



香港中文大學

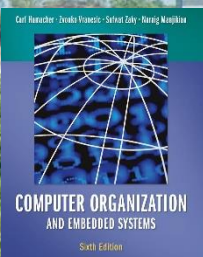
The Chinese University of Hong Kong

CSCI2510 Computer Organization

Lecture 03: Memory Basics

Ming-Chang YANG

mcyang@cse.cuhk.edu.hk



Reading: Chap. 2.1~2.2

Recall: Example of Program Execution



- Considering a program of 3 instructions:

PC → **I₀: Load R0, LOC**

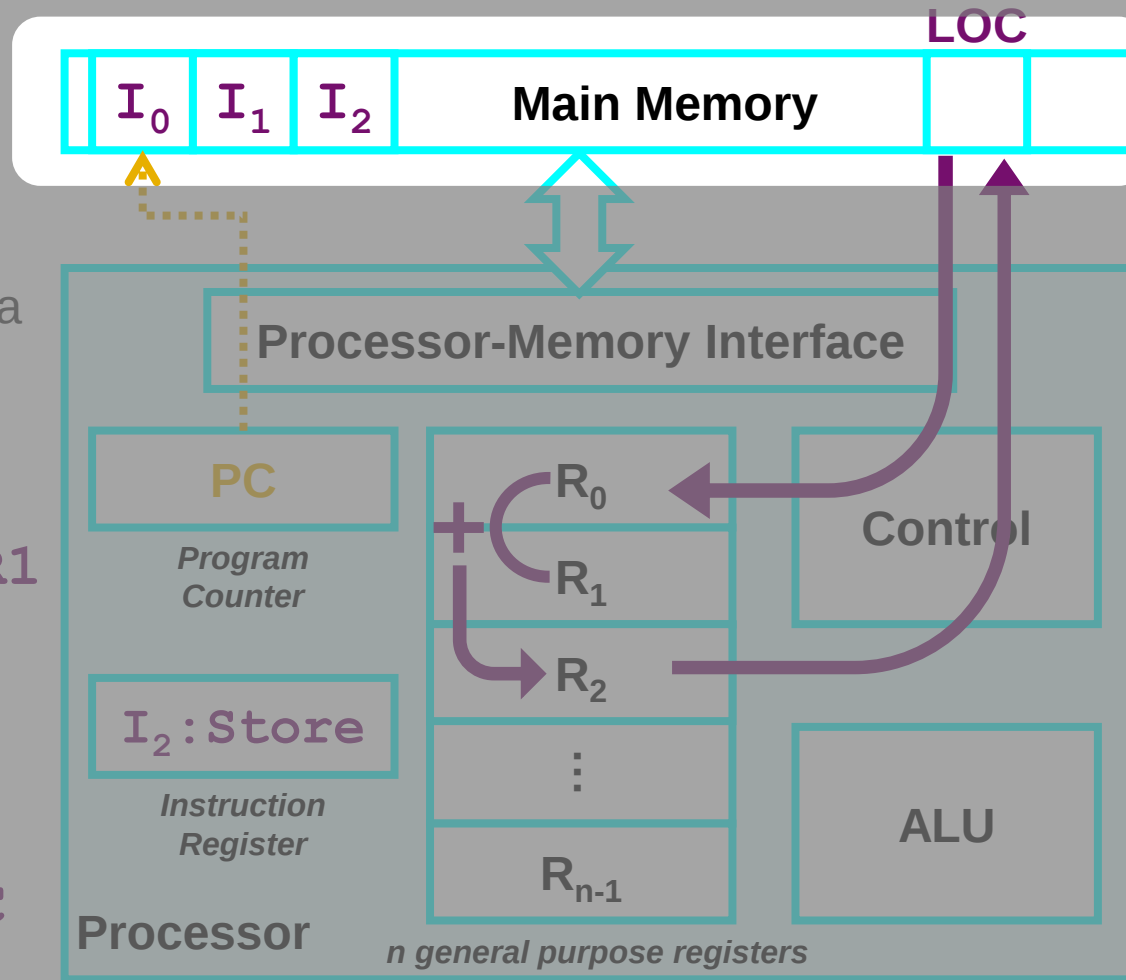
- Reads the contents of a memory location LOC
- Loads them into processor register R0

– **I₁: Add R2, R0, R1**

- Adds the contents of registers R0 and R1
- Places their sum into register R2

– **I₂: Store R2, LOC**

- Copies the operand in register R2 to memory location LOC



PC: contains the memory address of the next instruction to be fetched and executed.

IR: holds the instruction that is currently being executed.

R₀~R_{n-1}: n general-purpose registers.

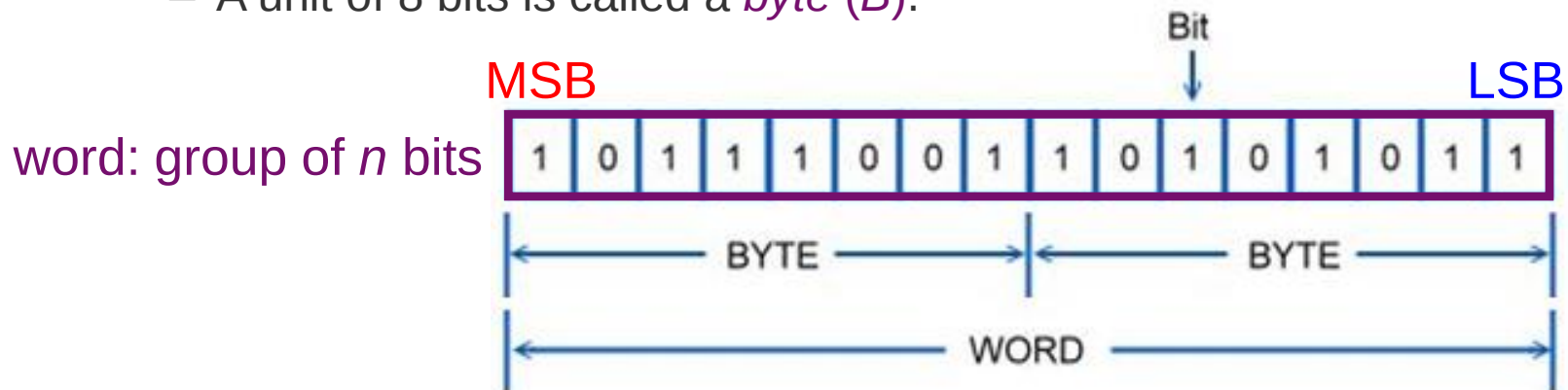
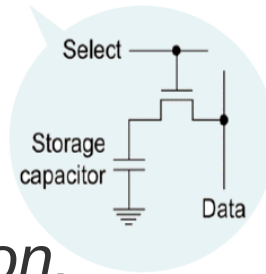


