

# Cortex-M Processors - Memory Systems

Lecture 4

Yeongpil Cho

Hanyang University

# Topics

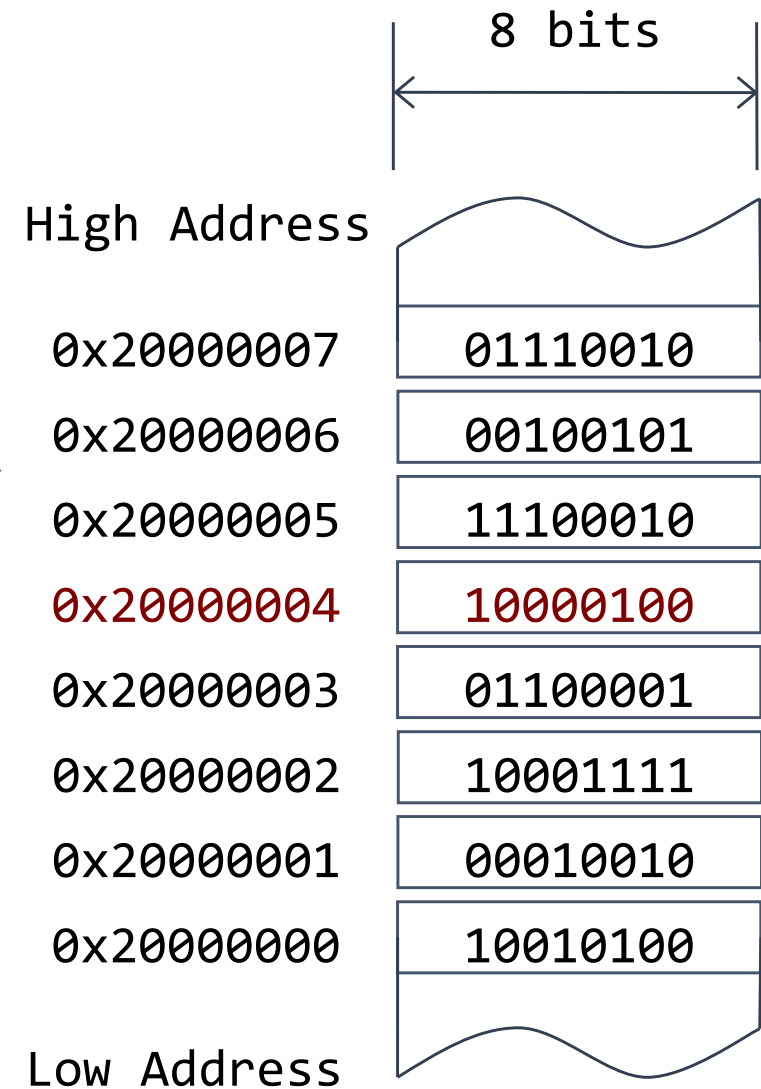
- Memory Map
- MPU
- Bit-banding
- Miscellaneous features

# Logic View of Memory

- By grouping bits together, we can store more values
  - 8 bits = **1 byte**
  - 16 bits = 2 bytes = **1 halfword**
  - 32 bits = 4 bytes = **1 word**
- From software perspective, **memory is an addressable array of bytes**.
  - The byte stored at the memory address 0x20000004 is 0b10000100

