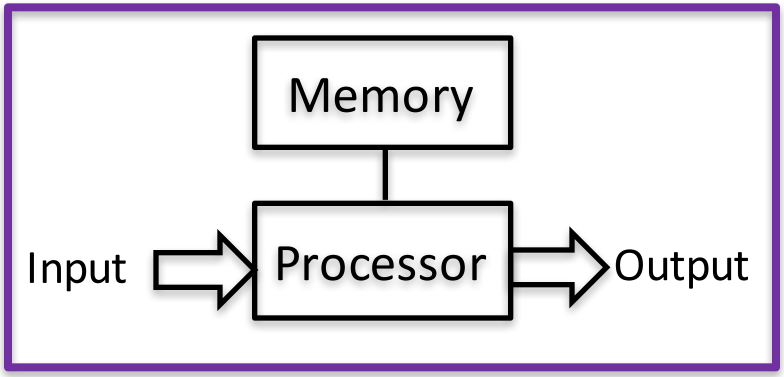
SISTEMI EMBEDDED

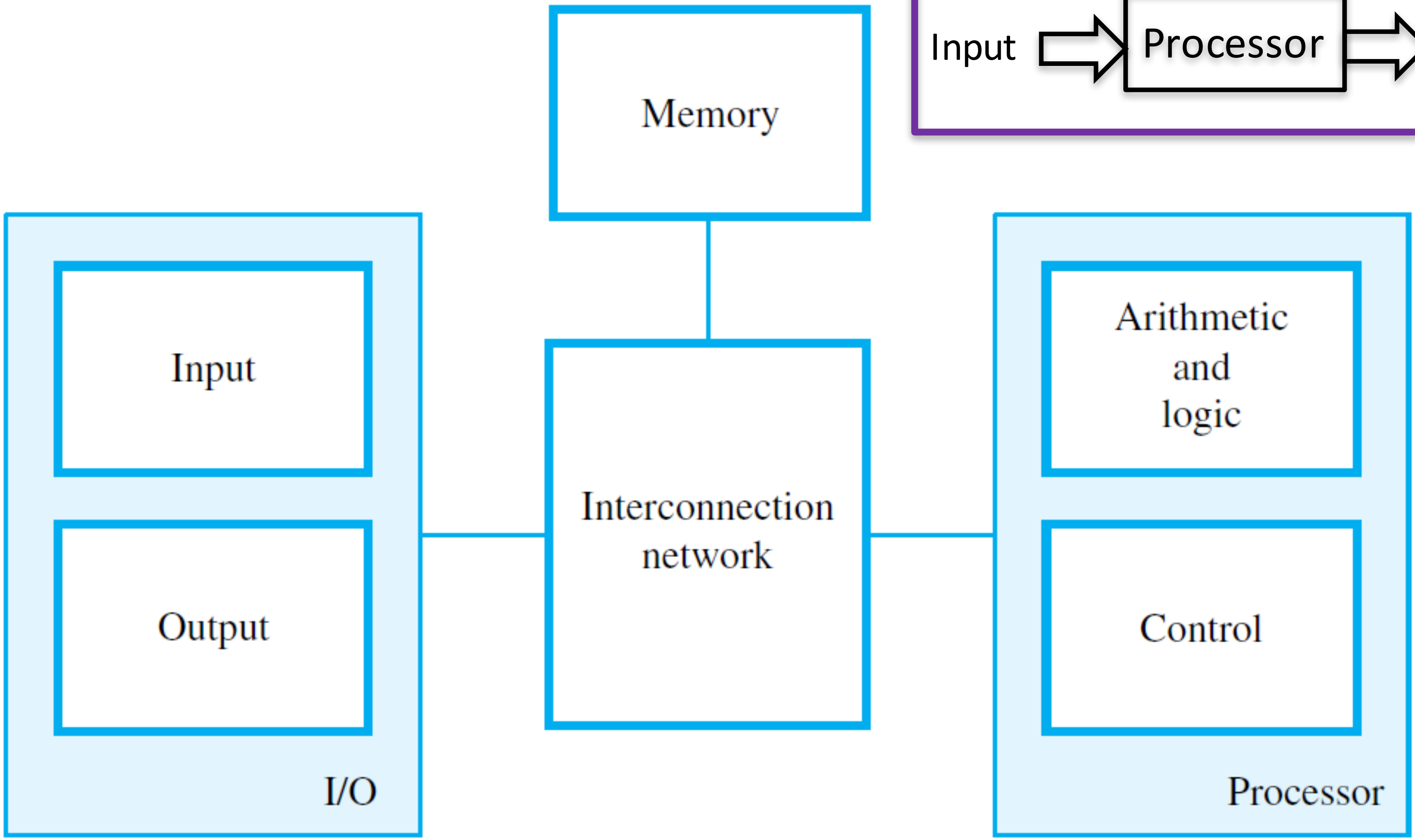
Basic Concepts about Computers

Federico Baronti

Last version: 20160301

Functional Units of a Computer





Instructions and Programs

- An instruction specifies an operation and the locations of its data operands
- A 32-bit word typically holds one encoded instruction
- A sequence of instructions, executed one after another, constitutes a program
- Both a program and its data are stored in the main memory

Instruction types

- Three basic instruction types:
  - Load: Read a data operand from memory or an input device into the processor
  - Store: Write a data operand from a processor register to memory or an output device
  - Operate: Perform an arithmetic or logic operation on data operands in processor registers
  - Branch: Alter if a condition is verified the sequential execution of the instructions

Program Example

- A, B, and C, are labels representing memory word addresses; Ri are processor registers

- A program for the calculation

  C = A + B

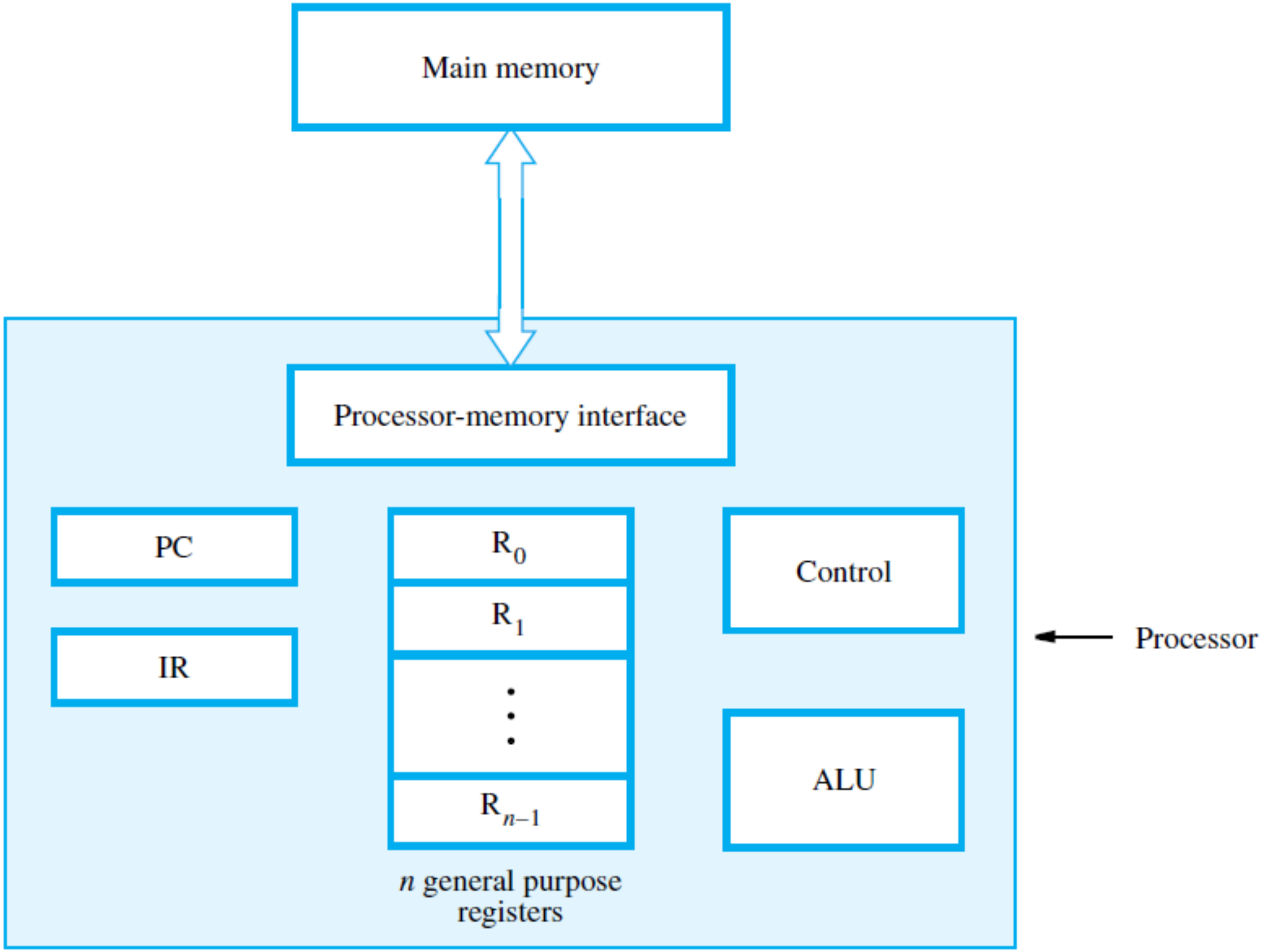
  is:

```
Load    R2, A
Load    R3, B
Add     R4, R2, R3
Store   R4, C
```

