Chapter 5
Memory Access

Dr. Yifeng Zhu
Electrical and Computer Engineering
University of Maine

Spring 2015

Course Objective

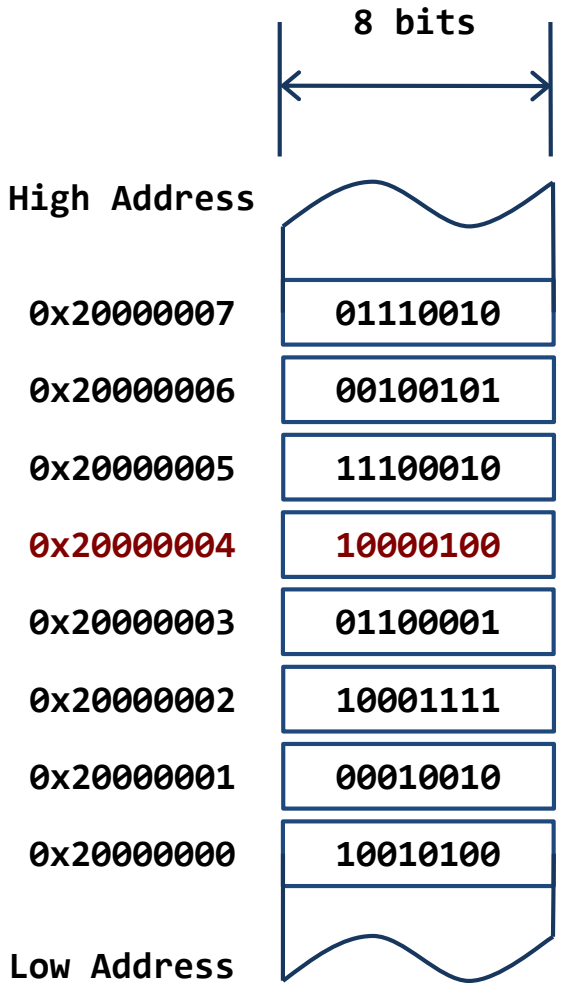
- How data is organized in memory?
  - Big Endian vs Little Endian
- How data is addressed?
  - Pre-index
  - Post-index
  - Pre-index with update

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 array of bytes, each of which is addressable.
  - The byte stored at the memory address `0x20000004` is `10000100`

`0b10000100` → `0x84` → `132`

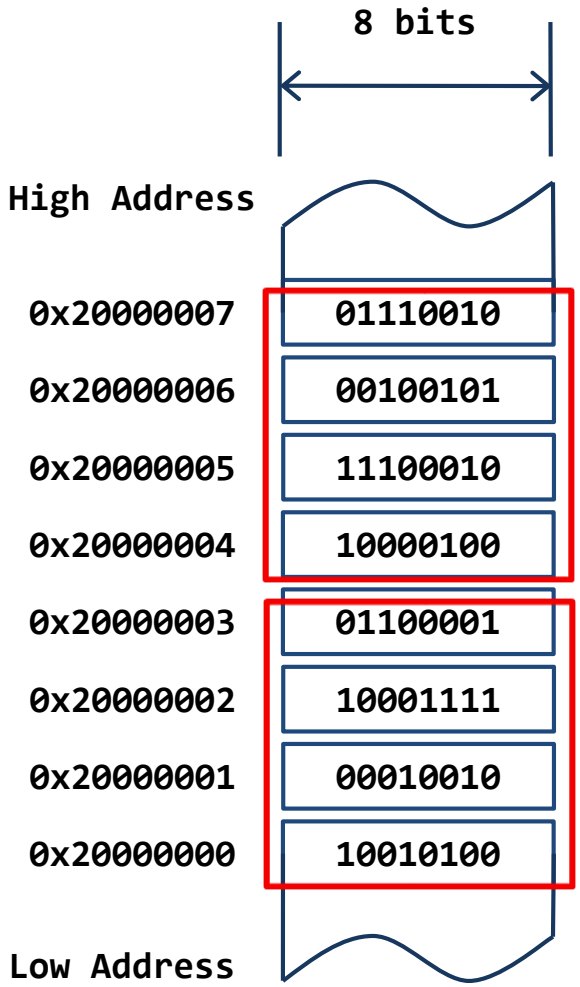
`Binary` `Hexadecimal` `Decimal`



Logic View of Memory

- When we refer to memory locations by address, we can only do so in units of bytes, halfwords or words
- Words
  - 32 bits = 4 bytes = 1 word
  - We have two words:
    - 0x20000000
    - 0x20000004
  - Can you store a word anywhere? **NO.**
  - A word can only be stored at an address that's divisible by 4.
  - Memory address of a word is the lowest address of all four bytes in that word.

Word-address mod 4 = 0

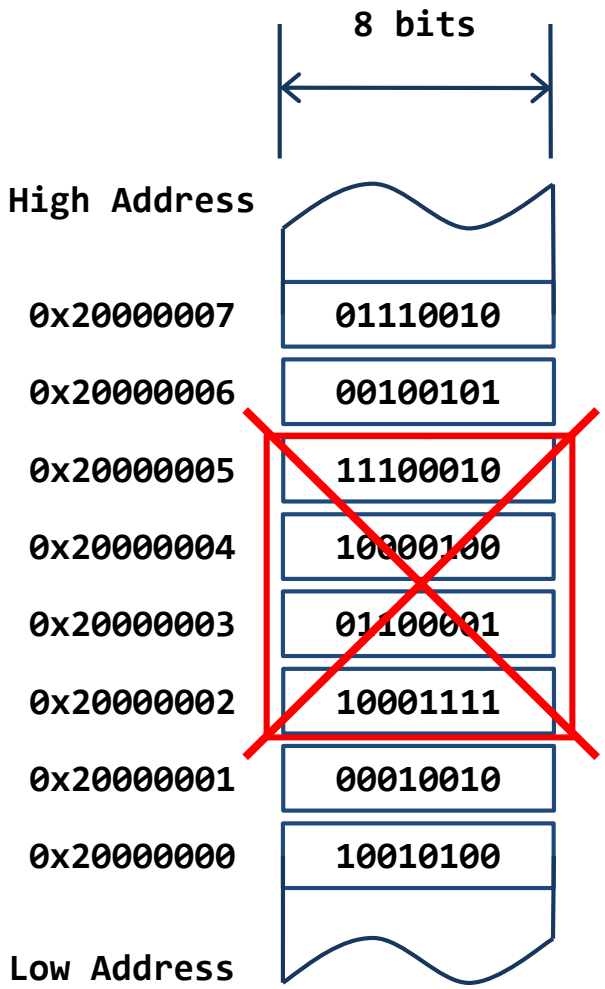


Logic View of Memory

Cannot store a word at address 0x20000002.

