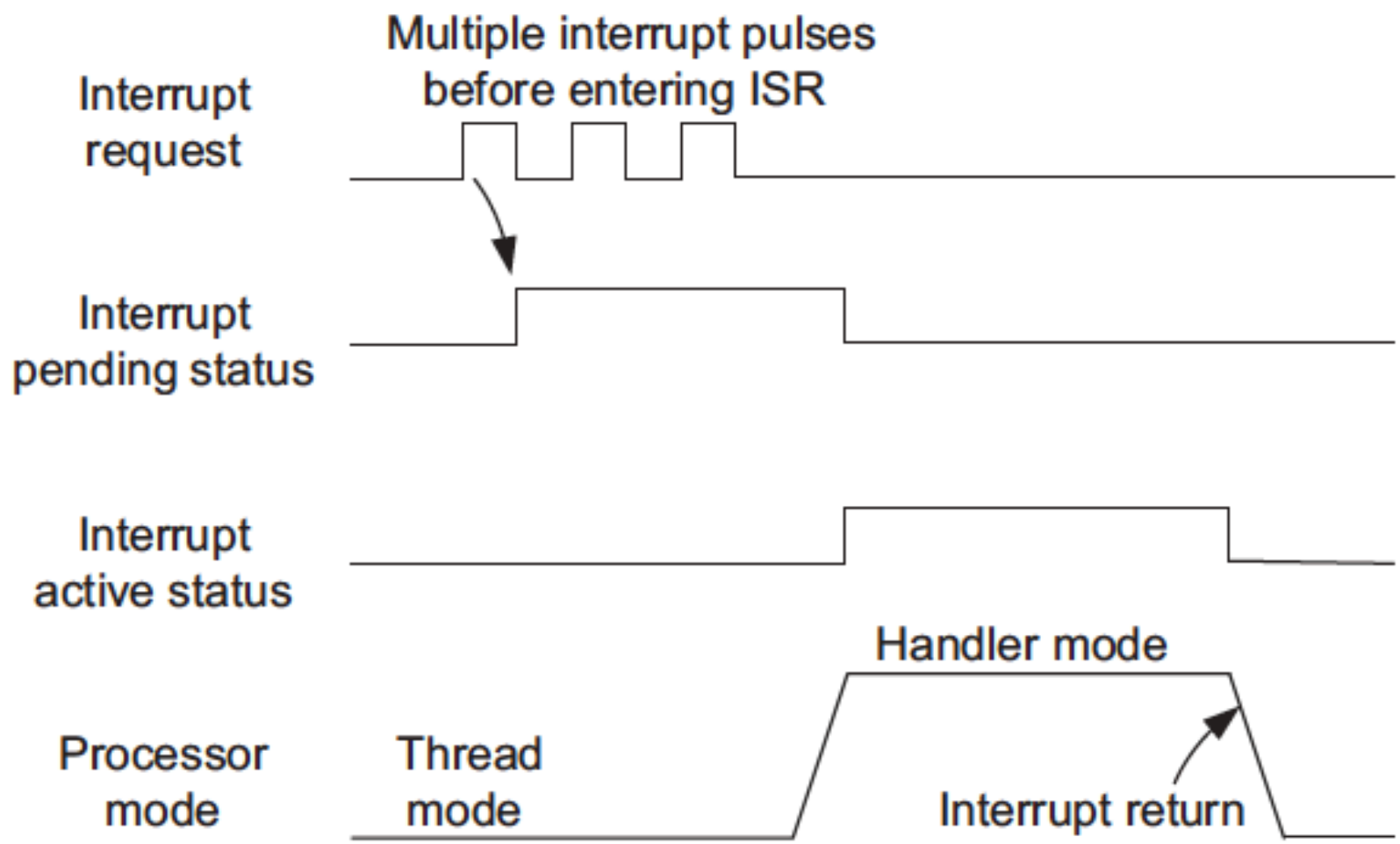
- If an interrupt source continues to hold the interrupt request signal active, the interrupt will be pended again at the end of the interrupt service routine.



Continuous Interrupt Request Pends Again After Interrupt Exit

Interrupt Inputs and Pending Behaviors

- If an interrupt is pulsed several times before the processor starts processing it, it will be treated as one single interrupt request.