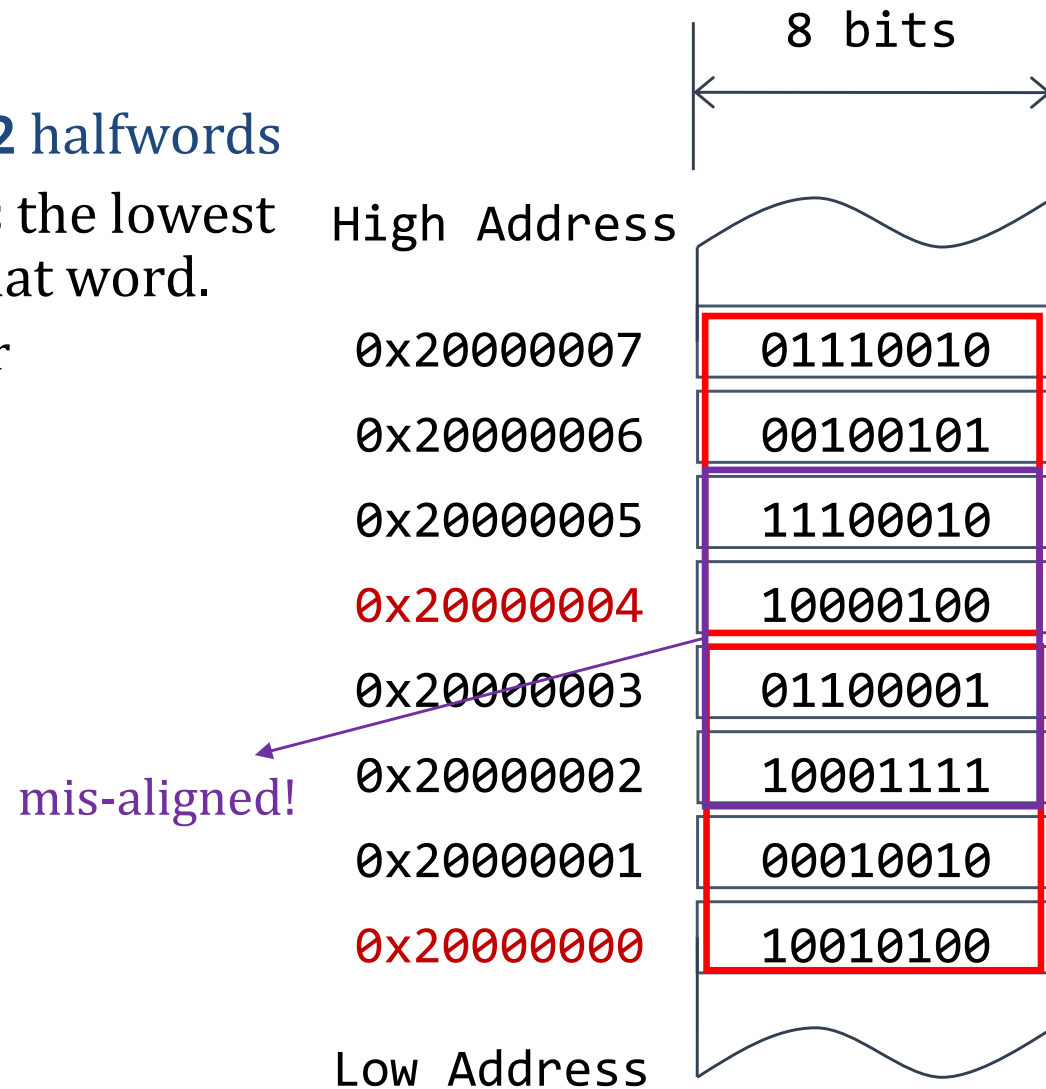
0b10000100 → 0x84 → 132

Binary                      Hexadecimal                      Decimal



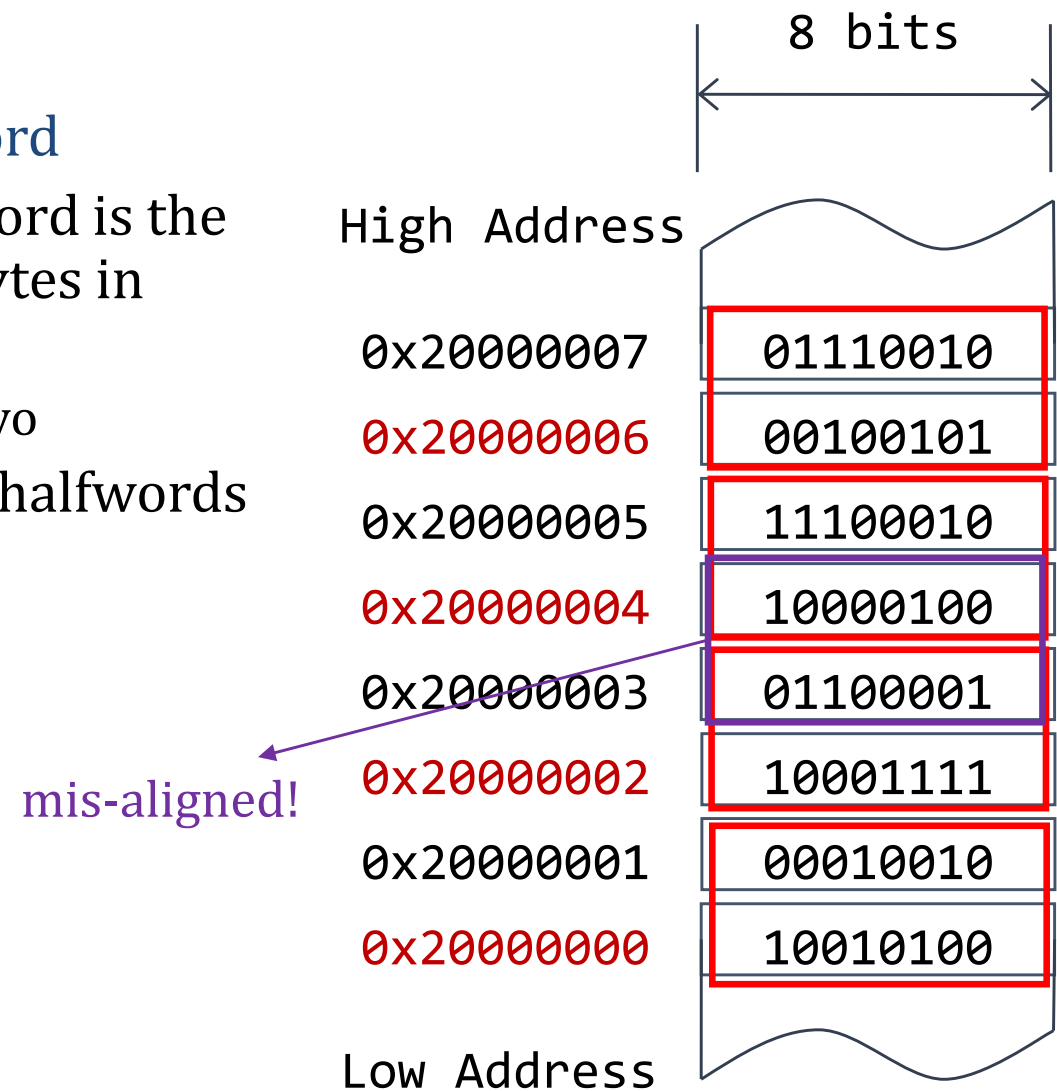
# Logic View of Memory

- Words
  - 32 bits = 4 bytes = **1 word** = **2 halfwords**
  - Memory address of a word is the lowest address of all four bytes in that word.
    - aligned to a multiple of four
  - Two words at addresses:
    - 0x20000000
    - 0x20000004



# Logic View of Memory

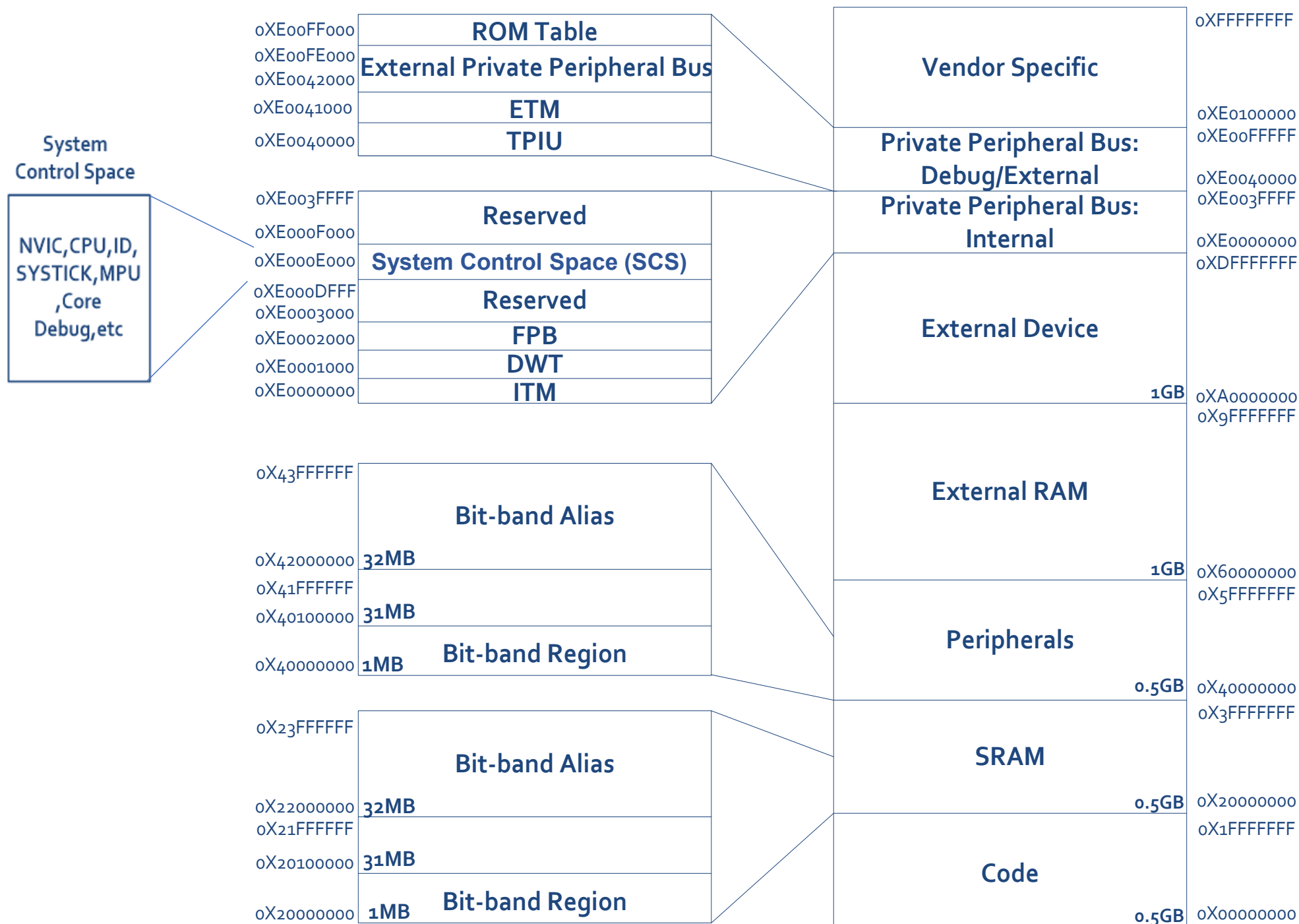
- Halfwords
  - **16 bits = 2 bytes = 1 halfword**
  - Memory address of a halfword is the lowest address of all two bytes in that word.
    - aligned to a multiple of two
  - The right diagram has four halfwords at addresses of:
    - 0x20000000
    - 0x20000002
    - 0x20000004
    - 0x20000006