- When we refer to memory locations by address, we can only do so in units of bytes, halfwords or words
- Words
  - 32 bits = 4 bytes = 1 word
  - We have two words:
    - 0x20000000
    - 0x20000004
  - Can you store a word anywhere? **NO.**
  - A word can only be stored at an address that's divisible by 4.
  - Memory address of a word is the lowest address of all four bytes in that word.

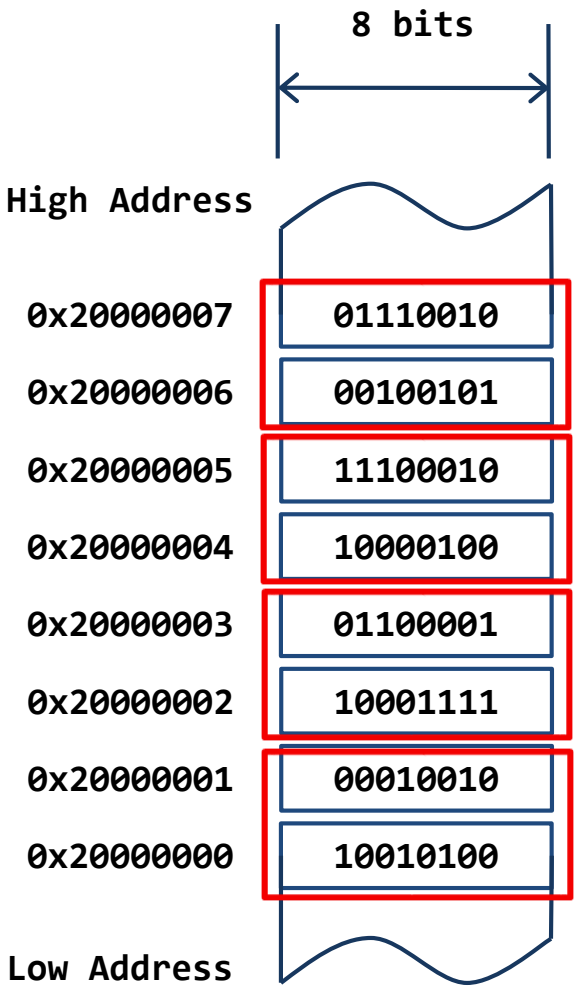
Word-address mod 4 = 0



Logic View of Memory

- When we refer to memory locations by address, we can only do so in units of bytes, halfwords or words
- Halfwords
  - 16 bits = 2 bytes = 1 halfword
  - We have four halfwords:
    - 0x20000000
    - 0x20000002
    - 0x20000004
    - 0x20000006
  - Can you store a halfword anywhere? **NO.**
  - A halfword can only be stored at an address that's divisible by 2.
  - Memory address of a halfword is the lowest address of its two bytes.

```
Halfword-address mod 2 = 0
```

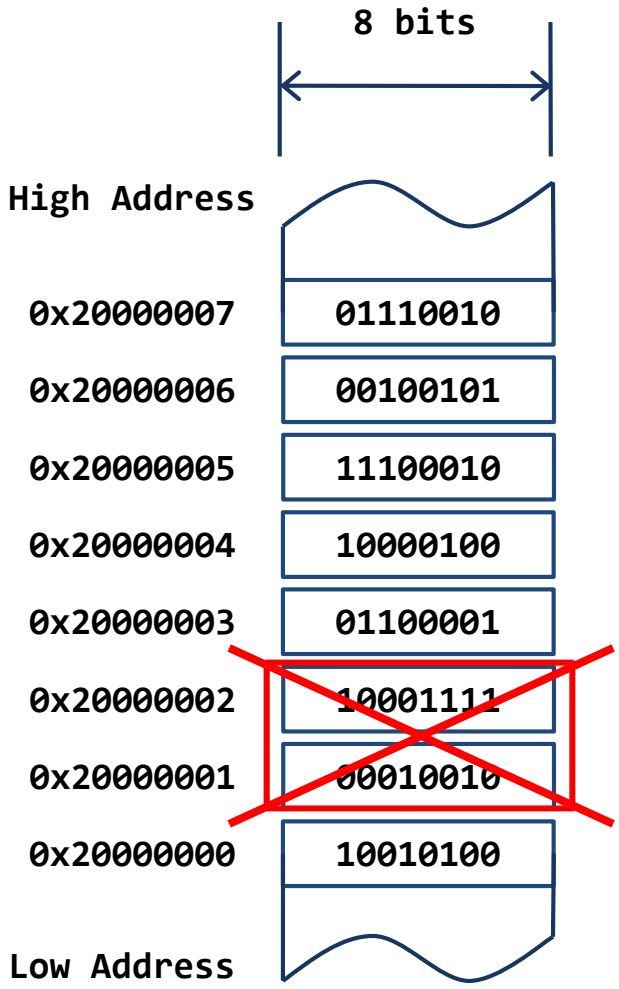


Logic View of Memory

Cannot store a halfword at address 0x20000001.

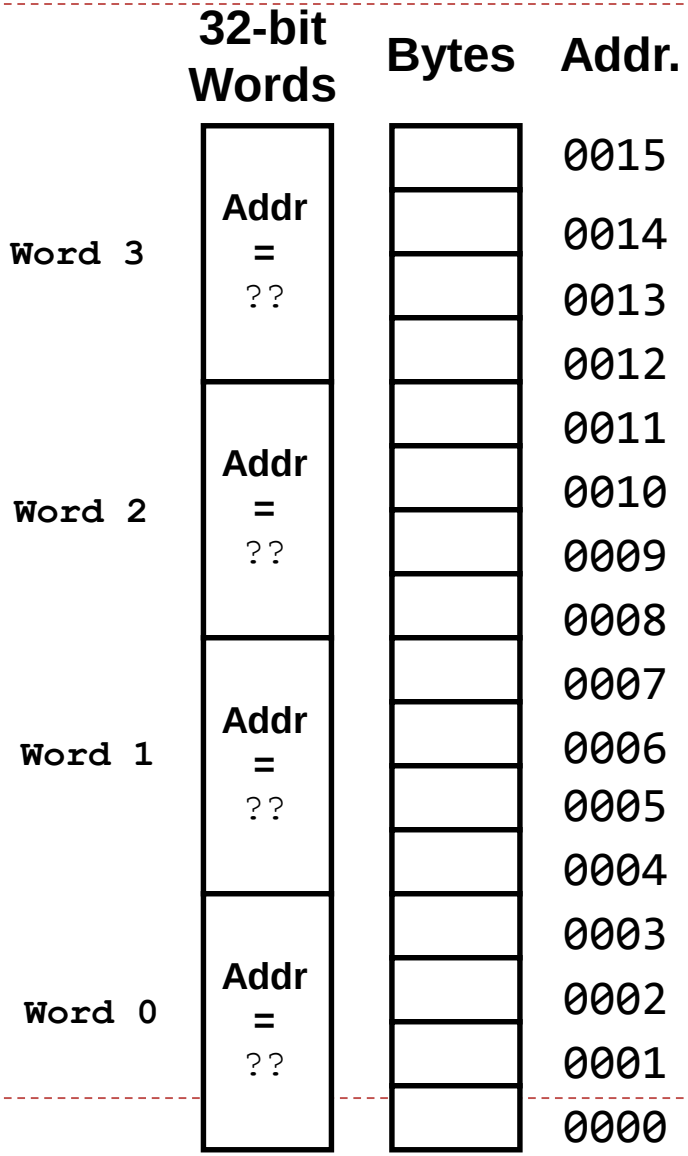
- When we refer to memory locations by address, we can only do so in units of bytes, halfwords or words
- Halfwords
  - 16 bits = 2 bytes = 1 halfword
  - We have four halfwords:
    - 0x20000000
    - 0x20000002
    - 0x20000004
    - 0x20000006
  - Can you store a halfword anywhere? **NO.**
  - A halfword can only be stored at an address that's divisible by 2.
  - Memory address of a halfword is the lowest address of its two bytes.

```
Halfword-address mod 2 = 0
```



