Embedded Systems with ARM Cortex-M3 Microcontrollers in Assembly Language and C

Chapter 5 Memory Access

Dr. Yifeng Zhu Electrical and Computer Engineering University of Maine

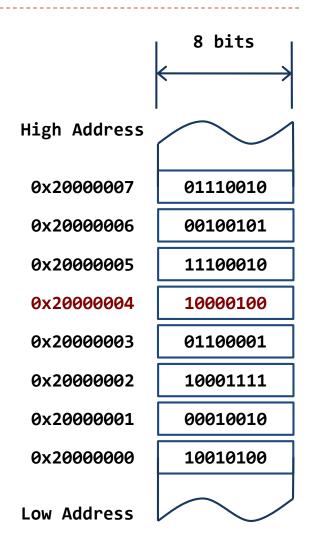
Spring 2015

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

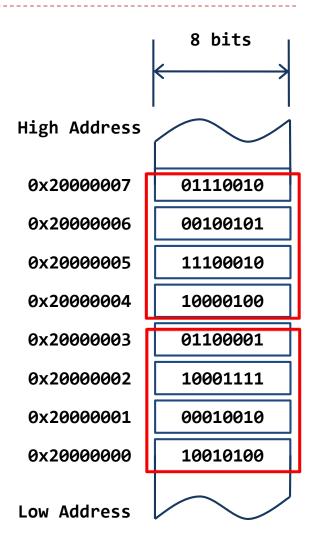
- By grouping bits together we can store more values
 - 8 bits = 1 byte
 - ▶ 16 bits = 2 bytes = 1 halfword
 - 32 bits = 4 bytes = 1 word
- From software perspective, memory is an array of bytes, each of which is addressable.
 - The byte stored at the memory address 0x20000004 is 10000100

0b10000100 → 0x84 → 132
Binary Hexadecimal Decimal



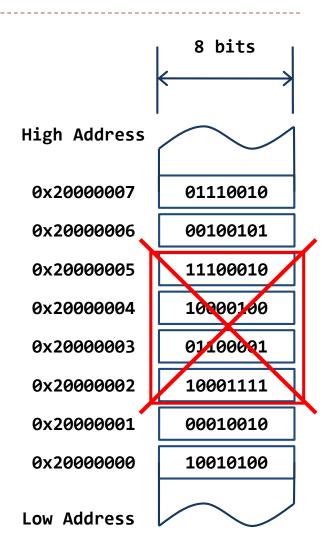
- When we refer to memory locations by address, we can only do so in units of bytes, halfwords or words
- Words
 - ▶ 32 bits = 4 bytes = I word
 - We have two words:
 - ▶ 0×20000000
 - ▶ 0x20000004
 - Can you store a word anywhere? NO.
 - A word can only be stored at an address that's divisible by 4.
 - Memory address of a word is the lowest address of all four bytes in that word.

Word-address mod 4 = 0



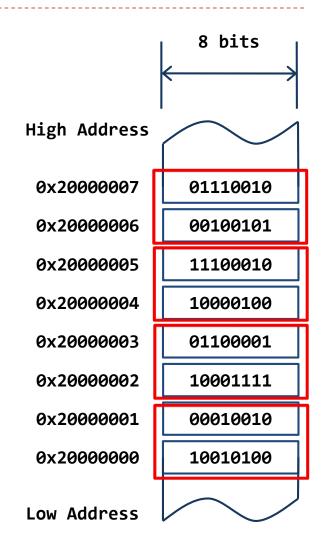
- When we refer to memory locations by address, we can only do so in units of bytes, halfwords or words
- Words
 - 32 bits = 4 bytes = I word
 - We have two words:
 - ▶ 0×20000000
 - ▶ 0x20000004
 - Can you store a word anywhere? NO.
 - A word can only be stored at an address that's divisible by 4.
 - Memory address of a word is the lowest address of all four bytes in that word.

Word-address mod 4 = 0



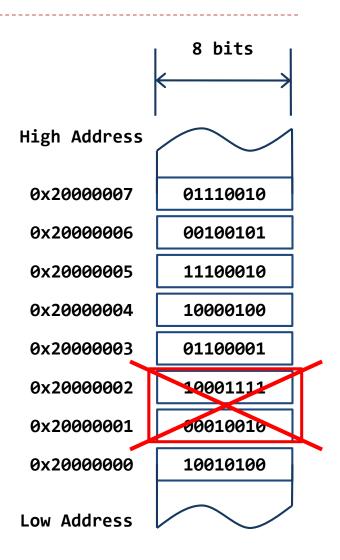
- When we refer to memory locations by address, we can only do so in units of bytes, halfwords or words
- Halfwords
 - I6 bits = 2 bytes = I halfword
 - We have four halfwords:
 - > 0x20000000
 - ▶ 0×20000002
 - ▶ 0x20000004
 - ▶ 0×20000006
 - Can you store a halfword anywhere? NO.
 - A halfword can only be stored at an address that's divisible by 2.
 - Memory address of a halfword is the lowest address of its two bytes.