- Memory Locations and Addresses
 - Memory Organization and Address
 - Byte Addressability
 - Big-Endian and Little-Endian Ordering
 - Accessing Numbers, Characters, and Strings
- Memory Operations
 - Load and Store
 - Operational Unit
 - Word Alignment

Memory Organization (1/2)

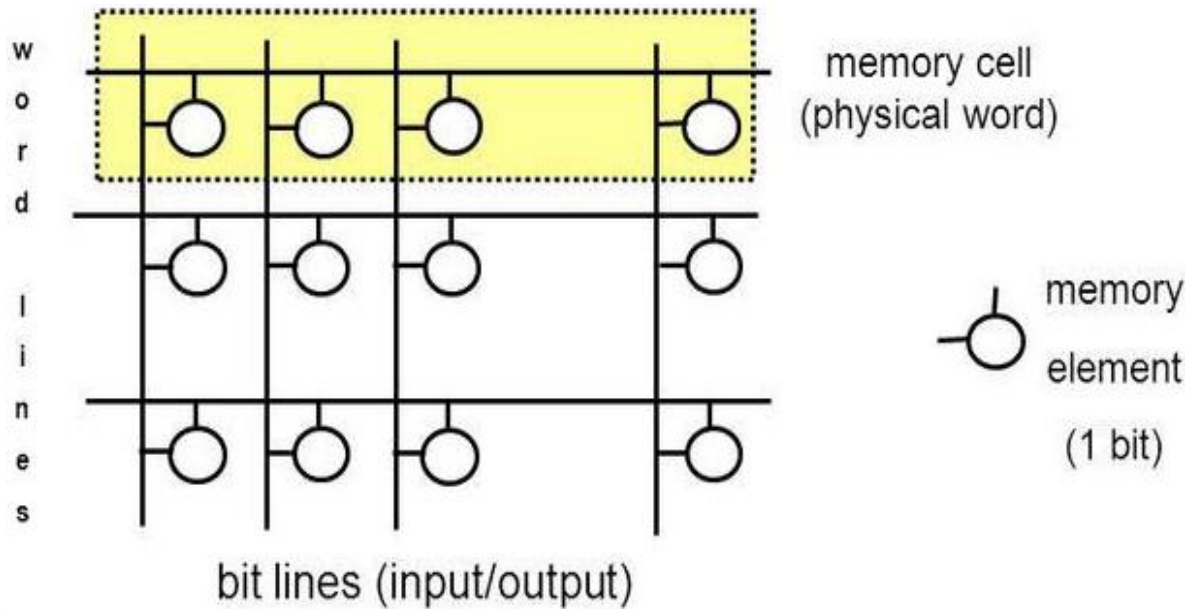


- Memory consists of many millions of memory *cells*.
 - Each cell can store a bit of information (0 or 1).
- Cells (bits) are organized in *groups of n bits*.
 - Reason: *A single bit represents very little information.*
 - A group of n bits: a *word* (where n is the *word length*).
 - A word can be **stored** or **retrieved** in a single, basic operation.
 - Common word lengths in modern computers: 16 to 64 bits.
 - The number of bytes in a word is usually a power of 2.
 - A unit of 8 bits is called a *byte (B)*.

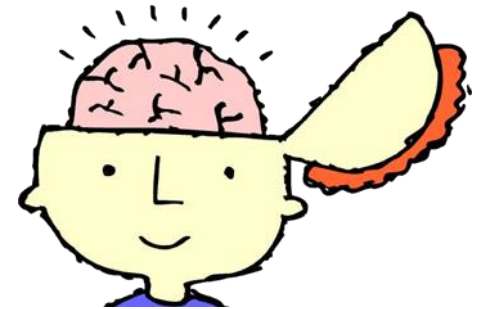


Example: A word of 16 bits

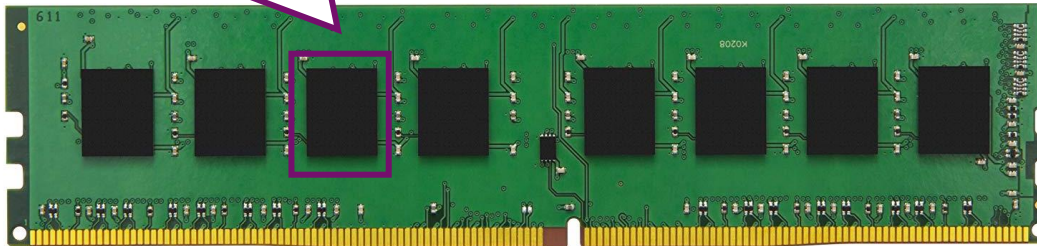
Memory Organization (2/2)



Question: How to access the contents of memory?



DDR4
RAM



Memory Address (1/2)



- Accessing the contents of memory requires **distinct addresses** for **each memory location**.
 - Format:** k -bit addresses can represent 2^k distinct locations.

$k = 1 \rightarrow 2^1 = 2$

0	1 st location
1	2 nd location

$k = 2 \rightarrow 2^2 = 4$

00	1 st location
01	2 nd location
10	3 rd location
11	4 th location

$k = 3 \rightarrow 2^3 = 8$

000	1 st location
001	2 nd location
010	3 rd location
011	4 th location
100	5 th location
101	6 th location
110	7 th location
111	8 th location

$k = 4 \rightarrow 2^4 = 16$

0000	1 st location
0001	2 nd location
0010	3 rd location
0011	4 th location
⋮	⋮
1111	16 th location

Example:

- The address “100” ($k=3$) denotes the memory location at -----

*Note: Addresses do **NOT** have to be really stored in memory!*

Memory Address (2/2)