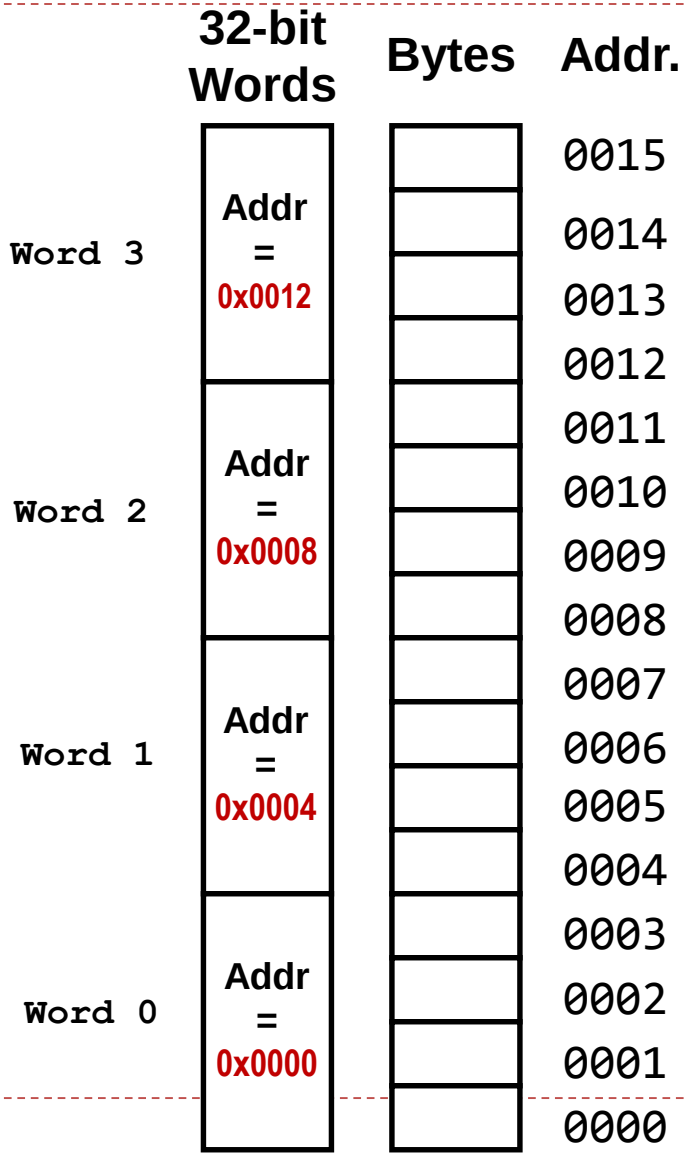
Quiz

What are the memory address of these four words?



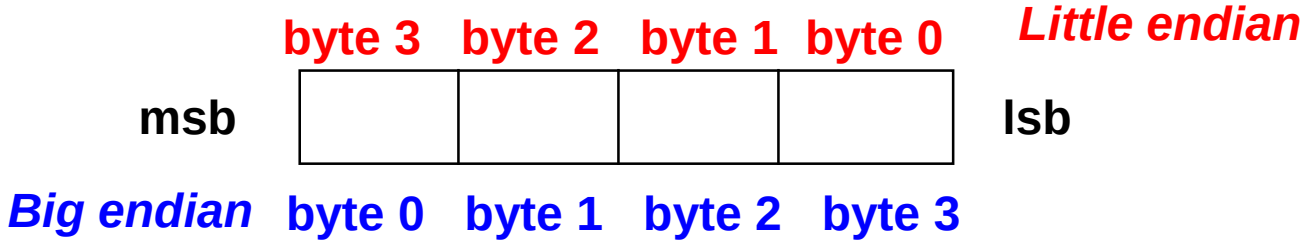
Quiz (Answer)

What are the memory address of these four words?



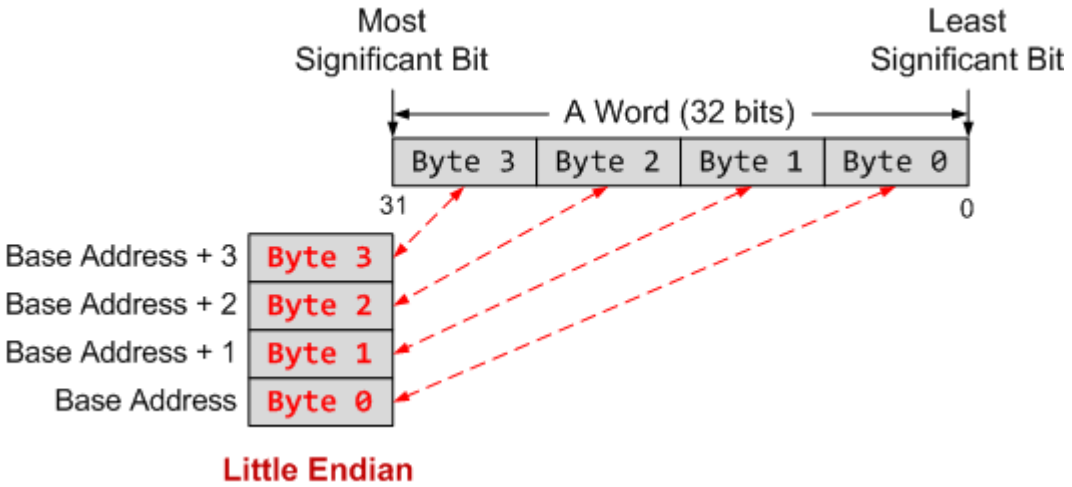
Endianess

- Big Endian: address of most significant byte = word address (xx00 = Big End of word)
- Little Endian: address of least significant byte = word address (xx00 = Little End of word)
- ARM is *Little Endian by default*. However it can be made Big Endian by configuration.