Main Processor Elements (1)

- The program counter (PC) register holds the memory address of the current instruction
- The instruction register (IR) holds the current instruction
- General-purpose registers hold data and addresses
- Control circuits and the arithmetic and logic unit (ALU) fetch and execute instructions

Main Processor Elements (2)



Fetching and executing instructions

Example: Load R2, LOC

The processor control circuits do the following:

- Send address in PC to memory; issue Read
- Load instruction from memory into IR
- Increment PC to point to next instruction
- Send address LOC to memory; issue Read
- Load word from memory into register R2

Representation of Information

- Whatever is the source of information, data are represented by an array of bits (usually in a number multiple of 8, i.e. 1 BYTE)
- An array of bits directly represents a **Natural number in base 2** (positional binary notation)
  - $B = b_{n-1} \ldots b_1 b_0$ represents the number $V(B) = b_{n-1} \times 2^{n-1} + \ldots b_1 \times 2^1 + b_0 \times 2^0$
- Any other information can be encoded by a Natural using a specific representation
  - E.g. signed numbers, floating point numbers, chars,...
  - Representations typically use 1, 2, 4, 8 BYTES

Signed Numbers (1)

For signed integers, the leftmost bit (MSB) is used to indicate the sign:

0 for positive

1 for negative

There are three ways to represent signed integers:

- Sign and magnitude
- 1's complement
- **2's complement** (the MSB has weight -2^{n-1})

Signed Numbers (2)

B	Values represented		
$b_3 b_2 b_1 b_0$	Sign and magnitude	1's complement	2's complement
0 1 1 1	+ 7	+ 7	+ 7
0 1 1 0	+ 6	+ 6	+ 6
0 1 0 1	+ 5	+ 5	+ 5
0 1 0 0	+ 4	+ 4	+ 4
0 0 1 1	+ 3	+ 3	+ 3
0 0 1 0	+ 2	+ 2	+ 2
0 0 0 1	+ 1	+ 1	+ 1
0 0 0 0	+ 0	+ 0	+ 0
1 0 0 0	− 0	− 7	− 8
1 0 0 1	− 1	− 6	− 7
1 0 1 0	− 2	− 5	− 6
1 0 1 1	− 3	− 4	− 5
1 1 0 0	− 4	− 3	− 4
1 1 0 1	− 5	− 2	− 3
1 1 1 0	− 6	− 1	− 2
1 1 1 1	− 7	− 0	− 1

