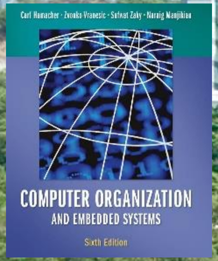
香港中文大學
The Chinese University of Hong Kong

CSCI2510 Computer Organization

Lecture 03: Memory Basics

Ming-Chang YANG

mcyang@cse.cuhk.edu.hk



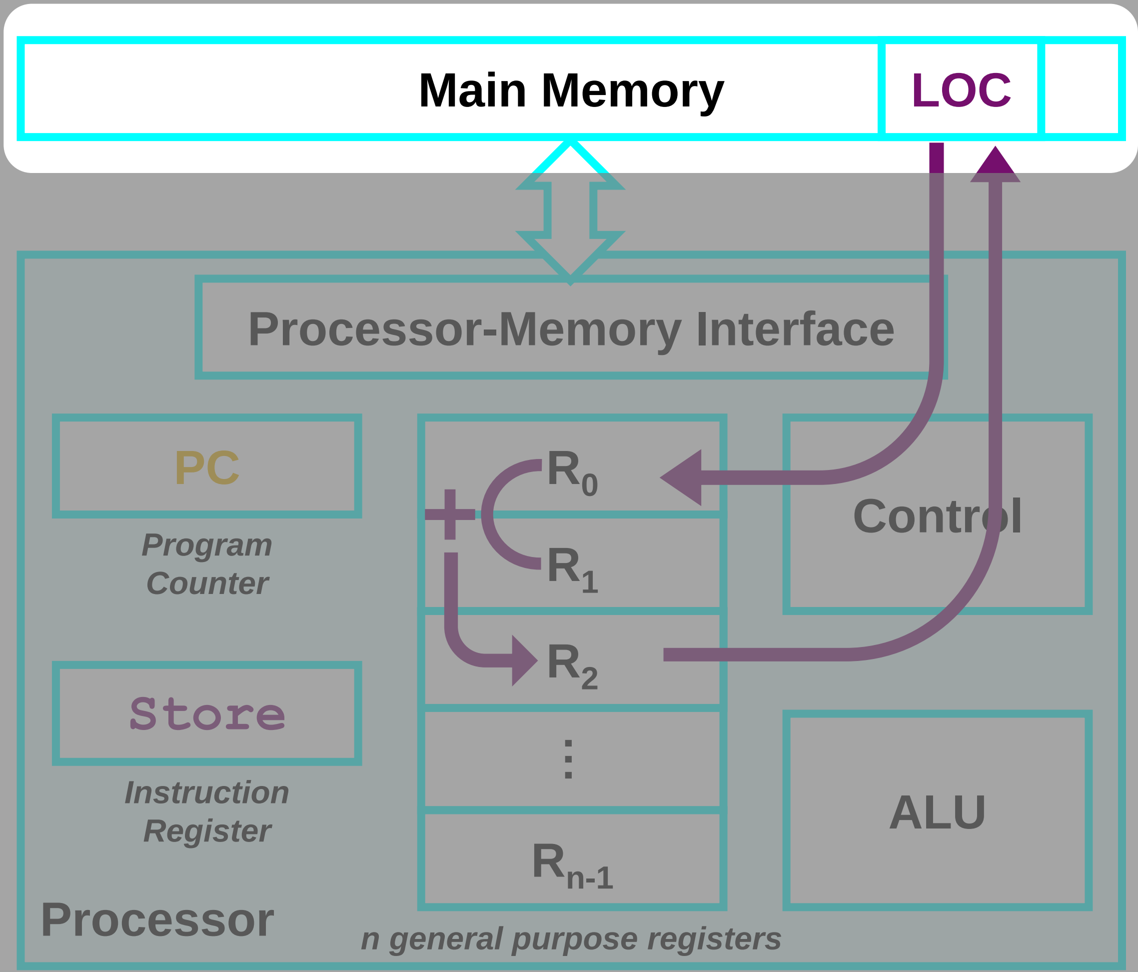
Reading: Chap. 2.1~2.2

Recall: Program Execution

- Considering a program of 3 instructions:
  - `Load R0, LOC`
    - Reads the contents of a memory location LOC
    - Loads them into processor register R0
  - `Add R2, R0, R1`
    - Adds the contents of registers R0 and R1
    - Places their sum into register R2

PC → `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.
R_0~R_{n-1}: n general-purpose registers.