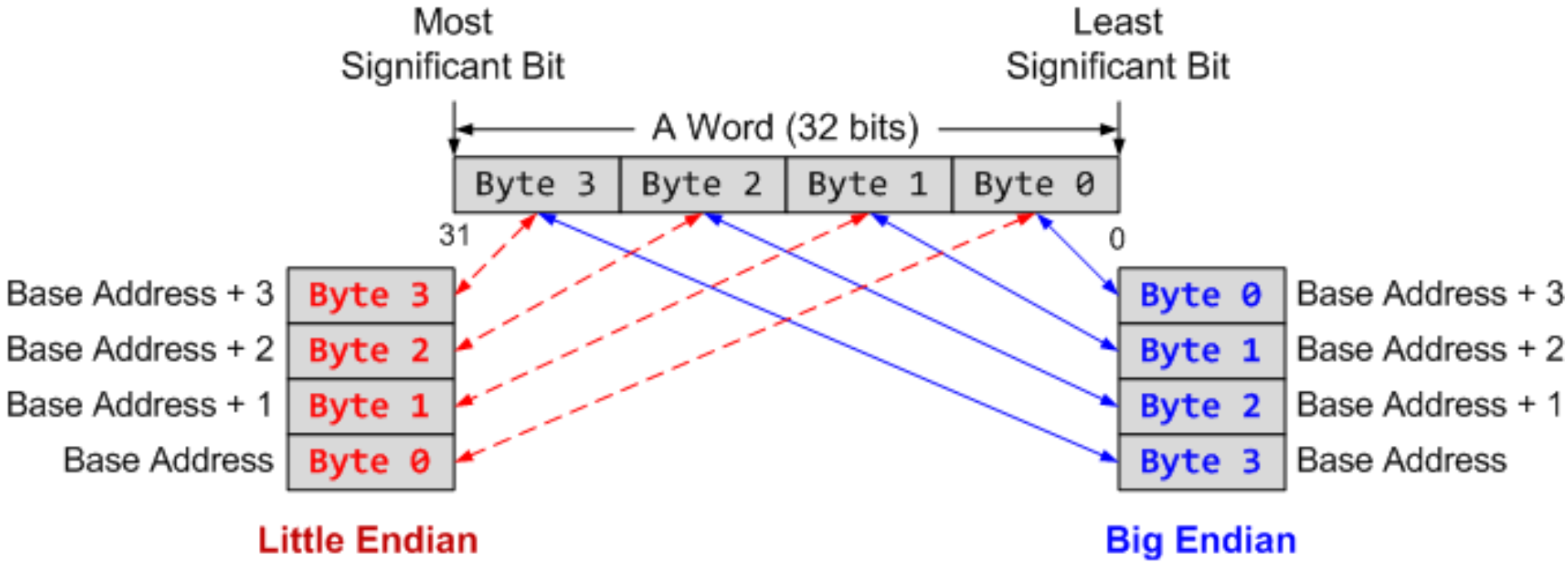
Endianess

- Little Endian: address of least significant byte = word address (xx00 = Little End of word)
- Big Endian: address of most significant byte = word address (xx00 = Big End of word)
- ARM is *Little Endian by default*. However it can be made Big Endian by configuration.



Endianess

- Little Endian: The little end comes first.
- Big Endian: The big end comes first.



Example

If big endianess is used

The word stored at address 0x20008000 is

0xEE8C90A7

Memory Address	Memory Data
0x20008003	0xA7
0x20008002	0x90
0x20008001	0x8C
0x20008000	0xEE

Example

If little endianess is used

The word stored at address 0x20008000 is

0xA7908CEE

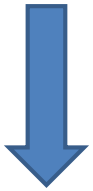
Endian only specifies byte order, not bit order in a byte!

Memory Address	Memory Data
0x20008003	0xA7
0x20008002	0x90
0x20008001	0x8C
0x20008000	0xEE

Load-Modify-Store

C statement

```
x = x + 1;
```



```
; Assume the memory address of x is stored in r1

LDR r0, [r1]      ; load value of x from memory
ADD r0, r0, #1    ; x = x + 1
STR r0, [r1]      ; store x into memory
```

Load Instructions

- **LDR rt, [rs]**
  - fetch data from memory into register rt.
  - The memory address is specified in register rs.
  - For Example:

```
; Assume r0 = 0x82000004
; Load a word:
LDR r1, [r0]                    ; r1 = Memory.word[0x82000004]
```

Store Instructions

- **STR rt, [rs]:**
  - save data in register rt into memory
  - The memory address is specified in a base register rs.
  - For Example:

```
; Assume r0 = 0x82000004
; Store a word
STR r1, [r0]                    ; Memory.word[0x82000004] = r1
```

Single register data transfer

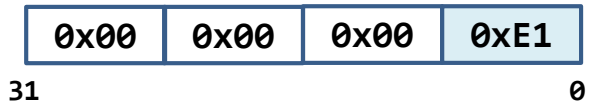
LDR	Load Word
LDRB	Load Byte
LDRH	Load Halfword
LDRSB	Load Signed Byte
LDRSH	Load Signed Halfword