 $Halfword-address\ mod\ 2=0$



- When we refer to memory locations by address, we can only do so in units of bytes, halfwords or words
- Halfwords
 - I6 bits = 2 bytes = I halfword
 - We have four halfwords:
 - ▶ 0×20000000
 - > 0×20000002
 - > 0x20000004
 - ▶ 0×20000006
 - 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

C				
		32-bit Words	Bytes	Addr.
				0015
	Word 3	Addr = ??		0014
				0013
VA / I				0012
What are the memory		A al al s		0011
address of these four words?	Word 2	Addr =	\perp	0010
		3.5	\perp	0009
			\perp	0008
		A al al a	\perp	0007
	Word 1	Addr =	\perp	0006
		3.5	\perp	0005
				0004
	Mord 0	A -1-1		0003
		Addr =		0002
		3.5		0001
				0000

Quiz (Answer)

		32-bit Words	Bytes	Addr.
				0015
	Word 3	Addr =		0014
	NOIG 3	0x0012		0013
\ \ \/\				0012
What are the memory		Addr		0011
address of these four words?	Word 2	Addr =		0010
		0x0008		0009
				8000
		Addr		0007
	Word 1	Addi =	\vdash	0006
	0x0004	0x0004	\vdash	0005
			\vdash	0004
		Addr	\vdash	0003
	Word 0	Addi =		0002
		0x0000		0001
				0000

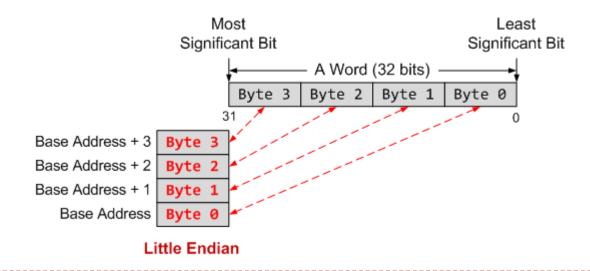
Endianess

- 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.



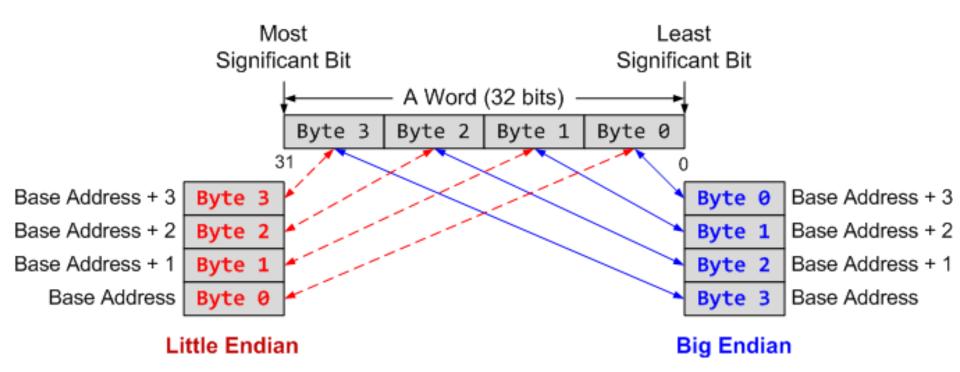
Endianess

- 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.



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

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

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

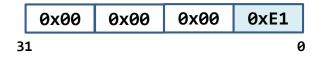
LDR	Load Word
LDRB	Load Byte
LDRH	Load Halfword
LDRSB	Load Signed Byte
LDRSH	Load Signed Halfword

STR	Store Word
STRB	Store Lower Byte
STRH	Store Lower Halfword

Load a Byte, Half-word, Word

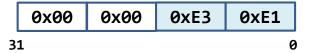
Load a Byte

LDRB r1, [r0]