# Memory Map

# Memory Maps

- Memory maps describe an address space layout of hardware components including memory, system and I/O peripherals
- The Cortex-M3 processor has a total of 4 GB of address space.
  - No 4 GB of physical memory
- The Cortex-M3 processor has a fixed memory map.
  - Easy software porting from one Cortex-M3 product to another
  - For example, Nested Vectored Interrupt Controller (NVIC) and Memory Protection Unit (MPU) have the same memory locations in all Cortex-M3 products

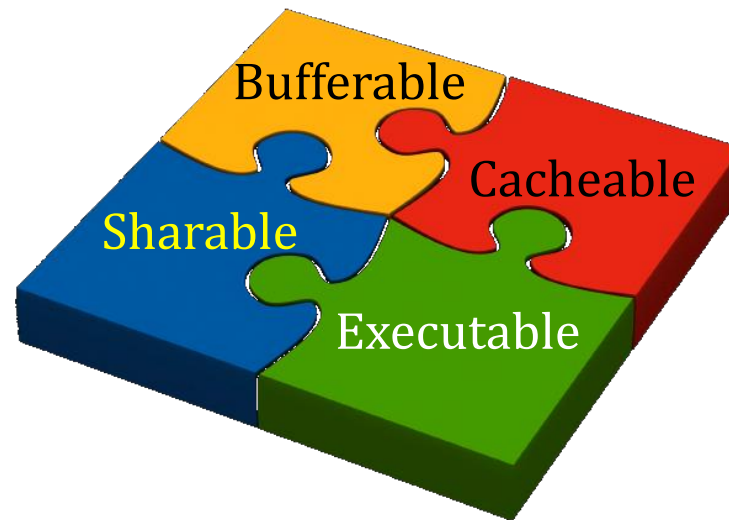


**A Cortex-M3 Predefined Memory Map**



# Memory Access Attributes

- The memory map also defines the memory attributes of accessing each memory block or device:



- Cortex-M3 has the default memory attribute settings
  - can be overridden if MPU is present and the region is programmed differently from the default.

# Memory Access Attributes

- **Bufferable** *Reduce right access —*
  - Write to memory can be carried out by a write buffer while the processor continues on next instruction execution.
- **Cacheable**
  - Data obtained from memory read can be copied to a memory cache so that next time it is accessed the value can be obtained from the cache to speed up the program execution.
- **Executable**
  - The processor can fetch and execute program code from this memory region.
- **Sharable**
  - Data in this memory region could be shared by **multiple bus masters**. **Memory system** needs to **ensure coherency of data between different bus masters** in shareable memory region.

*Bus structure*

# Default Memory Attributes

Region	Address	For	Cacheable	Executable, buffered
Code memory region	0x00000000–0x1FFFFFFF	Code and data as well	Cacheable	Executable, Buffered
SRAM memory region	0x20000000–0x3FFFFFFF	On-chip RAM	Cacheable	Executable, Buffered
Peripheral region	0x40000000–0x5FFFFFFF	Peripherals	Noncacheable	Nonexecutable, Buffered
External RAM region	0x60000000–0x7FFFFFFF	Either on-chip or off-chip memory	Cacheable	Executable, Buffered
External RAM region	0x80000000–0x9FFFFFFF	Either on-chip or off-chip memory	Cacheable	Executable, Buffered
External devices	0xA0000000–0xBFFFFFFF	External devices and/or shared memory	Noncacheable	Nonexecutable, Nonbuffered
External devices	0xC0000000–0xDFFFFFFF	External devices and/or shared memory	Noncacheable	Nonexecutable, Nonbuffered
System region	0xE0000000–0xFFFFFFFF	Private peripherals and Vendor-specific devices	Noncacheable	Nonexecutable, Nonbuffered (Buffered for vendor-specific memory)

## Address Space Division and Attributes

# Default Memory Access Permissions

- The Cortex-M3 memory map has a default configuration for memory access permissions
- The default memory access permission is used when either:
  - 1. No MPU is present
  - 2. MPU is present but disabled
- Otherwise, the MPU will determine whether memory accesses are allowed
- When a memory access is blocked, the **fault exception** takes place immediately.