STR	Store Word
STRB	Store Lower Byte
STRH	Store Lower Halfword

Load a Byte, Half-word, Word

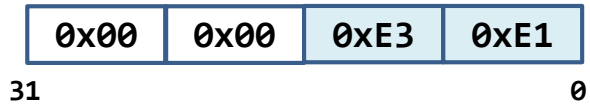
Load a Byte

```
LDRB r1, [r0]
```



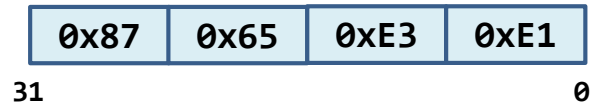
Load a Halfword

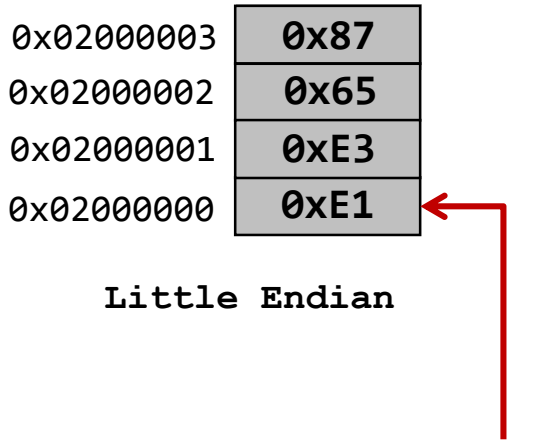
```
LDRH r1, [r0]
```



Load a Word

```
LDR r1, [r0]
```





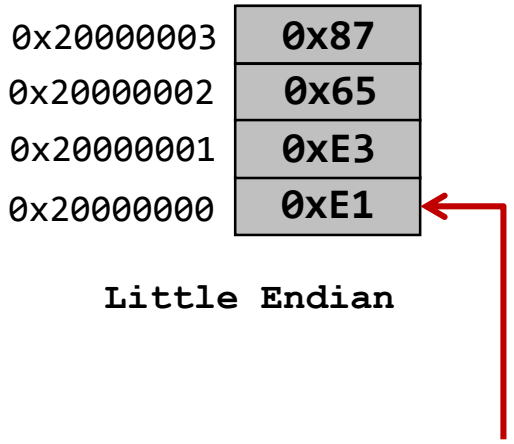
Sign Extension

Load a Signed Byte

```
LDRSB r1, [r0]
```

Load a Signed Halfword

```
LDRSH r1, [r0]
```



Facilitate subsequent 32-bit signed arithmetic!

Address

- Address accessed by LDR/STR is specified by a base register plus an offset
- For word and unsigned byte accesses, offset can be
  - An unsigned 12-bit immediate value (i.e. 0 - 4095 bytes).

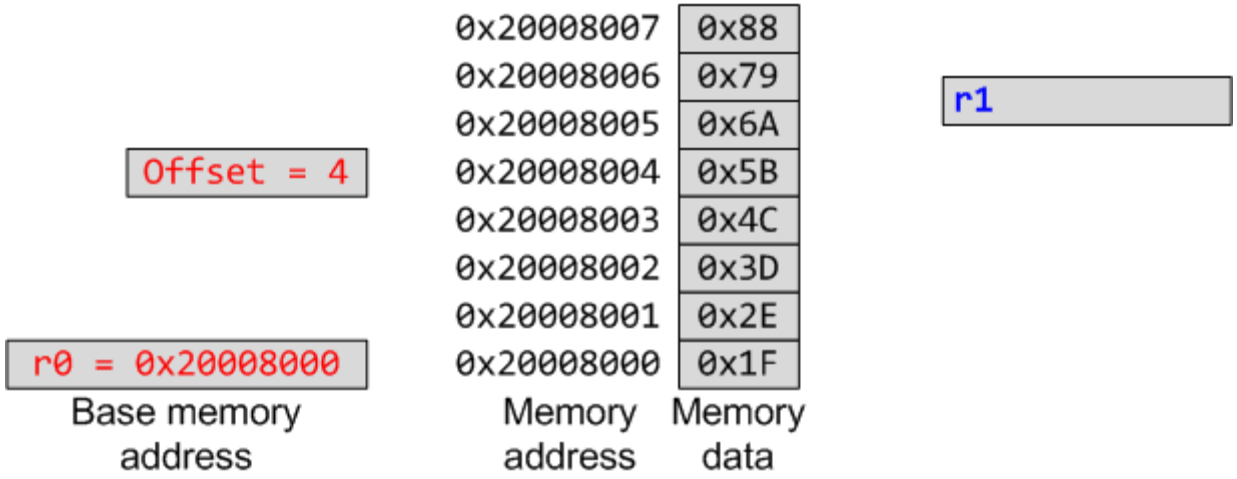
    ```
    LDR r0,[r1,#8]
    ```
  - A register, optionally shifted by an immediate value

    ```
    LDR r0,[r1,r2]
    LDR r0,[r1,r2,LSL#2]
    ```
- This can be either added or subtracted from the base register:

  ```
  LDR r0,[r1,#-8]
  LDR r0,[r1,-r2]
  LDR r0,[r1,-r2,LSL#2]
  ```
- For halfword and signed halfword / byte, offset can be:
  - An unsigned 8 bit immediate value (ie 0-255 bytes).
  - A register (unshifted).
- Choice of *pre-indexed* or *post-indexed* addressing

Pre-index

Pre-Index: LDR r1, [r0, #4]

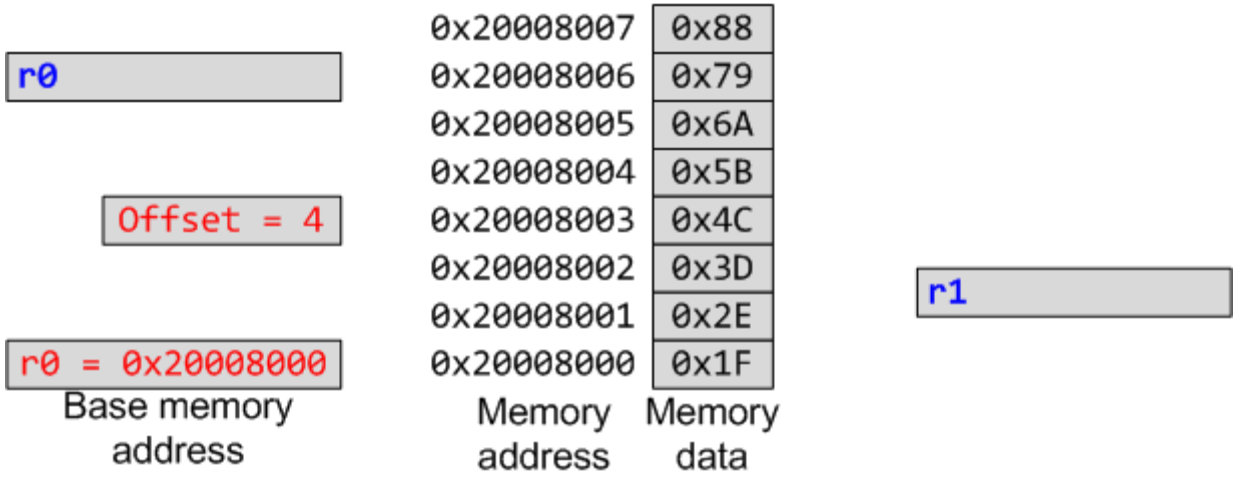


Pre-index

Pre-Index: LDR r1, [r0, #4]