Interrupt Pending Only Once, Even with Multiple Pulses Before the Handler

Interrupt Enable and Clear Enable

- The Interrupt Enable register is programmed via two addresses.
  - To set the enable bit, write 1 to the SETENA register address;
    Write 1 to set bit to 1; write 0 has no effect; read value indicates the current status
  - To clear the enable bit, write 1 to the CLRENA register address.
    Write 1 to clear bit to 0; write 0 has no effect; read value indicates the current status
  - SETENA/CLRENA registers are 32-bit, each bit represents one interrupt input
- If there are more than 32 external interrupts (up to 240), there would be more than one SETENA/CLRENA registers.
  - SETENA: 0xE000E100--0xE000E11C
  - CLRENA: 0xE000E180-0xE000E19C

Interrupt Pending and Clear Pending

- The interrupt-pending status can be programmed via two addresses.
  - To set the pending bit, write 1 to the Interrupt Set Pending (SETPEND) register;

    Write 1 to set bit to 1; write 0 has no effect; read value indicates the current status
  - To clear the pending bit, write 1 to the Interrupt Clear Pending (CLRPEND) register.

    Write 1 to clear bit to 0; write 0 has no effect; read value indicates the current status
  - SETPEND/ CLRPEND registers are 32-bit, each bit represents one interrupt input
  - User can set the certain bit of SETPEND to enter its handler by software.
- If there are more than 32 external interrupts (up to 240), there would be more than one SETPEND/ CLRPEND registers.
  - SETPEND: 0xE000E200-0xE000E21C
  - CLRPEND: 0xE000E280-0xE000E29C

Priority

- Each external interrupt has an associated priority-level register (a width of 3-8 bits)
- To find out the number of bits implemented for interrupt priority-level registers, write 0xFF to one of the priority-level registers, then read it back and see how many bits are set.

Interrupt Priority-Level Registers (0xE000E400-0xE000E4EF)

Name	Type	Address	Reset Value	Description
PRI_0	R/W	0xE000E400	0 (8-bit)	Priority-level external interrupt #0
PRI_1	R/W	0xE000E401	0 (8-bit)	Priority-level external interrupt #1
…	…	…	…	..
PRI_239	R/W	0xE000E4EF	0 (8-bit)	Priority-level external interrupt #239

Active Status

- Each external interrupt has an active status bit.
- When the processor starts the interrupt handler, the bit is set to 1 and cleared when the interrupt return is executed.

Interrupt Active Status Registers (0xE000E300-0xE000E31C)

Name	Type	Address	Reset Value	Description
ACTIVE0	R	0xE000E300	0	Active status for external interrupt #0-31
ACTIVE1	R	0xE000E304	0	Active status for external interrupt #32-63
…	…	…	…	…
ACTIVE7	R	0xE000E31C	0	Active status for external interrupt #224-239

Software Interrupts

- Software interrupts can be generated by using:
  1. The SETPEND register
  2. The ***Software Trigger Interrupt Register*** (STIR)

Software Trigger Interrupt Register (0xE000EF00)

Bits	Name	Type	Reset Value	Description
8:0	INTID	W	–	Writing the **interrupt number** sets the pending bit of the interrupt; for example, write 0 to pend external interrupt #0

- System exceptions (NMI, faults, PendSV, and so on) cannot be pended using STIR.
- STIR can be accessed in user level when the bit 1 (USERSETMPEND) of the *NVIC Configuration Control Register* is set.

Procedures in Setting Up an Interrupt

1. Set up the priority group register (group 0 by default).

2. Setup the hard fault and NMI handlers to a new vector table location if vector table relocation is required.

3. Set up the Vector Table Offset register if needed.

4. Set up the interrupt vector for the interrupt：[read the Vector Table Offset register and] calculate the correct memory location for the interrupt handler.