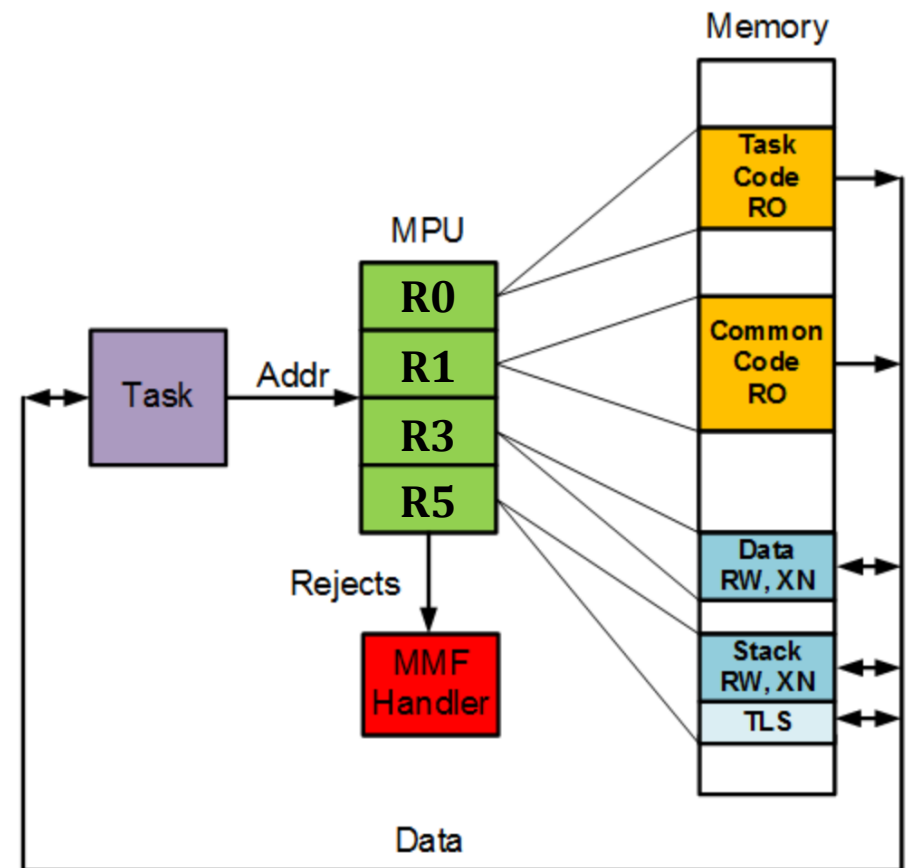
# Default Memory Access Permissions

Memory Region	Address	Access in User Program
Vendor specific	0xE0100000–0xFFFFFFFF	Full access
ETM	0xE0041000–0xE0041FFF	Blocked; user access results in bus fault
TPIU	0xE0040000–0xE0040FFF	Blocked; user access results in bus fault
Internal PPB	0xE000F000–0xE003FFFF	Blocked; user access results in bus fault
SCS	0xE000E000–0xE000EFFF	Blocked; user access results in bus fault
FPB	0xE0002000–0xE0003FFF	Blocked; user access results in bus fault
DWT	0xE0001000–0xE0001FFF	Blocked; user access results in bus fault
ITM	0xE0000000–0xE0000FFF	Read allowed; write ignored except for stimulus ports with user access enabled
External Ram	0x60000000–0x9FFFFFFF	Full access
Peripheral	0x40000000–0x5FFFFFFF	Full access
SRAM	0x20000000–0x3FFFFFFF	Full access
Code	0x00000000–0x1FFFFFFF	Full access

MPU

# MPU (Memory Protection Unit)

- MPU provides functionality to check memory and I/O accesses are allowed.
  - Supports 8~16 distinct memory regions
  - Each region can be set to some combinations of fetch, read and write permissions.
    - Access attributes as well e.g., cacheability, shareability
- A Trap (exception) arises when access is not permitted
  - MMF: Memory Management Fault



# MPU (Memory Protection Unit)

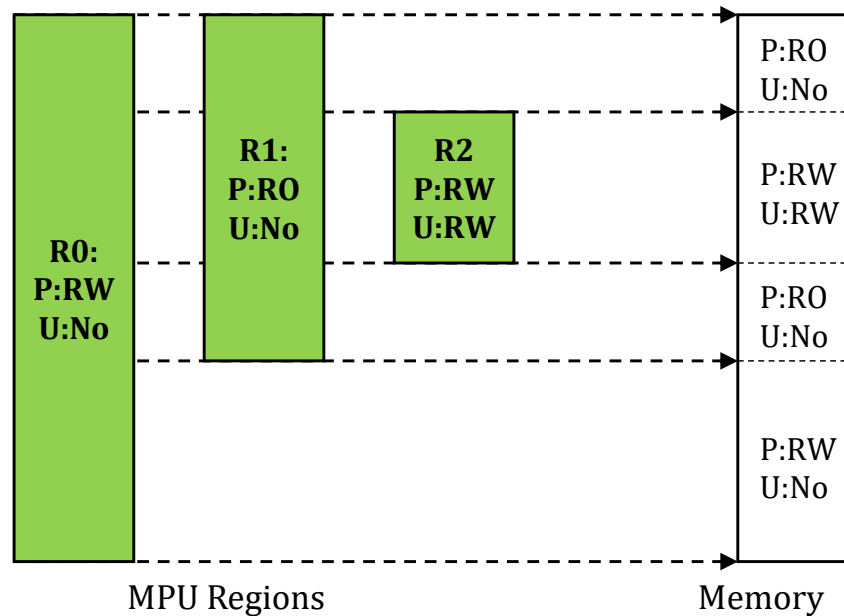
- Combinations of memory access permissions

<b>Privileged Permissions</b>	<b>Unprivileged Permissions</b>	<b>Description</b>
No Access	No Access	All accesses generate a permission fault
RO	No Access	Privileged read only
RO	RO	Un-/Privileged read only
RW	No Access	Privileged access only
RW	RO	Unprivileged writes generate permission faults
RW	RW	Full access



# MPU (Memory Protection Unit)

- Regions can be overlapped (ARMv7 only)
  - When memory regions overlap, a memory access is affected by the attributes of the region with the highest number.