Load a Halfword

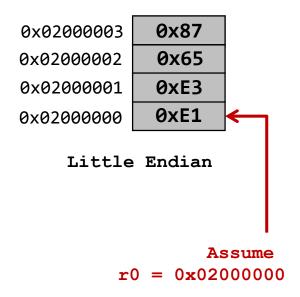
LDRH r1, [r0]



Load a Word

LDR r1, [r0]

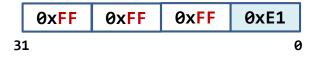




Sign Extension

Load a Signed Byte

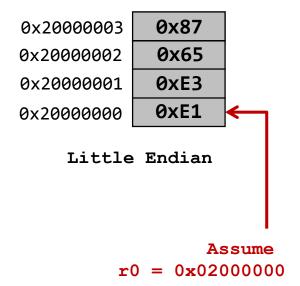
LDRSB r1, [r0]



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

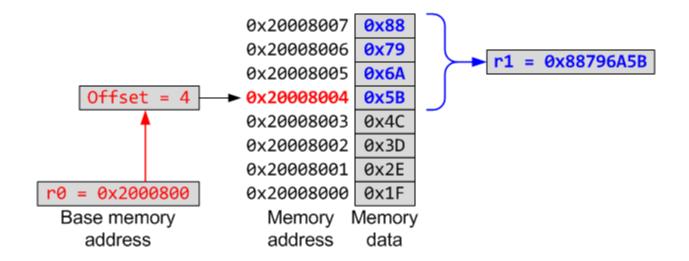
Pre-index

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

0x20008007 0x88 0x20008006 0x79 r1 0x20008005 0x6A Offset = 40x5B 0x20008004 0x4C 0x20008003 0x3D 0x20008002 0x20008001 0x2E r0 = 0x200080000x20008000 0x1F Base memory Memory Memory address address data

Pre-index

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



Post-index

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

```
0x20008007
                                 0x88
                                0x79
r0
                    0x20008006
                    0x20008005
                                0x6A
                    0x20008004 | 0x5B
     Offset = 4
                                0x4C
                    0x20008003 |
                                0x3D
                    0x20008002
                                             r1
                    0x20008001
                                0x2E
r0 = 0x20008000
                    0x20008000 | 0x1F
  Base memory
                       Memory Memory
     address
                       address
                                 data
```

Post-index

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



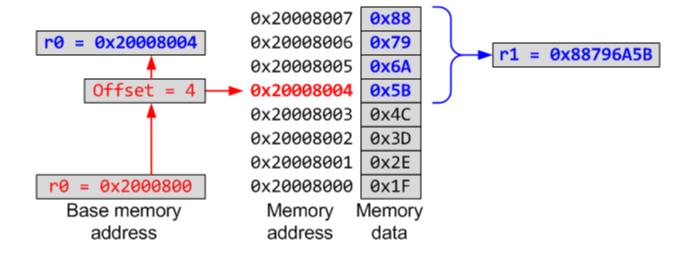
Pre-index with Updates

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

```
0x20008007
                                0x88
r0
                    0x20008006
                                0x79
                                             r1
                    0x20008005
                                0x6A
     Offset = 4
                                0x5B
                    0x20008004
                    0x20008003
                                0x4C
                                0x3D
                    0x20008002
                    0x20008001
                                0x2E
r0 = 0x20008000
                                0x1F
                    0x20008000
  Base memory
                      Memory
                               Memory
     address
                      address
                                 data
```

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]	$r1 \leftarrow memory[r0 + 4],$
		r0 is unchanged
Pre-index	LDR r1, [r0, #4]!	$r1 \leftarrow memory[r0 + 4]$
with update		$r0 \leftarrow r0 + 4$
Post-index	LDR r1, [r0], #4	r1 ← memory[r0]
		$r0 \leftarrow r0 + 4$

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

r1 before load

0x12345678

r1 after load

0x0000CDEF

Memory Address	Memory Data
0x20008003	0x89
0x20008002	0xAB
0x20008001	0xCD
0x20008000	0×EF