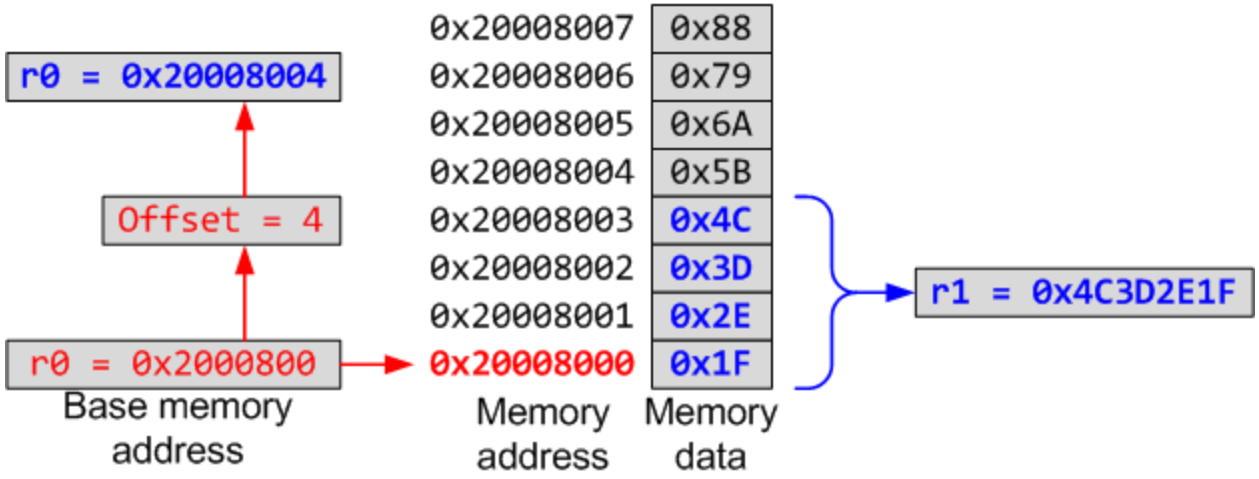
Post-index

Post-Index: LDR r1, [r0], #4



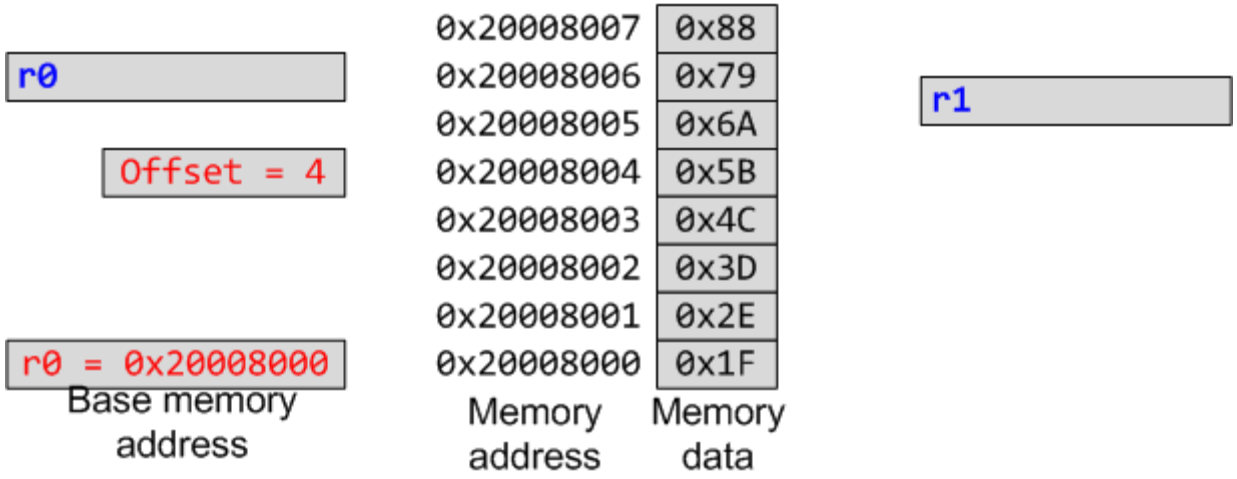
Post-index

Post-Index: LDR r1, [r0], #4



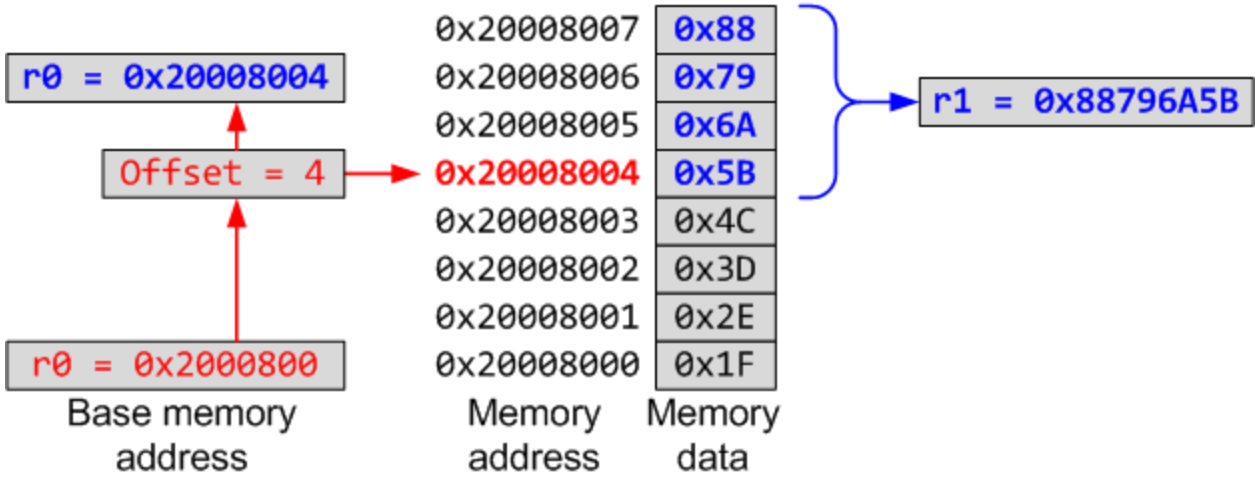
Pre-index with Updates

Pre-Index with Update: LDR r1, [r0, #4]!