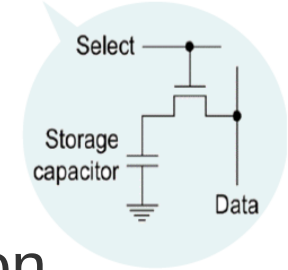
Outline

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

- Memory Operations
  - `Load`
  - `Store`

Memory Organization (1/2)

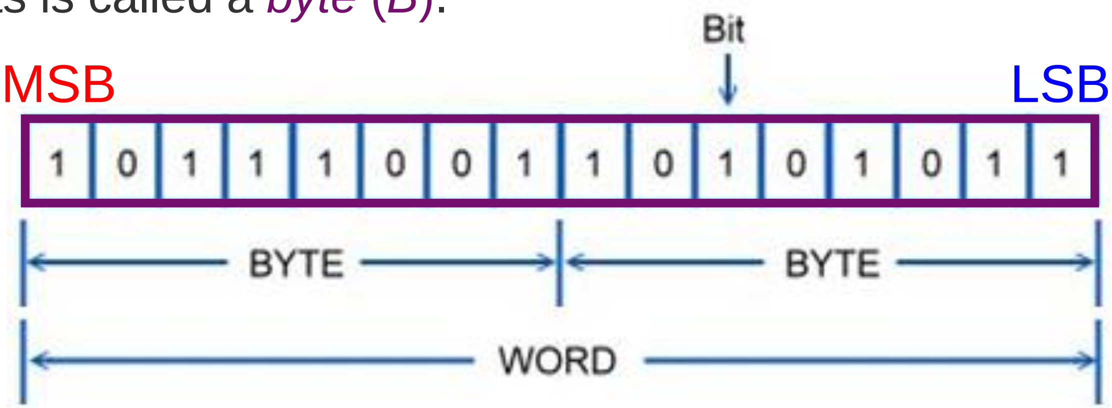
- Memory consists of many millions of storage *cells*.
  - Each cell can store a bit of information (0 or 1).



- 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 <u>single, basic operation</u>.
    - Common word lengths in modern computers: <u>*16 to 64 bits*</u>.
      - The number of bytes in a word is usually a <u>power of 2</u>.
      - A unit of 8 bits is called a *byte* (*B*).