*8 ~ 16 memory*

*concurrently*

# Bit-banding

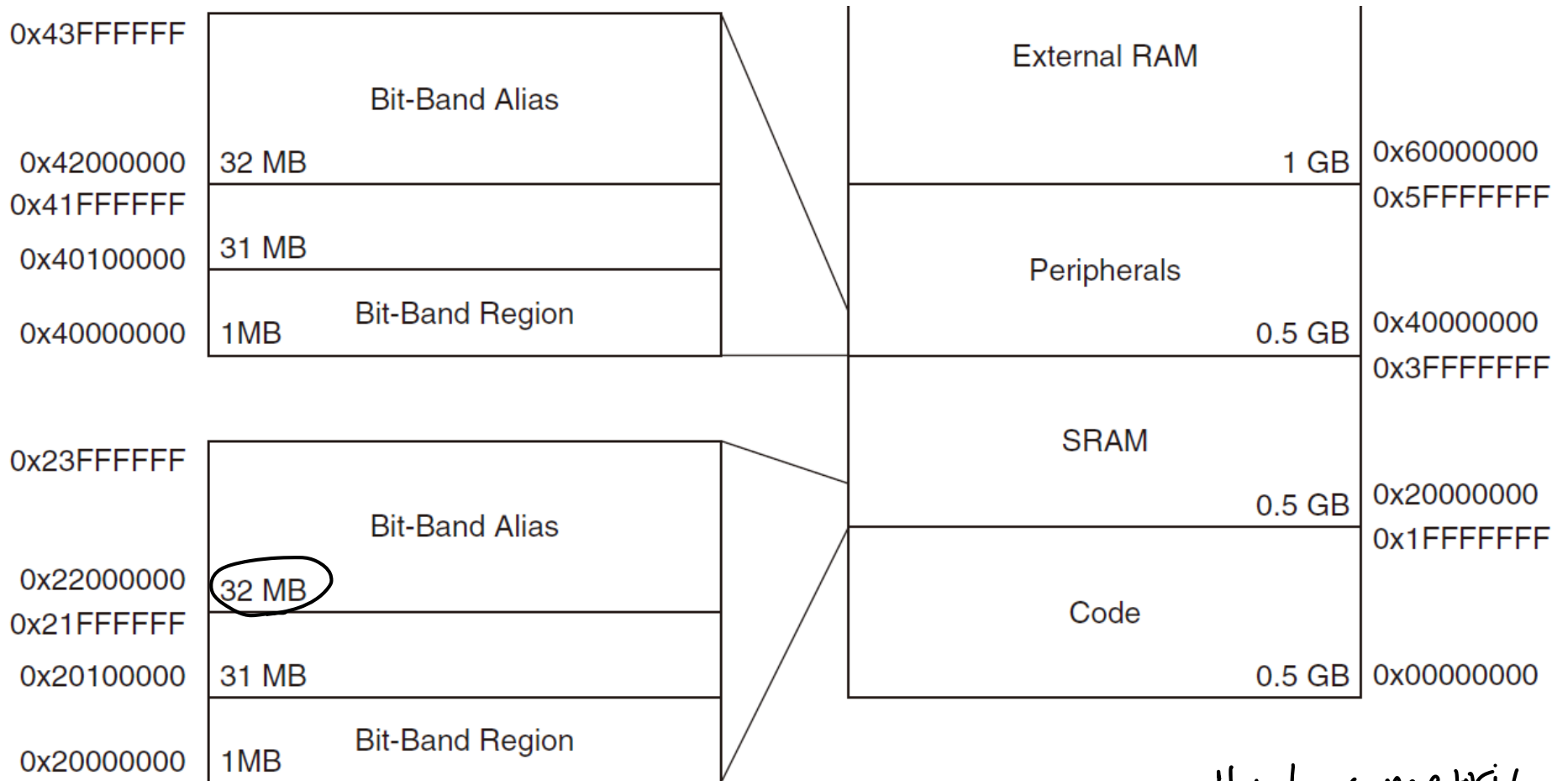
# Bit-Band Operations

Cortex-M

Bus matrix

- Bit-band operation support allows a single load/store (read/write) operation to access a single data bit.
- Bit-band regions:
  - The first 1 MB of the SRAM region
  - The first 1 MB of the peripheral region
- They can be accessed via a separate memory region called the bit-band alias.
- Bit Banding done transparently by bus matrix.

# Bit-Band Region and Bit-Band Alias



*transparently bus metric*

# Bit-Band Region and Bit-Band Alias

Bit-Band Region	Aliased Equivalent
0x20000000 bit[0]	0x22000000 bit[0]
0x20000000 bit[1]	0x22000004 bit[0]
0x20000000 bit[2]	0x22000008 bit[0]
...	...
0x20000000 bit[31]	0x2200007C bit[0]
0x20000004 bit[0]	0x22000080 bit[0]
...	...
0x20000004 bit[31]	0x220000FC bit[0]
...	...
0x200FFFFC bit[31]	0x23FFFFFC bit[0]

**Remapping of Bit-Band Addresses in  
SRAM Region**

Bit-Band Region	Aliased Equivalent
0x40000000 bit[0]	0x42000000 bit[0]
0x40000000 bit[1]	0x42000004 bit[0]
0x40000000 bit[2]	0x42000008 bit[0]
...	...
0x40000000 bit[31]	0x4200007C bit[0]
0x40000004 bit[0]	0x42000080 bit[0]
...	...
0x40000004 bit[31]	0x420000FC bit[0]
...	...
0x00FFFFC bit[31]	0x43FFFFFC bit[0]

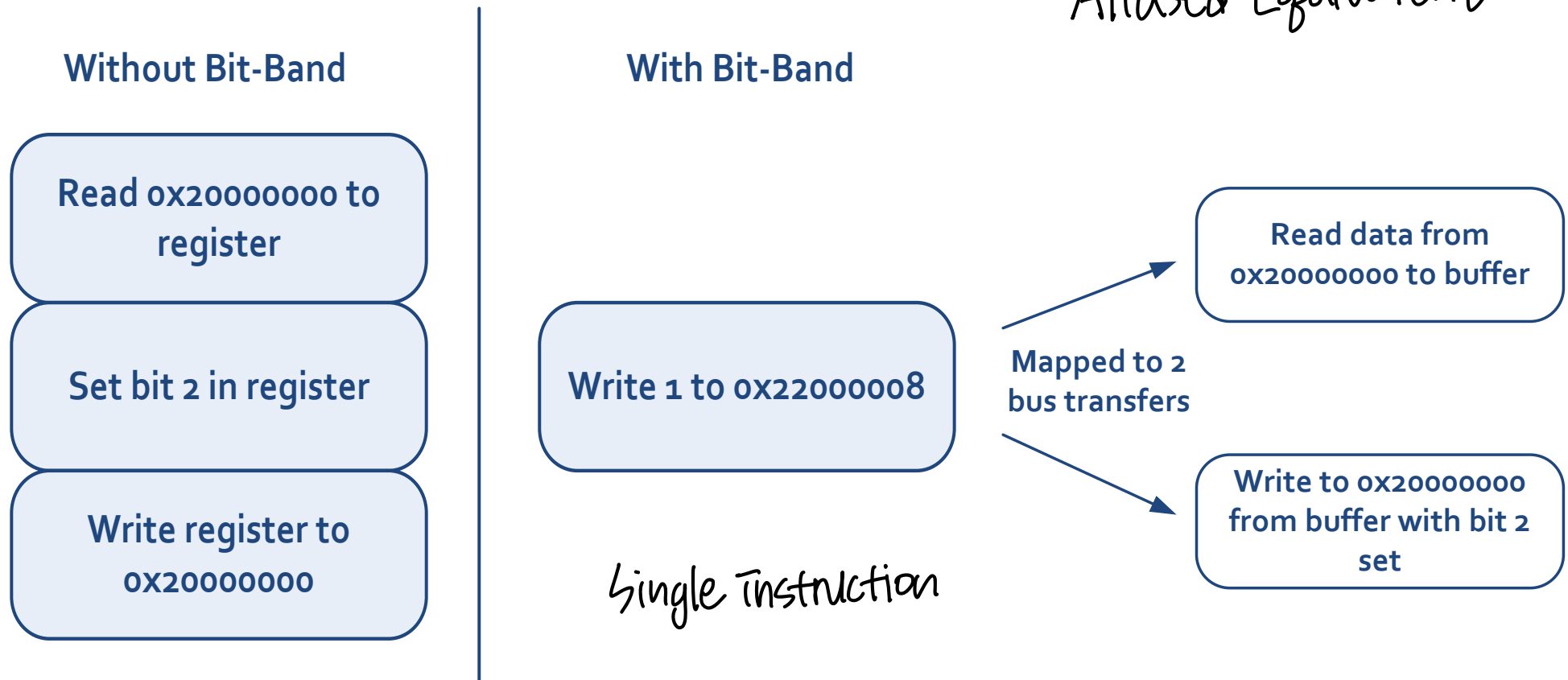
**Remapping of Bit-Band Addresses in  
Peripheral Memory Region**

# Write to Bit-Band Alias

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

- Write:

→ 0x20000008[0]  
Aliased Equivalent



# Write to Bit-Band Alias

- The assembler sequence to
- Write:

- Without Bit-Band:

```
LDR    R0, =0x20000000 ; Setup address
LDR    R1, [R0]         ; Read
ORR.W  R1, #0x4         ; Modify bit
STR    R1, [R0]         ; Write back result
```

- For write operations, the written bit data is shifted to the required bit position, and a READ-MODIFY-WRITE is performed.

- With Bit-Band:

```
LDR    R0, =0x22000008 ; Setup address (bit-band alias region)
MOV    R1, #1          ; Setup data
STR    R1, [R0]        ; Write
```

# Read from the Bit-Band Alias

- To get bit 2 in word data in address 0x20000000:
- Read:

## Without Bit-Band

Read 0x20000000 to register

Shift bit 2 to LSB and mask other bits

## With Bit-Band

Read from 0x22000008

Mapped to 1 bus transfers

Read data from 0x20000000 and extract bit 2 to register



# Read from the Bit-Band Alias

- The assembler sequence to read:

- Read:

- Without Bit-Band:

```
LDR    R0, =0x20000000 ; Setup address
```

```
LDR    R1, [R0]        ; Read
```

```
UBFX.W R1, R1, #2, #1   ; Extract bit[2]
```

- For read operations, the word is read and the chosen bit location is shifted to the LSB of the read return data.