Pre-index with Updates

Pre-Index: LDR r1, [r0, #4]



Summary of Pre-index and Post-index

Index Format	Example	Equivalent
Pre-index	LDR r1, [r0, #4]	r1 ← memory[r0 + 4], r0 is unchanged
Pre-index with update	LDR r1, [r0, #4]!	r1 ← memory[r0 + 4] r0 ← r0 + 4
Post-index	LDR r1, [r0], #4	r1 ← memory[r0] r0 ← r0 + 4

Example

```
LDRH r1, [r0]
; r0 = 0x20008000
```

r1 before load

0x12345678

r1 after load

0x0000CDEF

Memory Address	Memory Data
0x20008003	0x89
0x20008002	0xAB
0x20008001	0xCD
0x20008000	0xEF

Example

```
LDSB r1, [r0]
; r0 = 0x20008000
```

r1 before load

0x12345678

r1 after load

0xFFFFFFEF

Memory Address	Memory Data
0x20008003	0x89
0x20008002	0xAB
0x20008001	0xCD
0x20008000	0xEF

Example

```
STR r1, [r0], #4
; r0 = 0x20008000,  r1=0x76543210
```

r0 before store

0x20008000

r0 after store

Memory Address	Memory Data
0x20008007	0x00
0x20008006	0x00
0x20008005	0x00
0x20008004	0x00
0x20008003	0x00
0x20008002	0x00
0x20008001	0x00
0x20008000	0x00

Example

```
STR r1, [r0], #4
; r0 = 0x20008000,  r1=0x76543210
```

r0 before store

0x20008000

r0 after store

0x20008004

Memory Address	Memory Data
0x20008007	0x00
0x20008006	0x00
0x20008005	0x00
0x20008004	0x00
0x20008003	0x76
0x20008002	0x54
0x20008001	0x32
0x20008000	0x10

Example

```
STR r1, [r0, #4]
; r0 = 0x20008000,  r1=0x76543210
```

r0 before the store

0x20008000

r0 after the store

Memory Address	Memory Data
0x20008007	0x00
0x20008006	0x00
0x20008005	0x00
0x20008004	0x00
0x20008003	0x00
0x20008002	0x00
0x20008001	0x00
0x20008000	0x00

Example

```
STR r1, [r0, #4]
; r0 = 0x20008000,  r1=0x76543210
```

r0 before store

0x20008000

r0 after store

0x20008000

Memory Address	Memory Data
0x20008007	0x76
0x20008006	0x54
0x20008005	0x32
0x20008004	0x10
0x20008003	0x00
0x20008002	0x00
0x20008001	0x00
0x20008000	0x00

Example

```
STR r1, [r0, #4]!
; r0 = 0x20008000,  r1 = 0x76543210
```

r0 before store

0x20008000

r0 after store

Memory Address	Memory Data
0x20008007	0x00
0x20008006	0x00
0x20008005	0x00
0x20008004	0x00
0x20008003	0x00
0x20008002	0x00
0x20008001	0x00
0x20008000	0x00

Example

```
STR r1, [r0, #4]!
; r0 = 0x20008000,  r1=0x76543210
```

r0 before store

0x20008000

r0 after store

0x20008004

Memory Address	Memory Data
0x20008007	0x76
0x20008006	0x54
0x20008005	0x32
0x20008004	0x10
0x20008003	0x00
0x20008002	0x00
0x20008001	0x00
0x20008000	0x00