MSB LSB

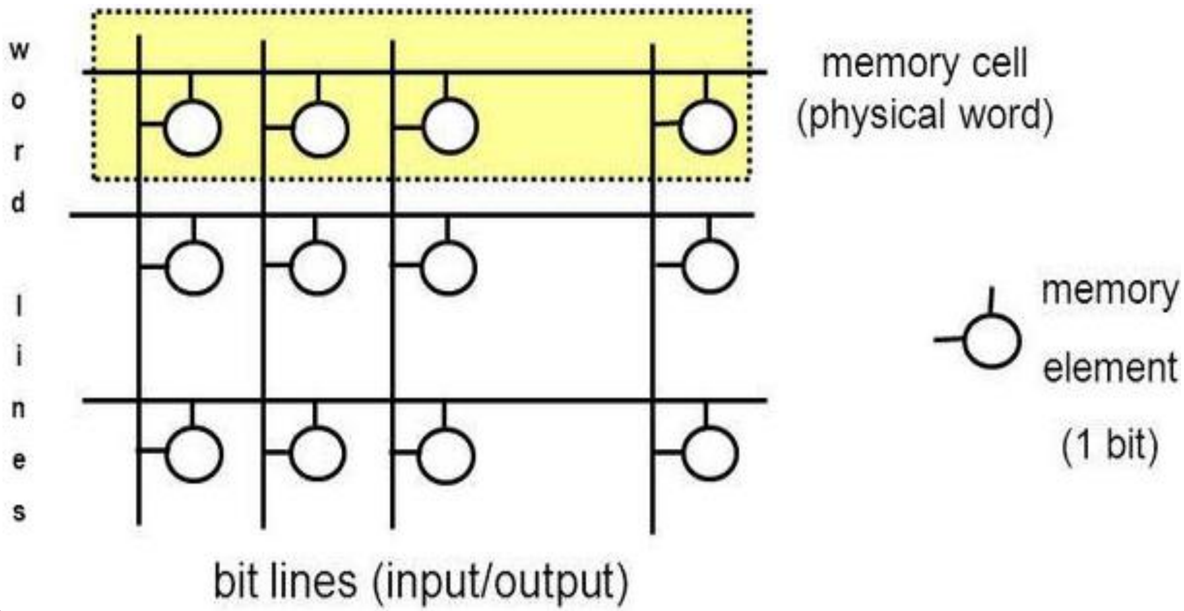
word: group of *n* bits



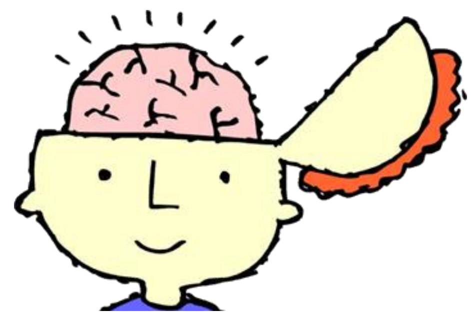
Example: A word of 16 bits

https://www.talentcookie.com/2016/04/bits-storage-unit/bytes-and-memory-basics-of-computer-

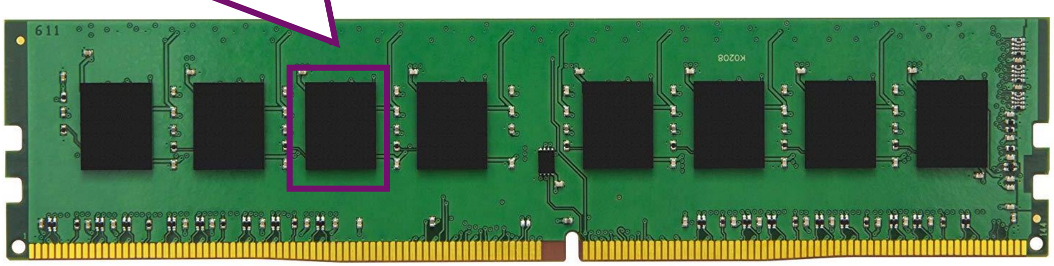
Memory Organization (2/2)



Question: How to access the contents of memory?



DDR4
RAM



http://www.myshared.ru/slide/1119020/

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.

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

Address	Location
0	1st location
1	2nd location

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

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

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

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

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

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

Example:

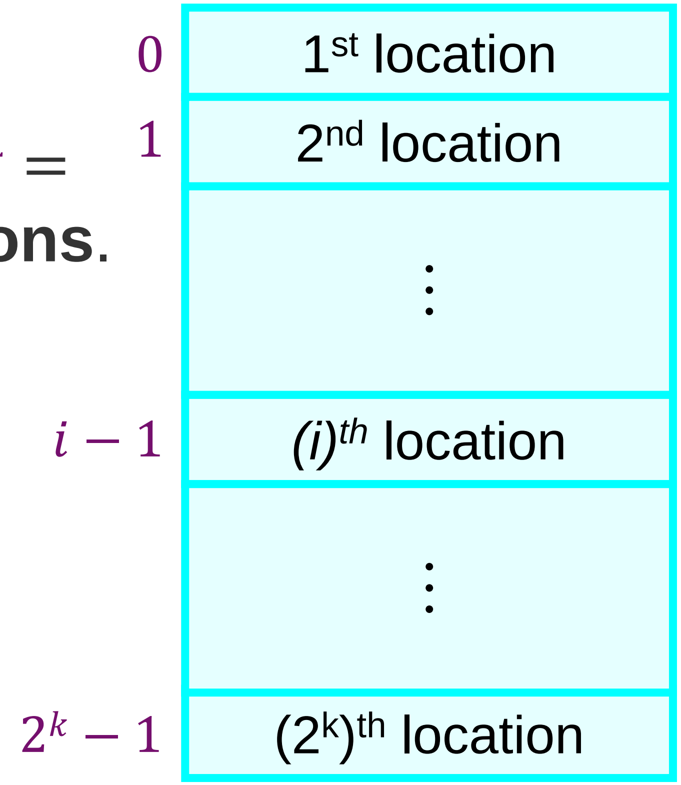
- The address "100" (k=3) denotes the memory location at