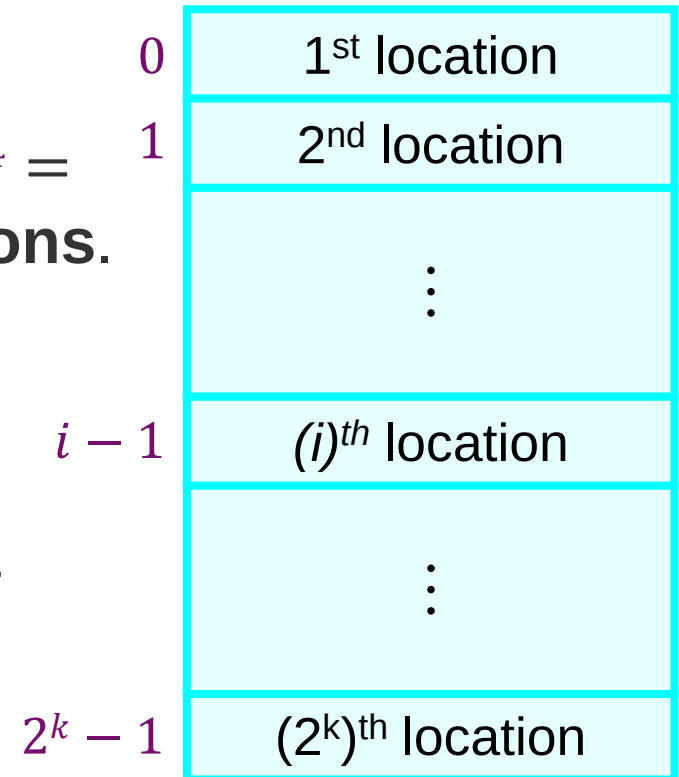
- General Rule: It is customary to use numbers from $0 \sim 2^k - 1$ as the successive addresses in the memory.
→ k -bit addresses have 2^k addressable locations.

- Example:

- A 24-bit address can represent $2^{24} = 16,777,216 = 16M$ **distinct locations**.

- Notational conventions:

- 1K is the number $2^{10} = 1,024$
 - 1M is the number $2^{20} = 1,048,576$
 - 1G is the number $2^{30} = 1,073,741,824$
 - 1T is the number $2^{40} = \dots$





- Memory Locations and Addresses
 - Memory Organization and Address
 - Byte Addressability
 - Big-Endian and Little-Endian Ordering
 - Accessing Numbers, Characters, and Strings
- Memory Operations
 - Load and Store
 - Operational Unit
 - Word Alignment

Unit of Location: Byte Addressability (1/2)

- Basic information quantities: **bit**, **byte**, and **word**.
 - A byte (B) is always 8 bits.
 - The word length typically ranges from 16 to 64 bits.
- What should be the **unit size of an address**?
 - It is **costly** to assign distinct addresses to individual **bit**.
 - The **word** lengths may be **different** in different computers.

Unit size: bit

0	1	2	...	15
16	17	18	...	31
32	33	34	...	47
⋮				

Unit size: 16-bit word

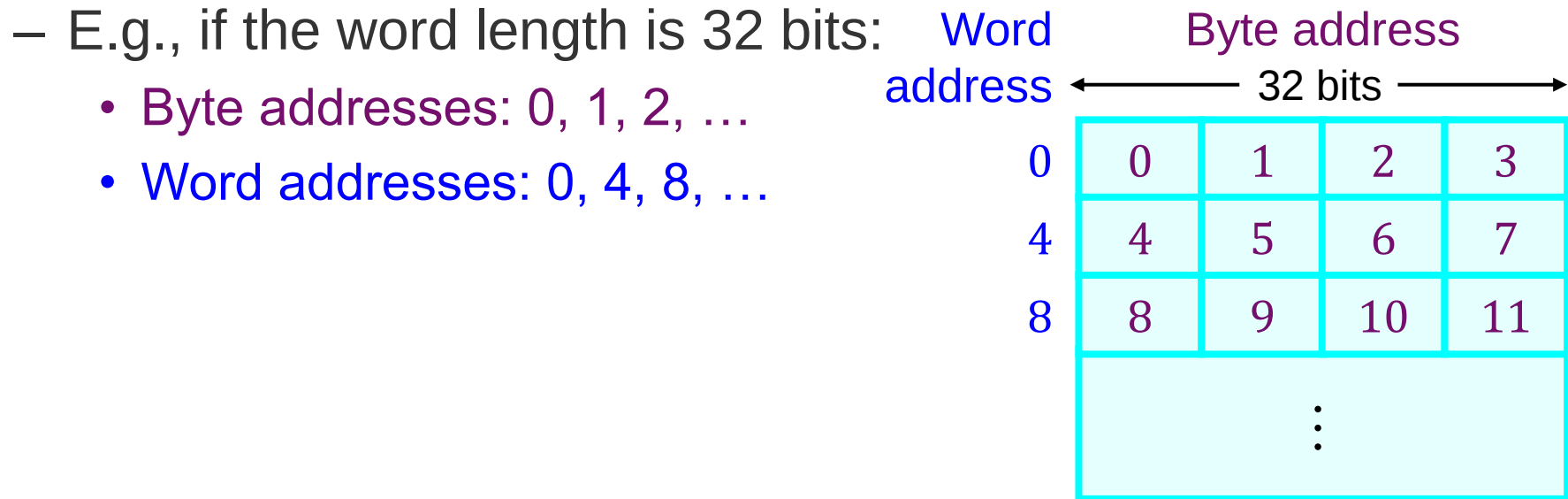
0	1 st word (bit: 0~15)
1	2 nd word (bit: 16~31)
2	3 rd word (bit: 32~47)
⋮	

Unit size: 32-bit word

0	1 st word (bit: 0~31)
1	2 nd word (bit: 32~63)
2	3 rd word (bit: 64~95)
⋮	

Unit of Location: Byte Addressability (2/2)

- The most practical assignment: **byte** addresses
 - **Successive addresses** represents **successive byte locations** in the memory.



- **Byte addressability:** Each **byte** location in the memory has its own address and is addressable.
- We need k -bit addresses to locate 2^k bytes.