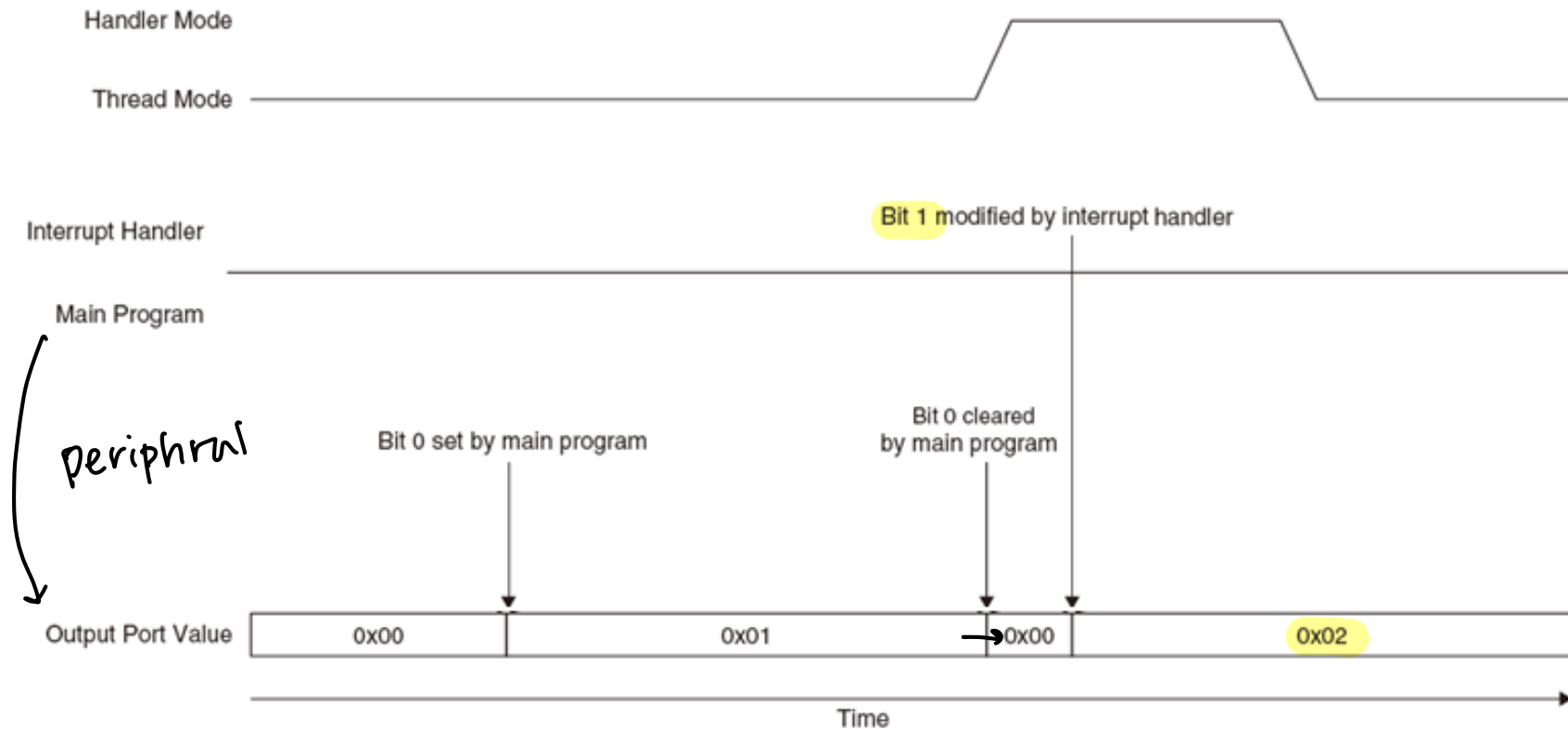
- With Bit-Band:

```
LDR    R0, =0x22000008 ; Setup address (bit-band alias region)
```

```
LDR    R1, [R0]        ; Read
```

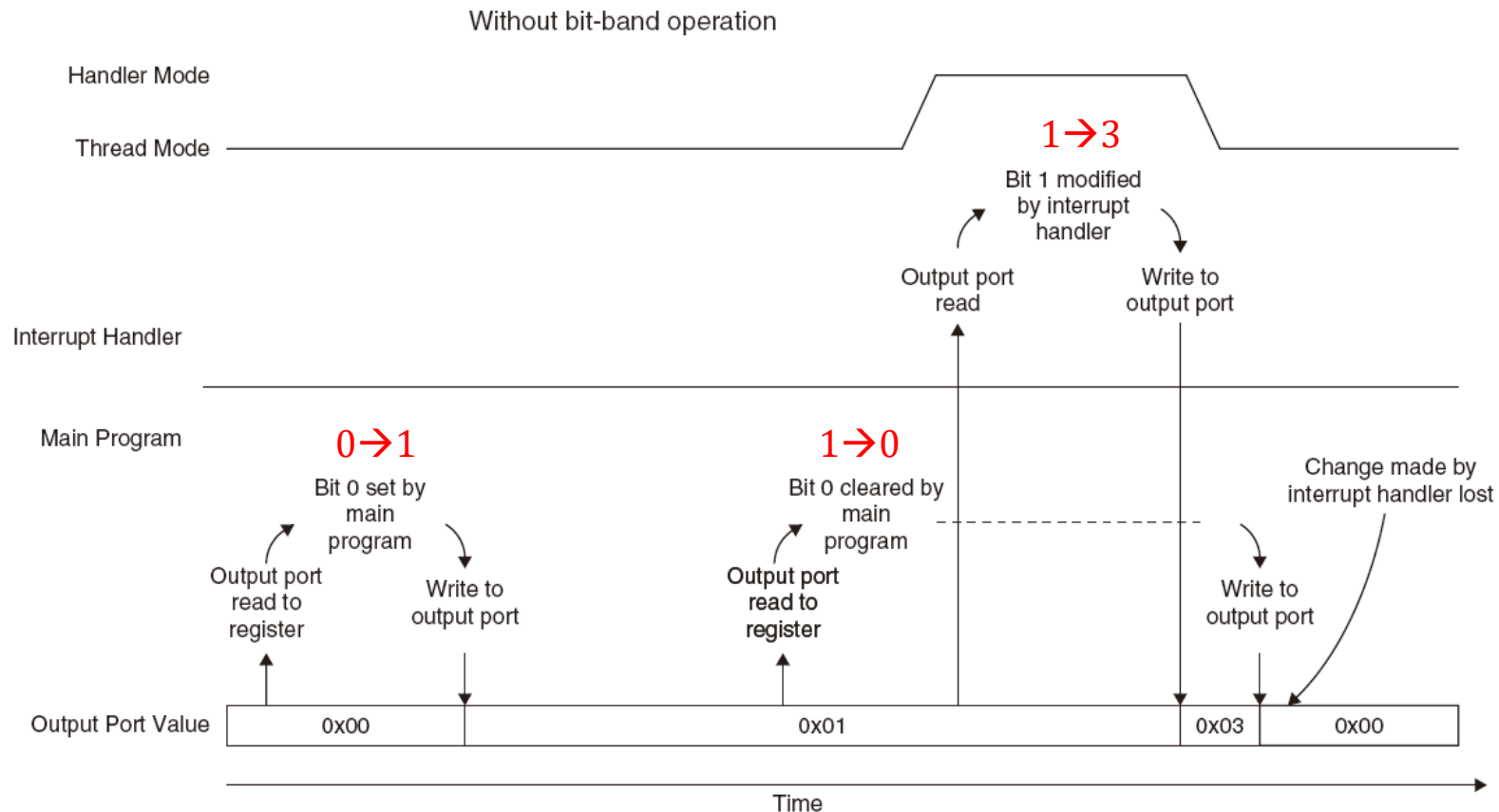
# Advantages of using bit-band operations:

- Faster bit operations with fewer instructions
- Prevent a race condition problem in bit modifications



# Advantages of using bit-band operations:

- Faster bit operations with fewer instructions
- Prevent a race condition problem in bit modifications

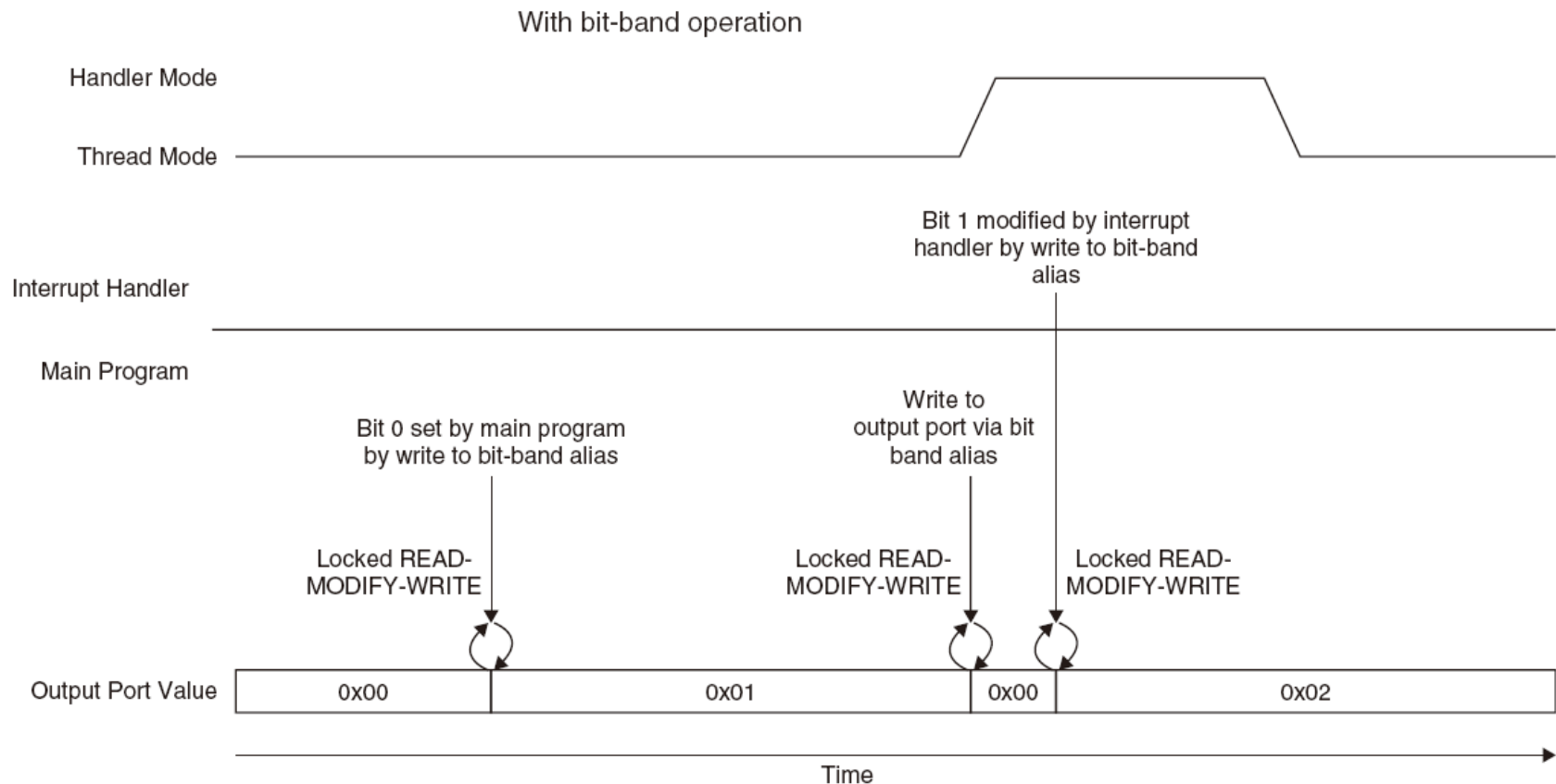


Data Are Lost When an Exception Handler Modifies a Shared Memory Location

3 Bit modification

# Advantages of using bit-band operations:

- Faster bit operations with fewer instructions
- Prevent a race condition problem in bit modifications



**Data Loss Prevention with Locked Transfers Using the Bit-Band Feature**

# Bit-Band Operations in C

- There is no native support of bit-band operation in most C compilers
  - The simplest solution is to separately declare the address and the bit-band alias of a memory location.

```
#define DEVICE_REG0      *((volatile unsigned long *) (0x40000000))
#define DEVICE_REG0_BIT0 *((volatile unsigned long *) (0x42000000))
#define DEVICE_REG0_BIT1 *((volatile unsigned long *) (0x42000004))
...
DEVICE_REG0 = 0xAB; // Accessing the hardware register by normal
                    // address

...
DEVICE_REG0 = DEVICE_REG0 | 0x2; // Setting bit 1 without using
                                bitwise // bitband feature

...
DEVICE_REG0_BIT1 = 0x1; // Setting bit 1 using bitband feature
                        simple // via the bit band alias address
                        assignment
```

# Bit-Band Operations in C

- C macros can make accessing the bit-band alias easier

```
// Convert bit band address and bit number into
// bit band alias address
#define BITBAND(addr,bitnum) ((addr & 0xF0000000)+0x20000000+((addr &
                                0xFFFF)<<5)+(bitnum <<2))

// Convert the address as a pointer
#define MEM_ADDR(addr) *((volatile unsigned long *) (addr))

...
MEM_ADDR(DEVICE_REG0) = 0xAB; // Accessing the hardware
                                // register by normal address

...
// Setting bit 1 without using bitband feature
MEM_ADDR(DEVICE_REG0) = MEM_ADDR(DEVICE_REG0) | 0x2;

...
// Setting bit 1 with using bitband feature
MEM_ADDR(BITBAND(DEVICE_REG0,1)) = 0x1;
```