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 $2^{40} = \cdots$



Outline

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

- Memory Operations
  - Load
  - Store

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

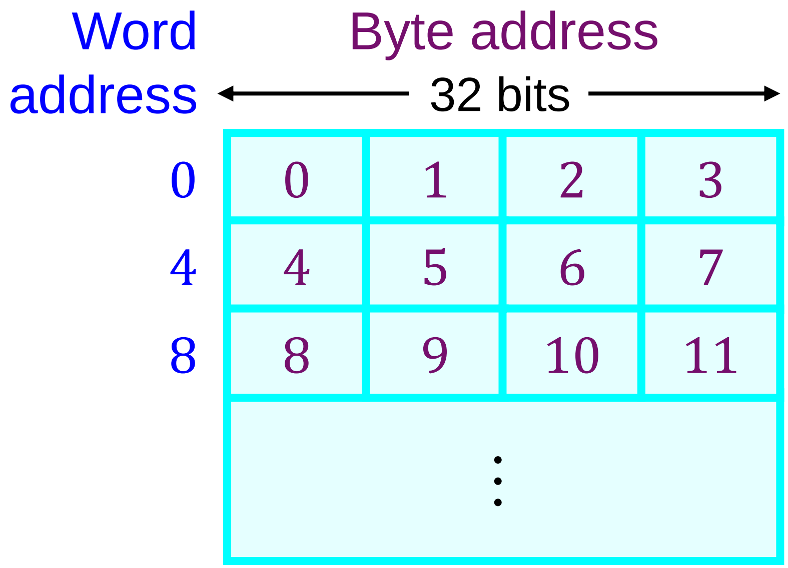
0	1st word (bit: 0~15)
1	2nd word (bit: 16~31)
2	3rd word (bit: 32~47)
	⋮

Unit size: 32-bit word

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

Byte Addressability (2/2)

- The most practical assignment: **byte** addresses
  - **Successive addresses** represents **successive byte locations** in the memory.
  - E.g. if the word length is 32 bits:
    - Byte addresses: 0, 1, 2, …
    - Word addresses: 0, 4, 8, …



- Byte addressability: Each **byte** location in the memory has its own address and is addressable.

Class Exercise 3.1

Student ID: ____________ Date:
Name: ________________ ________

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

- Given the information about the **12-inch 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: ______________________________

Outline

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

Big-Endian and Little-Endian

- Two ways to assign byte addresses across a word:
  - **Big-Endian**: Lower byte addresses are used for more significant bytes of the word (e.g. Motorola)
  - **Little-Endian**: Lower byte addresses are used for less significant bytes of the word (e.g. Intel)

Word address | Byte address (← 32 bits →)

Word address				
0	0	1	2	3
4	4	5	6	7
	⋮			
2^k-4	2^k-4	2^k-3	2^k-2	2^k-1

Big-Endian

Word address | Byte address (← 32 bits →)

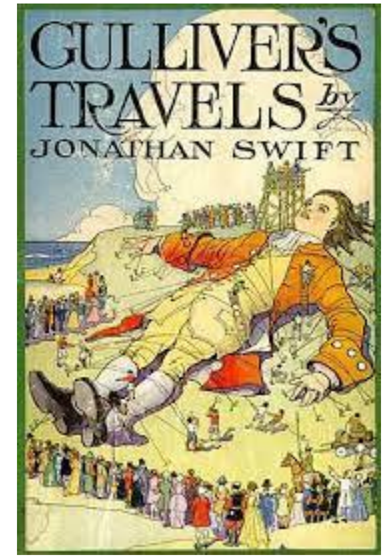
Word address				
0	3	2	1	0
4	7	6	5	4
	⋮			
2^k-4	2^k-1	2^k-2	2^k-3	2^k-4

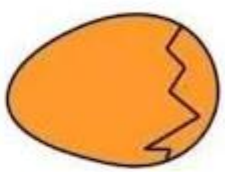
Little-Endian

- *Note: The words "more significant" and "less significant" are used in relation to the weights (powers of 2) assigned to bits when the word represents a number.*

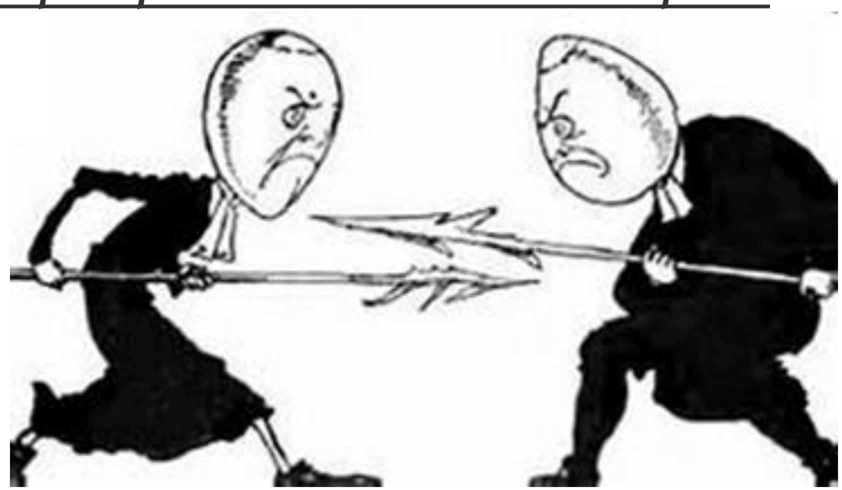
Fun Knowledge about "Endian"