Class Exercise 3.1

Student ID: _____ Date: _____

Name: _____

- **Online Price** HKD \$9,988
- **CPU** 1.2GHz dual-core 7th-generation Intel Core m3
- **Memory** 8 GB 1866MHz LPDDR3
- **Storage** 256 GB SSD



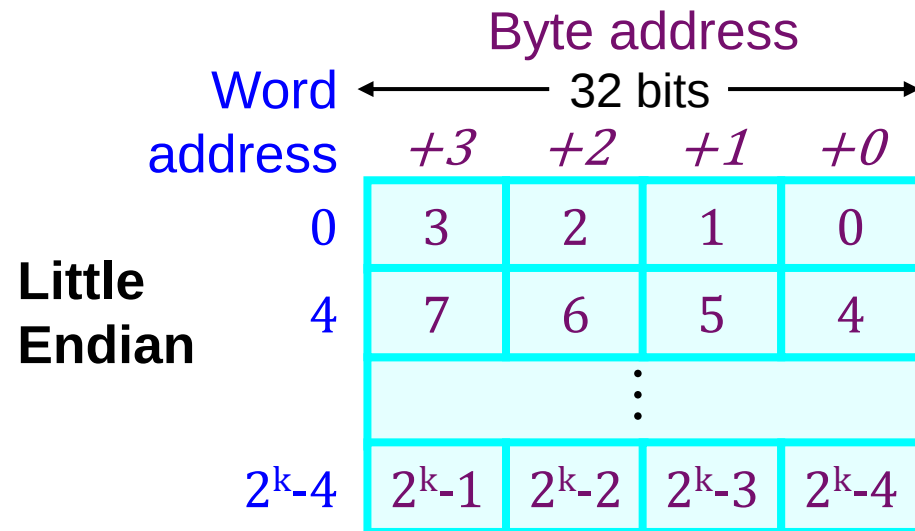
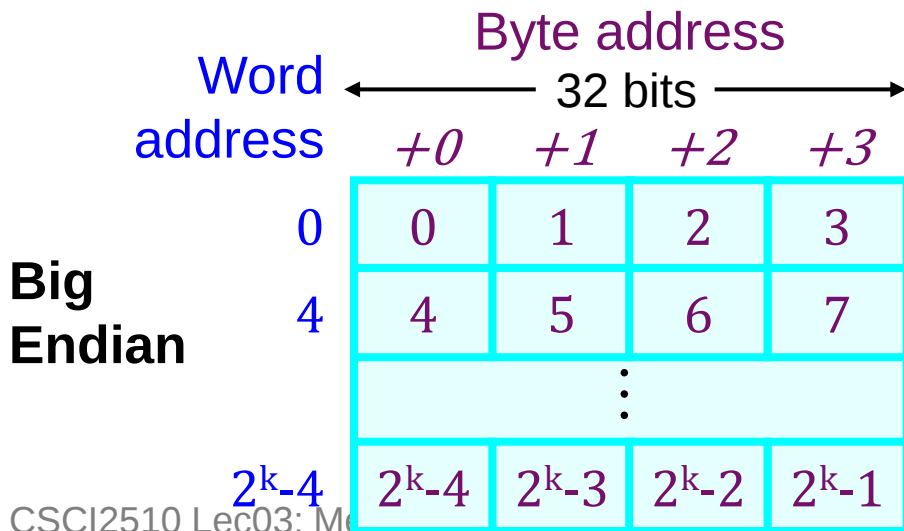
- Given the information about the **MacBook**:
 - 1) How many bits are there in the memory system?
– Answer: _____
 - 2) How many unique 64-bit word locations does it have?
– Answer: _____
 - 3) How many bits are required by the address if it is byte addressable memory?
– Answer: _____



- Memory Locations and Addresses
 - Memory Organization and Address
 - Byte Addressability
 - Big-Endian and Little-Endian Ordering
 - Accessing Numbers, Characters, and Strings
- Memory Operations
 - Load and Store
 - Operational Unit
 - Word Alignment

Big-Endian and Little-Endian Ordering

- **Big-Endian Ordering** (e.g., Motorola):
 - ① **Byte addresses** within a word are ordered **left-to-right** ($\rightarrow$);
 - ② For multi-byte data (e.g., a number): Lower byte addresses are used for **MORE** significant bytes.
- **Little-Endian Ordering** (e.g., Intel):
 - ① **Byte addresses** within a word are ordered **right-to-left** ($\leftarrow$);
 - ② For multi-byte data (e.g., a number): Lower byte addresses are used for **LESS** significant bytes.

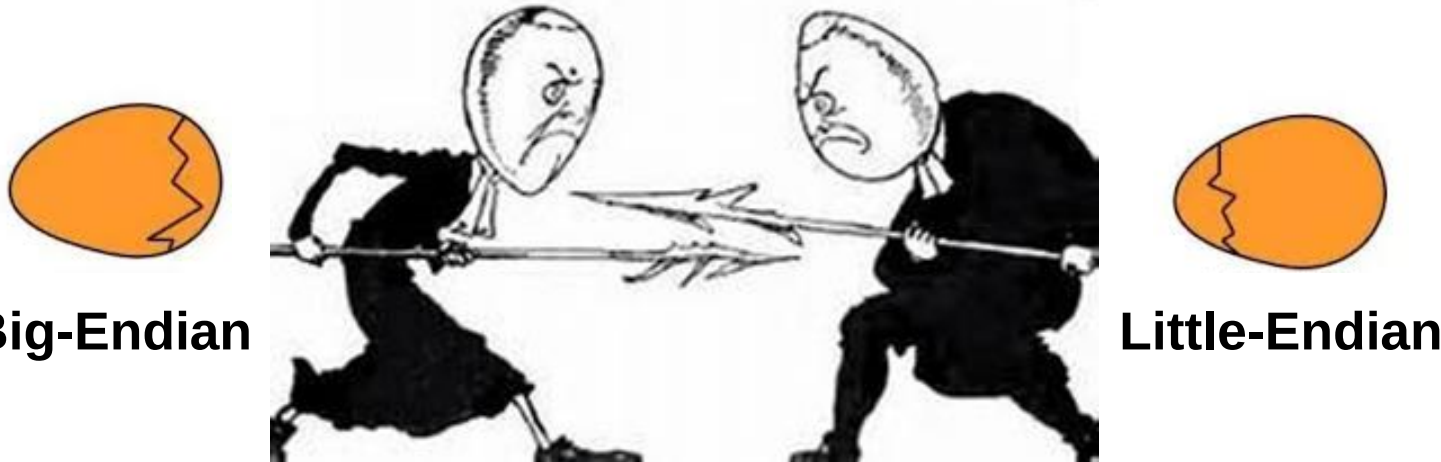
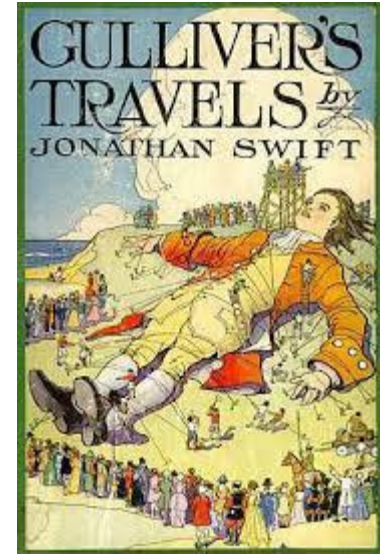


Fun Knowledge about “Endian”



- The word “endian” were drawn from Jonathan Swift's 1726 satire, “Gulliver's Travels”.