5. Set up the priority level for the interrupt.

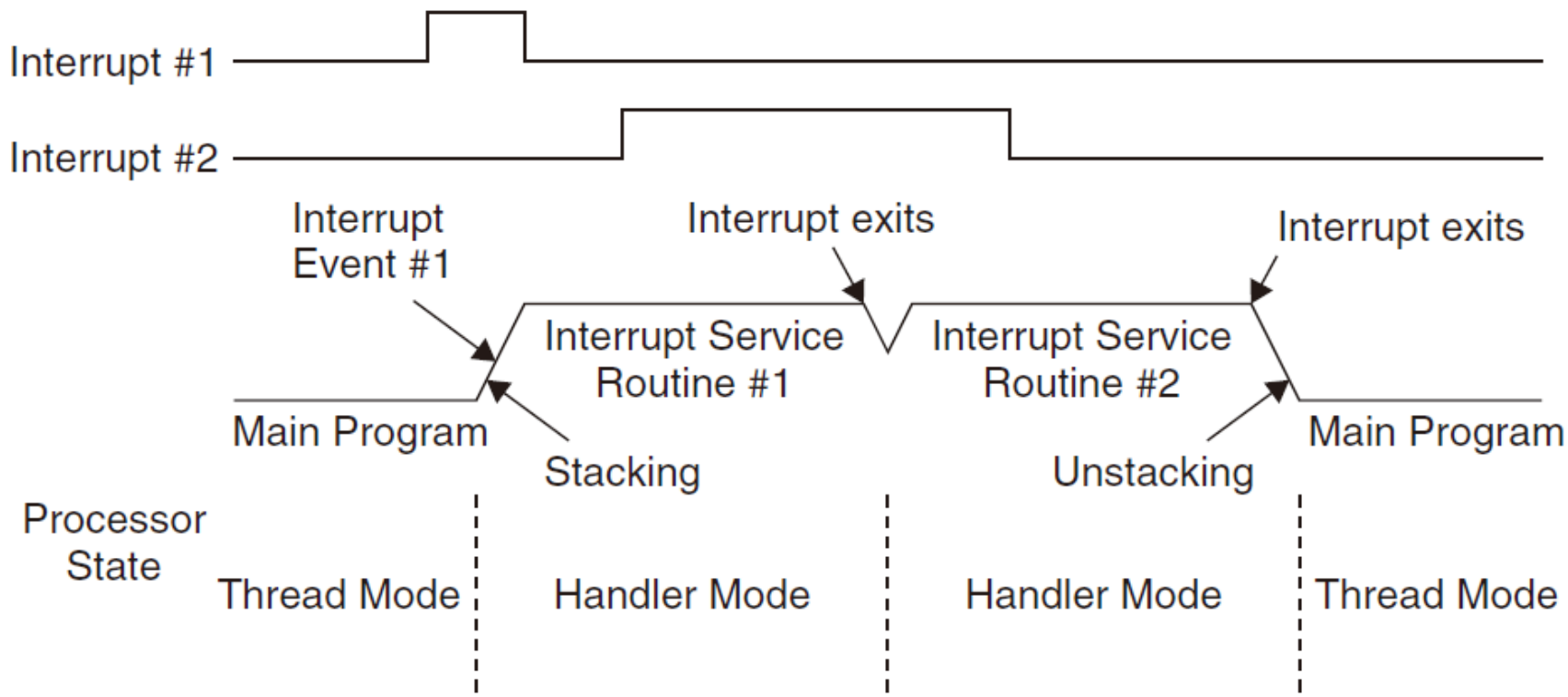
6. Enable the interrupt.

Nested Interrupts

- Being built into the Cortex-M3 processor core and the NVIC :
  - When the processor is handling an exception, all other exceptions with the same or lower priority will be blocked (pended)
  - The automatic hardware stacking and un-stacking allow the nested interrupt handler to execute without risk of losing data in registers
- Be Aware of:
  - Make sure that there is enough space in the main stack if lots of nested interrupts are allowed
  - Reentrant exceptions are not allowed in the Cortex-M3, e.g., SVC instructions cannot be used inside an SVC handler

Tail-Chaining Interrupts

- Used to improve interrupt latency:
  - When the processor is handling an exception, all other exceptions with the same or lower priority will be blocked (pended)
  - When the processor has finished executing the current exception handler, the un-stacking for this handler and the stacking for the next handler are skipped (reducing the latency of 12 cycles to 6)



Late Arrival Exception Handling

- Used to improve interrupt latency:
  - When an exception takes place and the processor has started the stacking process, if a new exception arrives with higher preemption priority, the late arrival exception will be processed first.