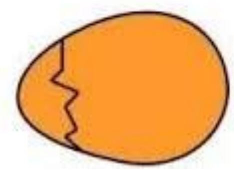
- The word "endian" were drawn from Jonathan Swift's 1726 satire, "Gulliver's Travels".
  - *In which, civil war erupts over whether <u>the big end</u> or <u>the little end</u> of a boiled egg is <u>the proper end to crack open</u> ...*





Big-Endian





Little-Endian

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

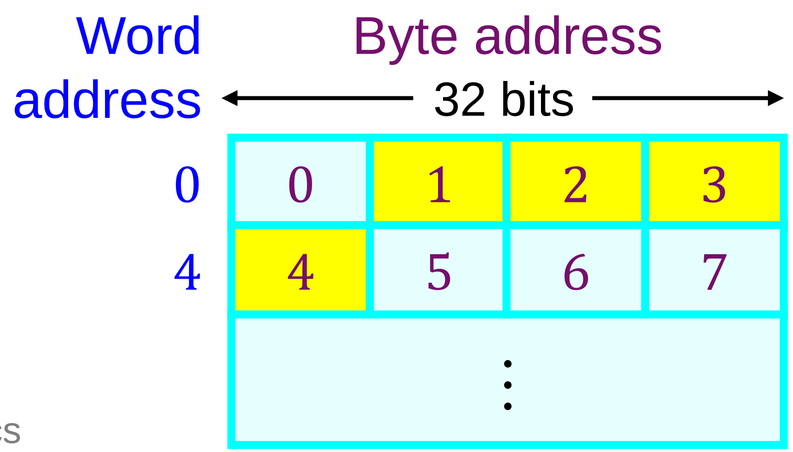
https://en.wikipedia.org/wiki/Endianness
http://fcpacompliancereport.com/2015/02/gullivers-travels-truth-or-fiction/
https://jordanramseyblog.wordpress.com/2017/04/03/gullivers-travels-part-1chapter-4/
https://www.talentcookie.com/2016/04/endianness-little-or-big/egg-endian-2/

Outline

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

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.