– *In which, civil war erupts over whether the big end or the little end of a boiled egg is the proper end to crack open ...*



- It is analogous to counting from the end that contains the most significant bit or the least significant bit.

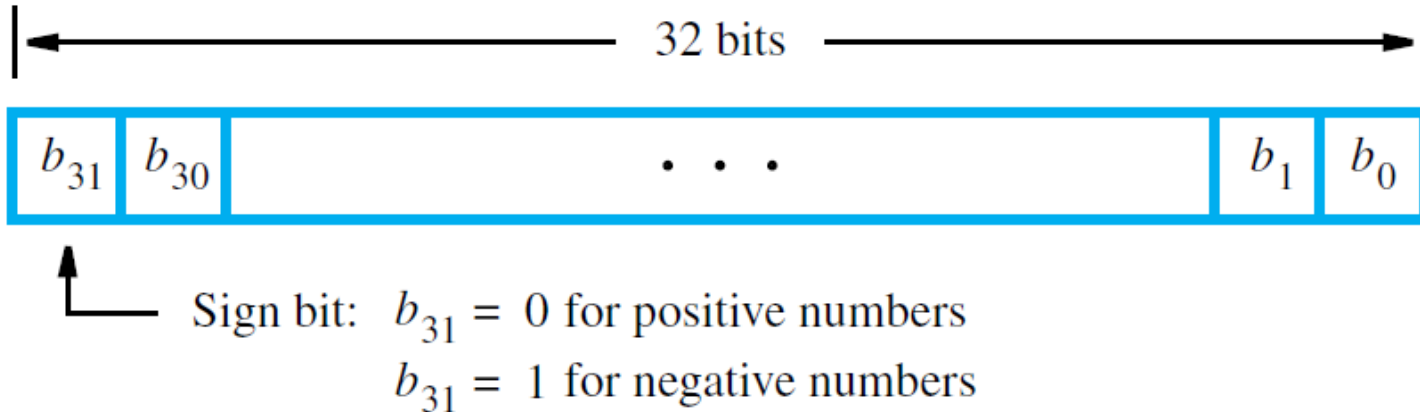


- Memory Locations and Addresses
 - Memory Organization and Address
 - Byte Addressability
 - Big-Endian and Little-Endian Ordering
 - Accessing Numbers, Characters, and Strings
- Memory Operations
 - Load and Store
 - Operational Unit
 - Word Alignment

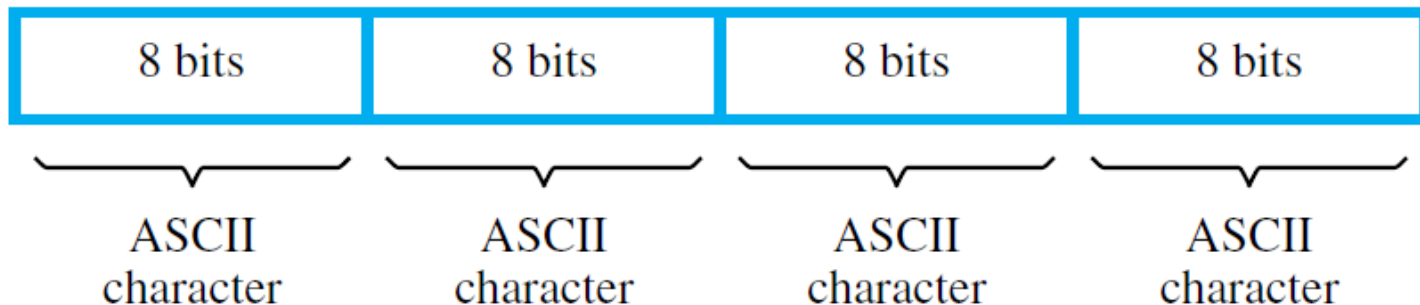
Accessing Numbers and Characters



- A number usually occupies one **word**, and can be accessed in the memory by its word address.



- Differently, each character can be represented by one **byte**, and can be accessed by their byte address.



Class Exercise 3.2



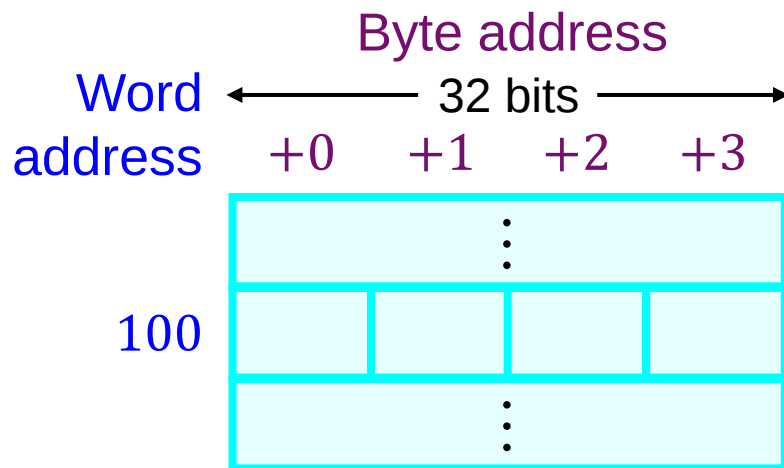
- Consider a computer with
 - Byte-addressable memory
 - 32-bit words
 - Big-endian or Little-endian
- A program reads ASCII characters, and stores them in successive byte locations, starting at **1000**.
- After entering “Exercise”, show the contents of memory words at locations
 - 1000: _____
 - 1004: _____