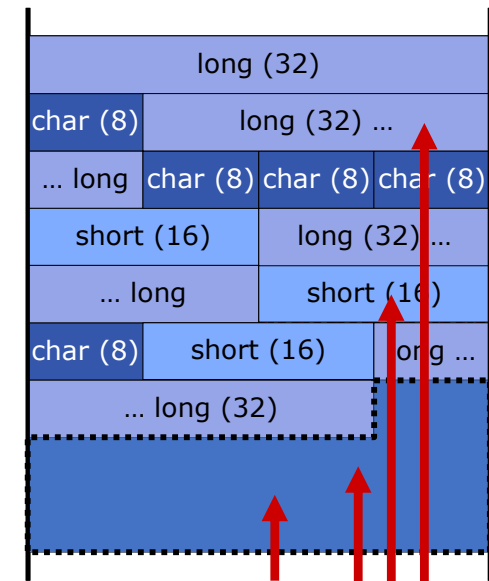
# Miscellaneous features

# Unaligned Transfers

- The Cortex-M3 supports unaligned transfers on single accesses.
- Data memory accesses can be defined as aligned or unaligned.

```
struct {
    long a;
    char b;
    long c;
    char d[3];
    short e;
    long f;
    short g;
    char h;
    short i;
    int j;
} __attribute__((__packed__))
```

variables are aligned to the multiple of its size



Free space for the rest of the application



# Unaligned Transfers

- When unaligned transfers are used, they are actually converted into multiple aligned transfers by the processor's bus interface unit
  - Transparent to application programmers.
  - It takes more clock cycles for a single data access
- There are a number of limitations:
  - Not supported in Load/Store multiple instructions.
  - Stack operations (PUSH/POP) must be aligned.
  - Exclusive accesses must be aligned.
  - Unaligned transfers are not supported in bit-band operations.

# Unaligned Transfers

- A word transfer on a bus of 32-bit width

	Byte 3	Byte 2	Byte 1	Byte 0
Address N+4				
Address N	[31:24]	[23:16]	[15:8]	[7:0]

**Aligned Transfer**

	Byte 3	Byte 2	Byte 1	Byte 0
Address N+4				[31:24]
Address N	[23:16]	[15:8]	[7:0]	

**Unaligned Transfer Example 1**

	Byte 3	Byte 2	Byte 1	Byte 0
Address N+4			[31:24]	[23:16]
Address N	[15:8]	[7:0]		

**Unaligned Transfer Example 2**

	Byte 3	Byte 2	Byte 1	Byte 0
Address N+4		[31:24]	[23:16]	[15:8]
Address N	[7:0]			

**Unaligned Transfer Example 3**

# Unaligned Transfers

- A halfword transfer on a bus of 32-bit width

	Byte 3	Byte 2	Byte1	Byte 0
Address N+4				
Address N			[15:8]	[7:0]

**Aligned Transfer Example 1**

	Byte 3	Byte 2	Byte 1	Byte 0
Address N+4				
Address N	[15:8]	[7:0]		

**Aligned Transfer Example 2**

	Byte 3	Byte 2	Byte1	Byte 0
Address N+4				
Address N		[15:8]	[7:0]	

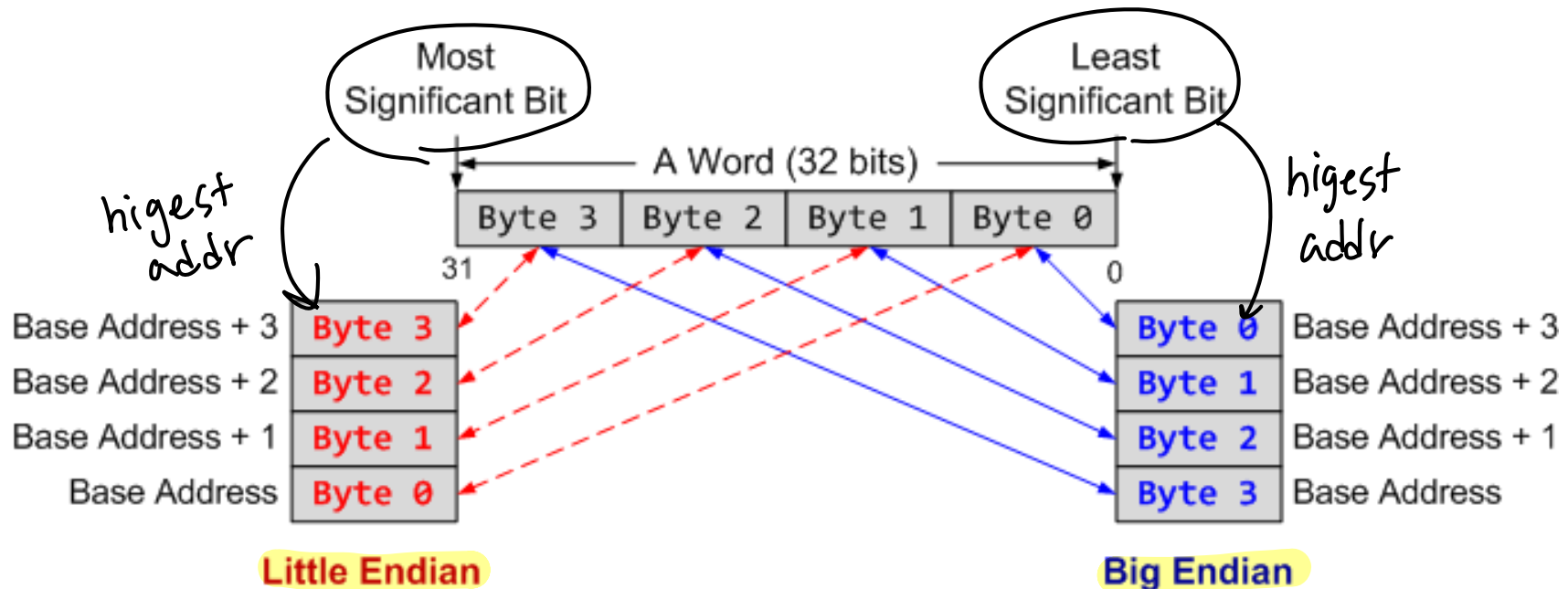
**Unaligned Transfer Example 1**

	Byte 3	Byte 2	Byte 1	Byte 0
Address N+4				[15:8]
Address N	[7:0]			

**Unaligned Transfer Example 2**

# Endian Mode

- The Cortex-M3 supports both little endian(recommended) and big endian modes.
  - However, the supported memory type also depends on the design of the rest of the microcontroller (bus connections, memory controllers, peripherals, and so on).
  - Make sure that you check your microcontroller datasheets in detail before developing your software



[will not be in exam]

## Exclusive Accesses (LDREX,STREX)

- A lock is usually used to safely share resources among tasks and handlers
- Before using a resource, a task should acquire a lock for the resource.
  - Now, other tasks are blocked from using the resources, because they cannot acquire the lock.
- After using the resource, the task should release the lock.

# Exclusive Access Instructions

- Exclusive access instructions can check the availability of hardware resource (maybe small memory block for a lock variable) using an exclusive monitor
  - LDREX (word)
  - LDREXB (byte)
  - LDREXH (half word)
  - STREX (word)
  - STREXB (byte)
  - STREXH (half word)

# Exclusive Access Operations