Example

If big endianess is used

```
LDR r11, [r0]
; r0 = 0x20008000
```

r11 before load

0x12345678

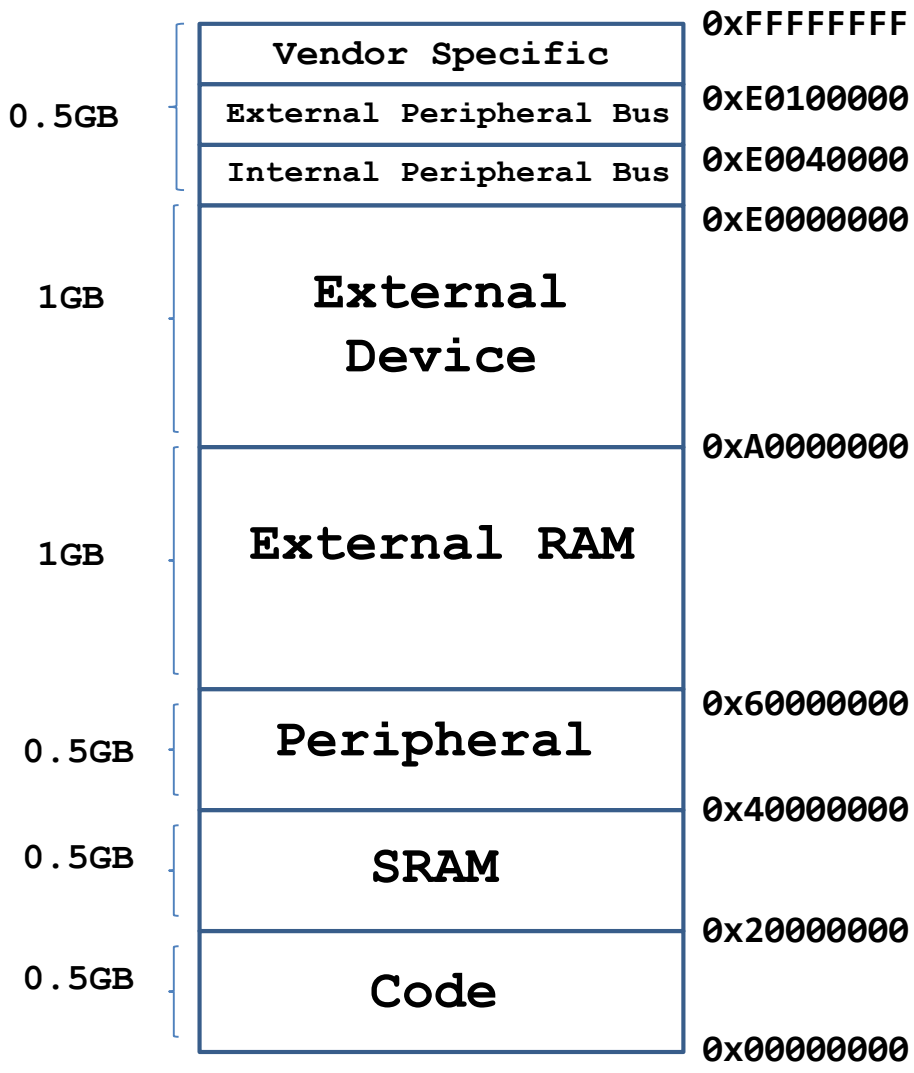
r11 after load

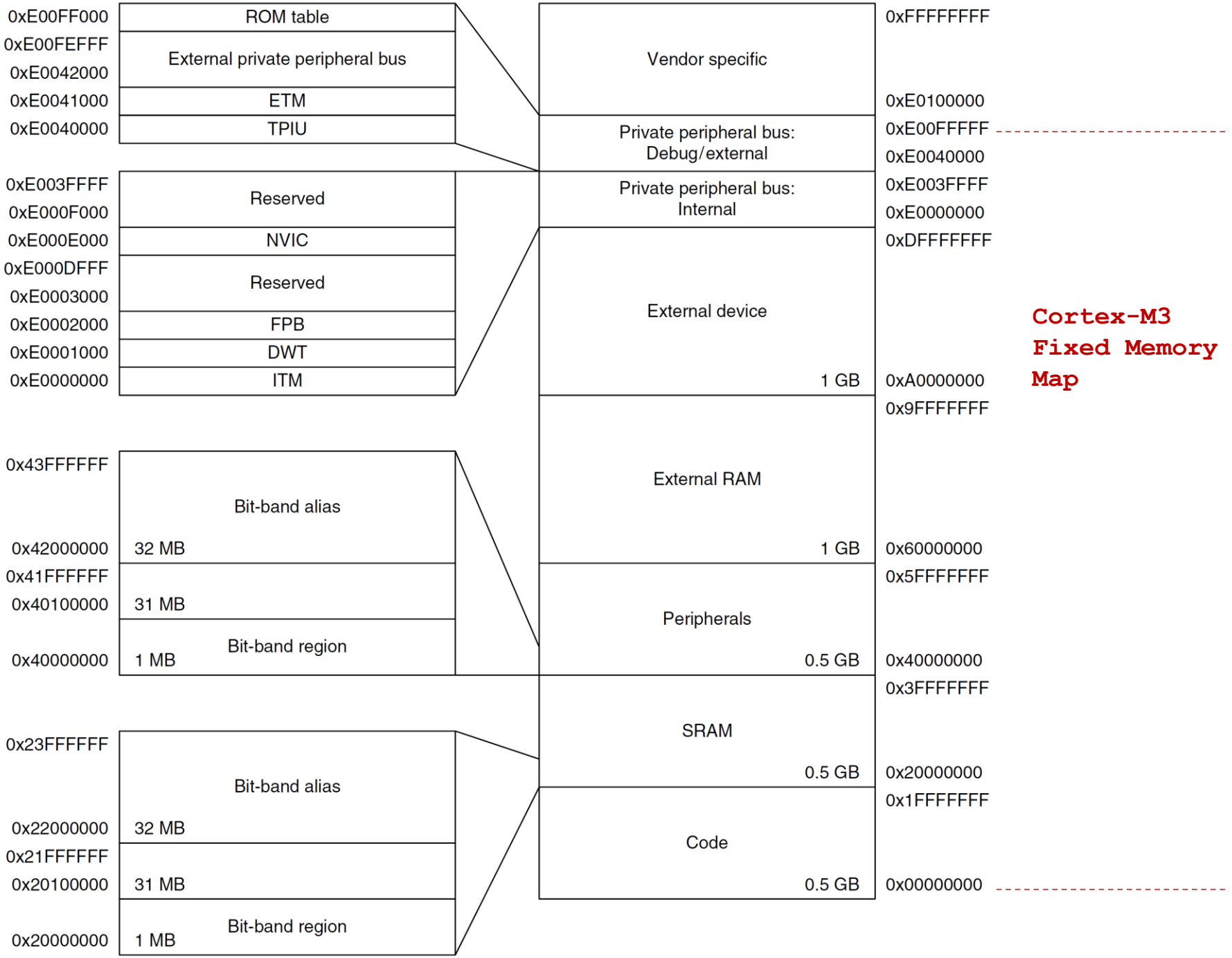
0xA7908CEE

Memory Address	Memory Data
0x20008003	0xEE
0x20008002	0x8C
0x20008001	0x90
0x20008000	0xA7

Cortex-M3 Memory Map

- 32-bit Memory Address
- 2^{32} bytes of memory space (4 GB)
- Harvard architecture: physically separated instruction memory and data memory

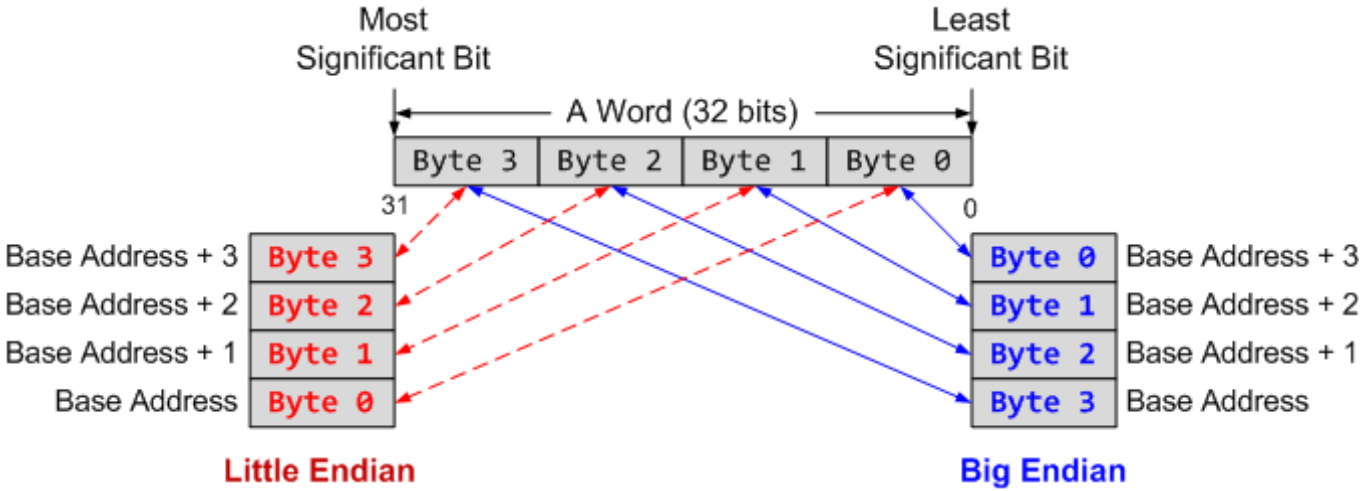




Cortex-M3 Fixed Memory Map

Summary

- Memory address is always in terms of bytes.
- How data is organized in memory?



- How data is addressed?

Addressing Format	Example	Equivalent
Pre-index	LDR r1, [r0, #4]	r1 ← memory[r0 + 4], r0 is unchanged
Pre-index with update	LDR r1, [r0, #4]!	r1 ← memory[r0 + 4] r0 ← r0 + 4
Post-Index	LDR r1, [r0], #4	r1 ← memory[r0] r0 ← r0 + 4