Dec	Bin	Hex	Char	Dec	Bin	Hex	Char
64	0100 0000	40	@	96	0110 0000	60	`
65	0100 0001	41	A	97	0110 0001	61	a
66	0100 0010	42	B	98	0110 0010	62	b
67	0100 0011	43	C	99	0110 0011	63	c
68	0100 0100	44	D	100	0110 0100	64	d
69	0100 0101	45	E	101	0110 0101	65	e
70	0100 0110	46	F	102	0110 0110	66	f
71	0100 0111	47	G	103	0110 0111	67	g
72	0100 1000	48	H	104	0110 1000	68	h
73	0100 1001	49	I	105	0110 1001	69	i
74	0100 1010	4A	J	106	0110 1010	6A	j
75	0100 1011	4B	K	107	0110 1011	6B	k
76	0100 1100	4C	L	108	0110 1100	6C	l
77	0100 1101	4D	M	109	0110 1101	6D	m
78	0100 1110	4E	N	110	0110 1110	6E	n
79	0100 1111	4F	O	111	0110 1111	6F	o
80	0101 0000	50	P	112	0111 0000	70	p
81	0101 0001	51	Q	113	0111 0001	71	q
82	0101 0010	52	R	114	0111 0010	72	r
83	0101 0011	53	S	115	0111 0011	73	s
84	0101 0100	54	T	116	0111 0100	74	t
85	0101 0101	55	U	117	0111 0101	75	u
86	0101 0110	56	V	118	0111 0110	76	v
87	0101 0111	57	W	119	0111 0111	77	w
88	0101 1000	58	X	120	0111 1000	78	x
89	0101 1001	59	Y	121	0111 1001	79	y
90	0101 1010	5A	Z	122	0111 1010	7A	z
91	0101 1011	5B	[	123	0111 1011	7B	{
92	0101 1100	5C	\	124	0111 1100	7C	
93	0101 1101	5D	]	125	0111 1101	7D	}
94	0101 1110	5E	^	126	0111 1110	7E	~
95	0101 1111	5F	_	127	0111 1111	7F	[DEL]

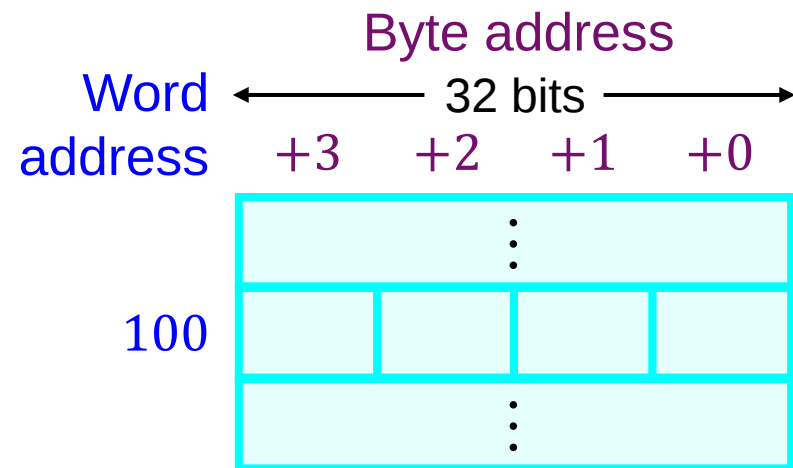
Class Exercise 3.3



- Consider a computer with
 - **Byte-addressable** memory
 - **32-bit words**
 - **Big-endian** or **Little-endian**
- Show the memory content at word address 100 if that word holds a **four-byte number** $(0A0B0C0D)_{16}$.



Big-Endian

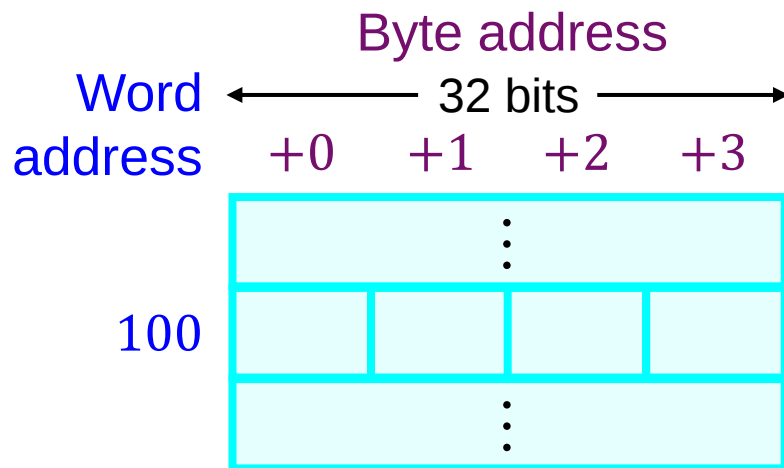


Little-Endian

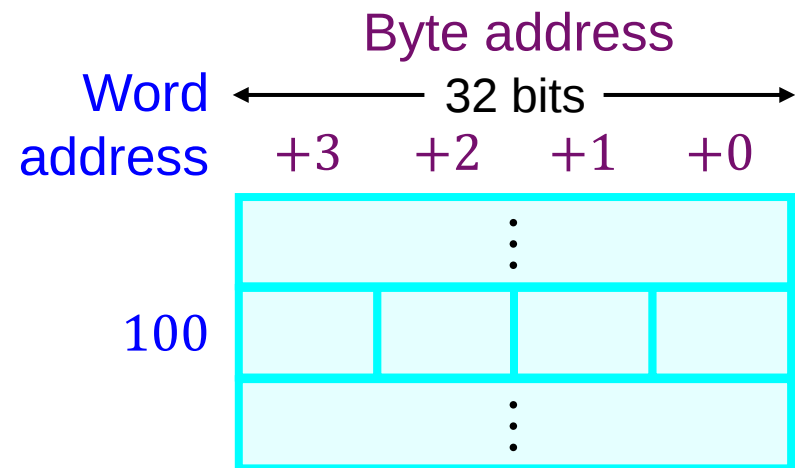
Class Exercise 3.4



- Consider a computer with
 - **Byte-addressable** memory
 - **32-bit words**
 - **Big-endian** or **Little-endian**
- Show the memory content at word address 100 if that word address holds a **two-byte number** $(0A0B)_{16}$.