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

r1 before load

0x12345678

r1 after load

0xFFFFFFF

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

r0 before store

0x20008000

r0 after store



Memory Data
0x00

r0 before store

0x20008000

r0 after store

0x20008004

Memory Address	Memory Data
0×20008007	0×00
0x20008006	0×00
0x20008005	0×00
0x20008004	0x00
0x20008003	0x76
0x20008002	0x54
0x20008001	0x32
0x20008000	0x10

r0 before the store

0x20008000

r0 after the store

Memory Address	Memory Data
0×20008007	0x00
0x20008006	0×00
0x20008005	0×00
0x20008004	0x00
0x20008003	0x00
0x20008002	0×00
0x20008001	0×00
0x20008000	0×00

r0 before store

0x20008000

r0 after store

0x20008000

Memory Data
0x76
0x54
0x32
0x10
0x00
0x00
0x00
0x00

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
0×20008001	0x00
0×20008000	0x00

r0 before store

0x20008000

r0 after store

0x20008004

Memory Data
0x76
0x54
0x32
0x10
0x00
0x00
0x00
0x00

If big endianess LDR r11, [r0] is used

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

r11 before load

0x12345678

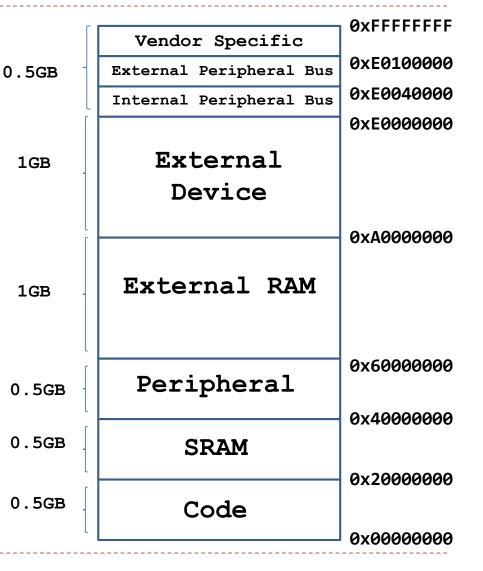
r11 after load

0xA7908CEE

Memory Address	Memory Data
0x20008003	0×EE
0x20008002	0x8C
0x20008001	0x90
0x20008000	0xA7

Cortex-M3 Memory Map

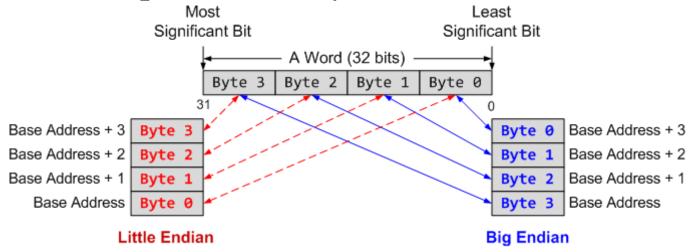
- 32-bit Memory Address
- 2³² bytes of memory space (4 GB)
- Harvard architecture: physically separated instruction memory and data memory



		\neg		1	
0xE00FF000	ROM table	_/\		0xFFFFFFF	
0xE00FEFFF	External private peripheral bus		Vendor specific		
0xE0042000			r straigt spreams		
0xE0041000	ETM			0xE0100000	
0xE0040000	TPIU	\perp	Private peripheral bus:	0xE00FFFFF	
_			Debug/external	0xE0040000	
0xE003FFFF	Reserved		Private peripheral bus:	0xE003FFFF	
0xE000F000	rieserved		Internal	0xE0000000	
0xE000E000	NVIC	/		0xDFFFFFF	
0xE000DFFF	Reserved				
0xE0003000	neserveu		External device		
0xE0002000	FPB	\Box /	External device		Cortex-M3
0xE0001000	DWT	\Box / \Box			Fixed Memory
0xE0000000	ITM	7/	1 GB	0xA0000000	Map
0x43FFFFFF		\neg		0x9FFFFFF	
OX+OITTIT	Bit-band alias		External RAM		
0x42000000	32 MB		1 GB	0x60000000	
0x41FFFFFF		$\neg \setminus \mid$		0x5FFFFFF	
0x40100000	31 MB		Davida anala		
	Dit be and as also	\neg	Peripherals		
0x40000000	Bit-band region 1 MB		0.5 GB	0x40000000	
•				0x3FFFFFF	
			ODAM		
0x23FFFFFF			SRAM		
	Dit beautel's		0.5 GB	0x20000000	
	Bit-band alias			0x1FFFFFFF	
0x22000000	32 MB				
0x21FFFFFF		7 / 1	Code		
0x20100000	31 MB		0.5 GB	0x00000000	
0x20000000	Bit-band region			1	

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]	rI ← memory[r0 + 4], r0 is unchanged
Pre-index with update	LDR r1, [r0, #4]!	$rI \leftarrow memory[r0 + 4]$ $r0 \leftarrow r0 + 4$
Post-Index	LDR r1, [r0], #4	$rI \leftarrow memory[r0]$ $r0 \leftarrow r0 + 4$