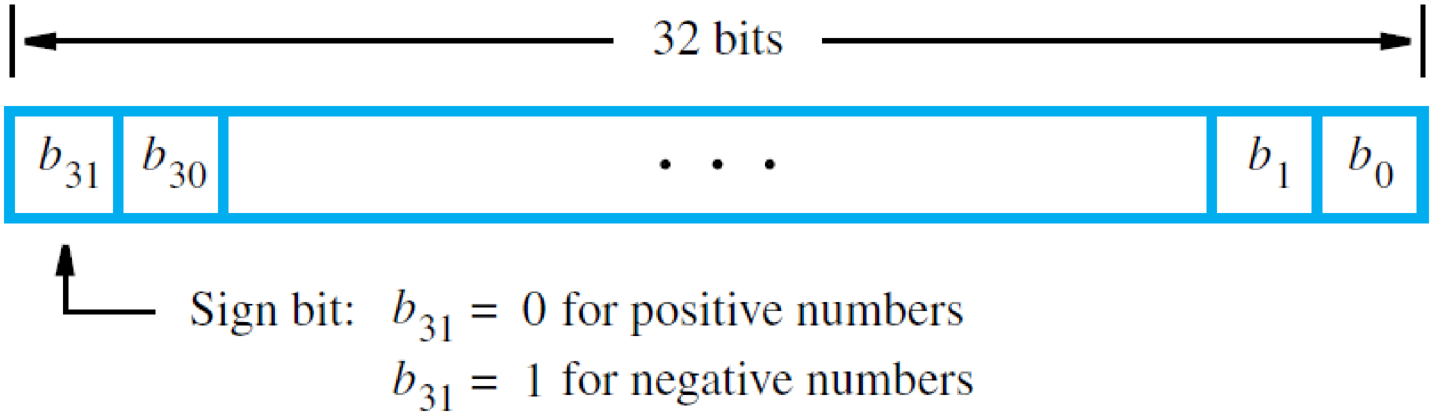


Outline

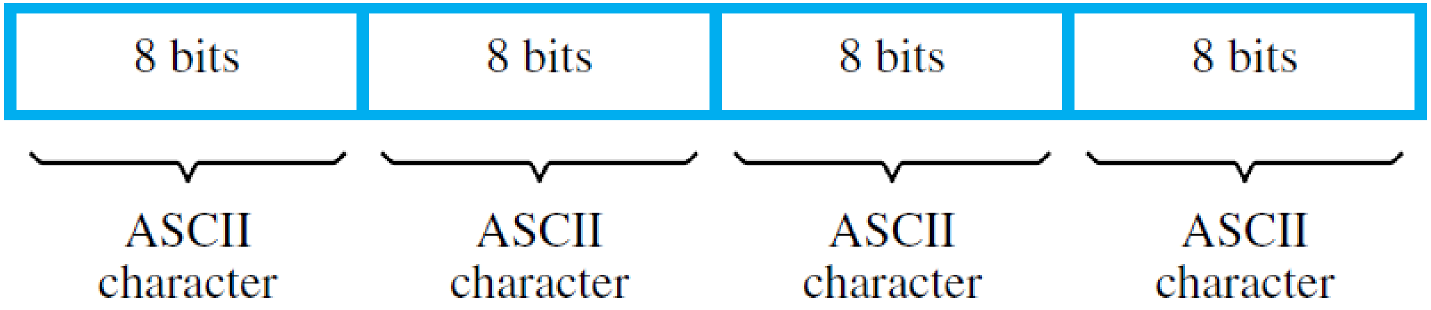
- Memory Locations and Addresses
  - Memory Organization and Address
  - Byte Addressability
  - Big-Endian and Little-Endian Assignments
  - Word Alignment
  - Accessing Numbers, Characters, and Strings
- Memory Operations
  - `Load`
  - `Store`

Accessing Numbers and Characters

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



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



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.

Class Exercise 3.2

- Consider a computer with
  - **Byte-addressable** memory
  - **32-bit words**
  - **Little-endian** scheme
- 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]

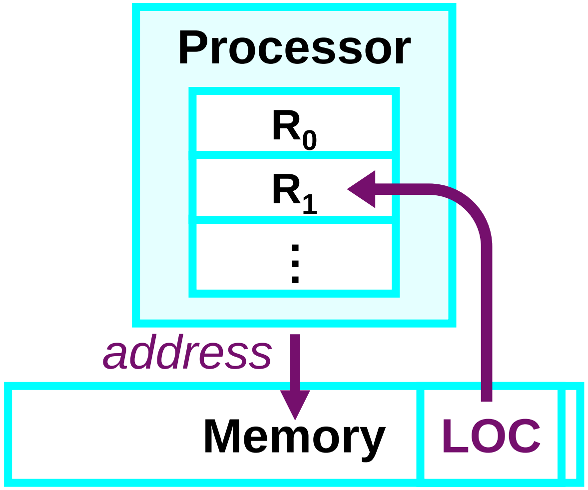
Outline

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

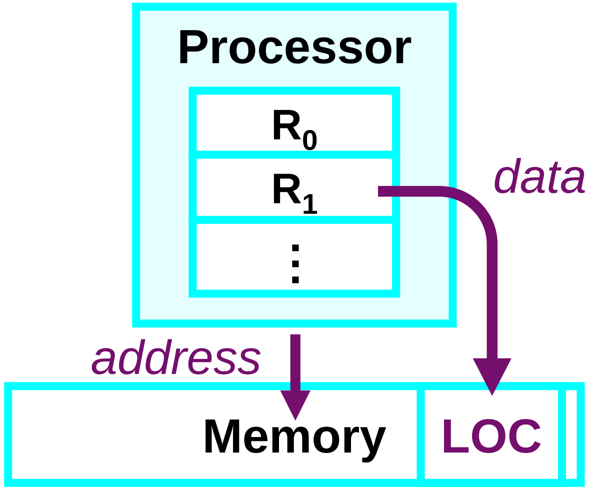
- Memory Operations
  - `Load`
  - `Store`

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 ← [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] ← R1



Summary

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

- Memory Operations
  - `Load`
  - `Store`