Big-Endian



Little-Endian

Accessing Strings



- How can we represent strings which could be of variable length? (E.g., “University”)
 - **Method 1:** Use a **null** character to mark the end
 - ‘U’, ‘n’, ‘i’, ‘v’, ‘e’, ‘r’, ‘s’, ‘i’, ‘t’, ‘y’, ‘\0’
 - C Language adopts this method.
 - **Method 2:** Use a **number** to represent the length
 - **10**, ‘U’, ‘n’, ‘i’, ‘v’, ‘e’, ‘r’, ‘s’, ‘i’, ‘t’, ‘y’
 - Pascal Language adopts this method.
- What are the pros and cons of them?
 - Consider length limit of the string, processing speed, convenience in handling, etc.

Recall: ASCII Table



Dec	Bin	Hex	Char	Dec	Bin	Hex	Char	Dec	Bin	Hex	Char	Dec	Bin	Hex	Char
0	0000 0000	00	[NUL]	32	0010 0000	20	space	64	0100 0000	40	@	96	0110 0000	60	`
1	0000 0001	01	[SOH]	33	0010 0001	21	!	65	0100 0001	41	A	97	0110 0001	61	a
2	0000 0010	02	[STX]	34	0010 0010	22	"	66	0100 0010	42	B	98	0110 0010	62	b
3	0000 0011	03	[ETX]	35	0010 0011	23	#	67	0100 0011	43	C	99	0110 0011	63	c
4	0000 0100	04	[EOT]	36	0010 0100	24	\$	68	0100 0100	44	D	100	0110 0100	64	d
5	0000 0101	05	[ENQ]	37	0010 0101	25	%	69	0100 0101	45	E	101	0110 0101	65	e
6	0000 0110	06	[ACK]	38	0010 0110	26	&	70	0100 0110	46	F	102	0110 0110	66	f
7	0000 0111	07	[BEL]	39	0010 0111	27	'	71	0100 0111	47	G	103	0110 0111	67	g
8	0000 1000	08	[BS]	40	0010 1000	28	(	72	0100 1000	48	H	104	0110 1000	68	h
9	0000 1001	09	[TAB]	41	0010 1001	29	)	73	0100 1001	49	I	105	0110 1001	69	i
10	0000 1010	0A	[LF]	42	0010 1010	2A	*	74	0100 1010	4A	J	106	0110 1010	6A	j
11	0000 1011	0B	[VT]	43	0010 1011	2B	+	75	0100 1011	4B	K	107	0110 1011	6B	k
12	0000 1100	0C	[FF]	44	0010 1100	2C	,	76	0100 1100	4C	L	108	0110 1100	6C	l
13	0000 1101	0D	[CR]	45	0010 1101	2D	-	77	0100 1101	4D	M	109	0110 1101	6D	m
14	0000 1110	0E	[SO]	46	0010 1110	2E	.	78	0100 1110	4E	N	110	0110 1110	6E	n
15	0000 1111	0F	[SI]	47	0010 1111	2F	/	79	0100 1111	4F	O	111	0110 1111	6F	o
16	0001 0000	10	[DLE]	48	0011 0000	30	0	80	0101 0000	50	P	112	0111 0000	70	p
17	0001 0001	11	[DC1]	49	0011 0001	31	1	81	0101 0001	51	Q	113	0111 0001	71	q
18	0001 0010	12	[DC2]	50	0011 0010	32	2	82	0101 0010	52	R	114	0111 0010	72	r
19	0001 0011	13	[DC3]	51	0011 0011	33	3	83	0101 0011	53	S	115	0111 0011	73	s
20	0001 0100	14	[DC4]	52	0011 0100	34	4	84	0101 0100	54	T	116	0111 0100	74	t
21	0001 0101	15	[NAK]	53	0011 0101	35	5	85	0101 0101	55	U	117	0111 0101	75	u
22	0001 0110	16	[SYN]	54	0011 0110	36	6	86	0101 0110	56	V	118	0111 0110	76	v
23	0001 0111	17	[ETB]	55	0011 0111	37	7	87	0101 0111	57	W	119	0111 0111	77	w
24	0001 1000	18	[CAN]	56	0011 1000	38	8	88	0101 1000	58	X	120	0111 1000	78	x
25	0001 1001	19	[EM]	57	0011 1001	39	9	89	0101 1001	59	Y	121	0111 1001	79	y
26	0001 1010	1A	[SUB]	58	0011 1010	3A	:	90	0101 1010	5A	Z	122	0111 1010	7A	z
27	0001 1011	1B	[ESC]	59	0011 1011	3B	;	91	0101 1011	5B	[	123	0111 1011	7B	{
28	0001 1100	1C	[FS]	60	0011 1100	3C	<	92	0101 1100	5C	\	124	0111 1100	7C	
29	0001 1101	1D	[GS]	61	0011 1101	3D	=	93	0101 1101	5D	]	125	0111 1101	7D	}
30	0001 1110	1E	[RS]	62	0011 1110	3E	>	94	0101 1110	5E	^	126	0111 1110	7E	~
31	0001 1111	1F	[US]	63	0011 1111	3F	?	95	0101 1111	5F	_	127	0111 1111	7F	[DEL]



- Memory Locations and Addresses
 - Memory Organization and Address
 - Byte Addressability
 - Big-Endian and Little-Endian Ordering
 - Accessing Numbers, Characters, and Strings
- Memory Operations
 - Load and Store
 - Operational Unit
 - Word Alignment

Memory Operations



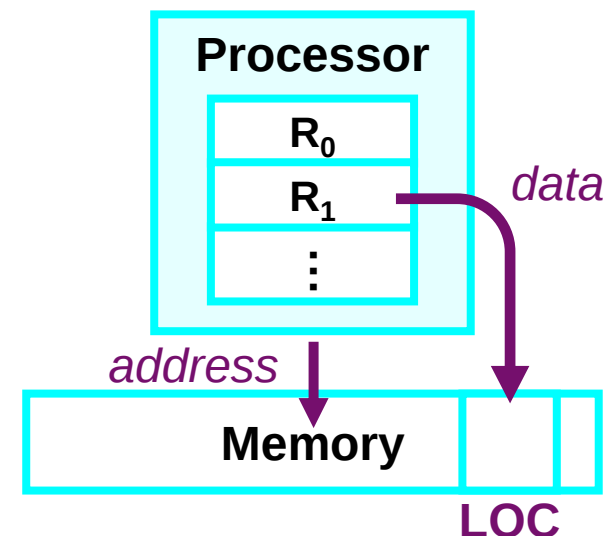
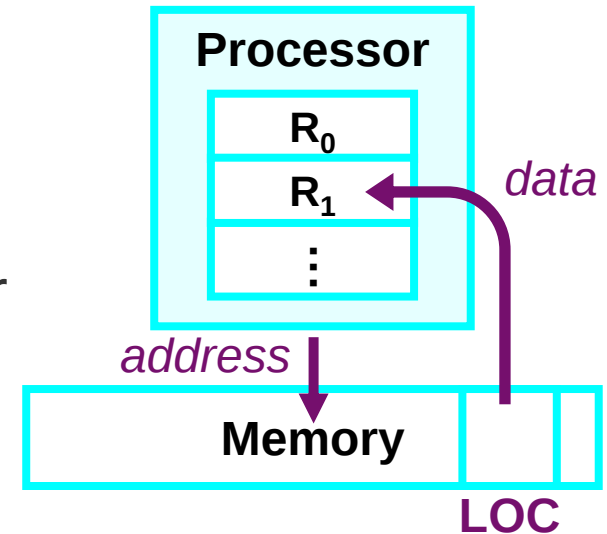
- Two operations for manipulating the memory:

- **Load** (read or fetch):

- Processor sends **address** to memory,
- Memory **returns data** to processor
e.g., $R1 \leftarrow [LOC]$
(R1 is an internal register in the processor)

- **Store** (write):

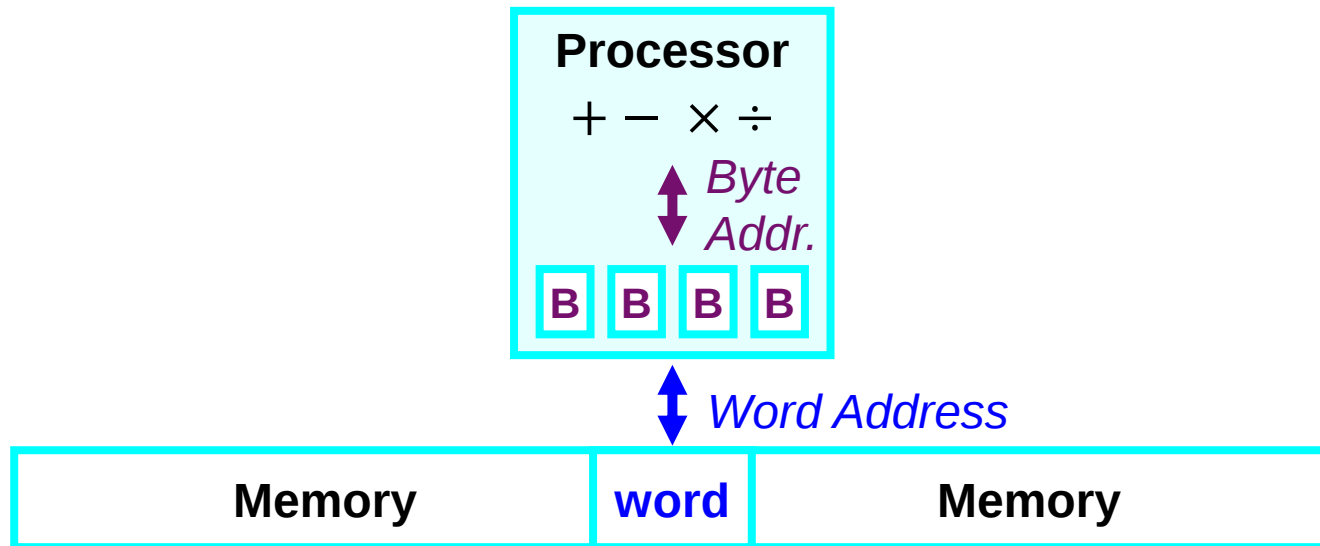
- Processor sends **address and data** to memory,
- Memory **overwrites** location with **new data**
e.g., $LOC \leftarrow [R1]$



Operational Unit



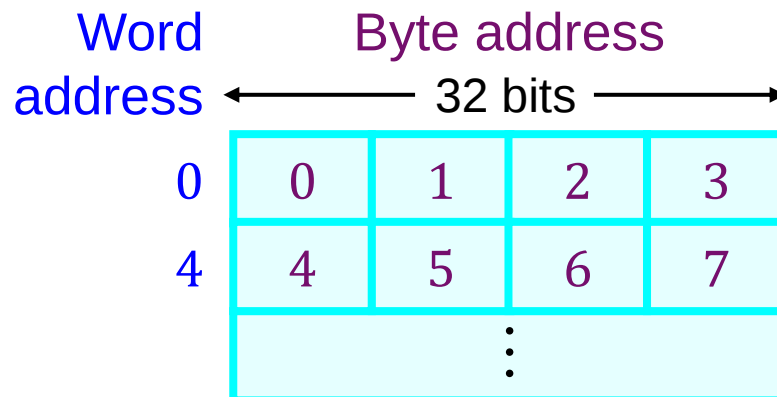
- Most machines are **byte-addressable**.
 - Each memory address refers to a **byte (B)**.
- Memory is designed to store/retrieve in **words**.
 - A **word** is usually of 16, 32 or 64 bits.
 - Reason? **Performance consideration** (see Lec08).



Word Alignment



- 32-bit words align naturally at addresses 0, 4, 8, ...
 - **Aligned addresses:** Word begins at a byte address that is a multiple of the number of bytes in a word.
 - The aligned addresses for 16-bit and 64-bit words:
 - 16-bit word: 0, 2, 4, 6, 8, 10, ...
 - 64-bit word: 0, 8, 16, ...
- Unaligned accesses are either **not allowed** or **slower**.
 - E.g., read a 32-bit word from the byte address `0x01`
 - Note: `0x` represents the *hexadecimal* number system.



- Memory Locations and Addresses
 - Memory Organization and Address
 - Byte Addressability
 - Big-Endian and Little-Endian Assignments
 - Accessing Numbers, Characters, and Strings
- Memory Operations
 - Load and Store
 - Operational Unit
 - Word Alignment