Signed Numbers (3)

2's-complement representation is used in current computers

Consider a four-bit signed integer example, where the value +5 is represented as:

0 1 0 1

To form the value -5, complement all bits of

0 1 0 1 to obtain 1 0 1 0

and then add 1 to obtain

1 0 1 1

Signed Numbers (4)

Replicate the sign bit to extend
4-bit signed integers to 8-bit signed integers

0 1 0 1 ➡ 0 0 0 0 0 1 0 1

1 1 1 0 ➡ 1 1 1 1 1 1 1 0

Character Encoding

- American Standard Code for Information Interchange (ASCII)
- Uses 7-bit codes (extended version 1 BYTE)
- Some examples:

character	binary code	(decimal,	0x hexadecimal)
A	1 0 0 0 0 0 1	(65,	0x41)
a	1 1 0 0 0 0 1	(97,	0x61)
0	0 1 1 0 0 0 0	(48,	0x30)
1	0 1 1 0 0 0 1	(49,	0x31)
9	0 1 1 1 0 0 1	(57,	0x39)

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

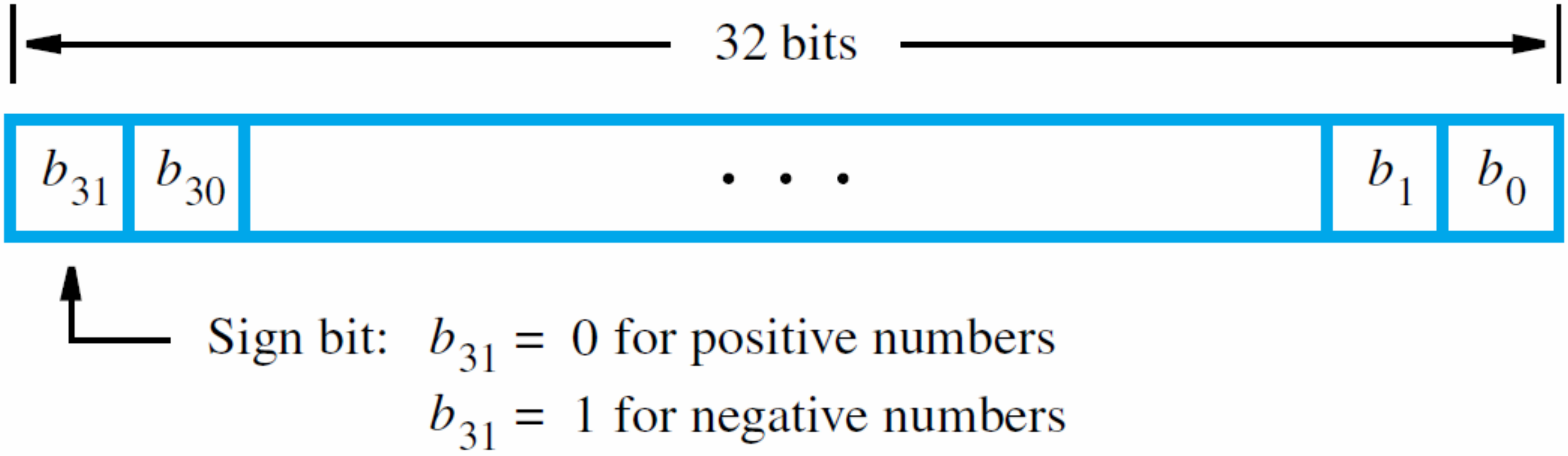
Source: www.LookupTables.com

Memory Organization

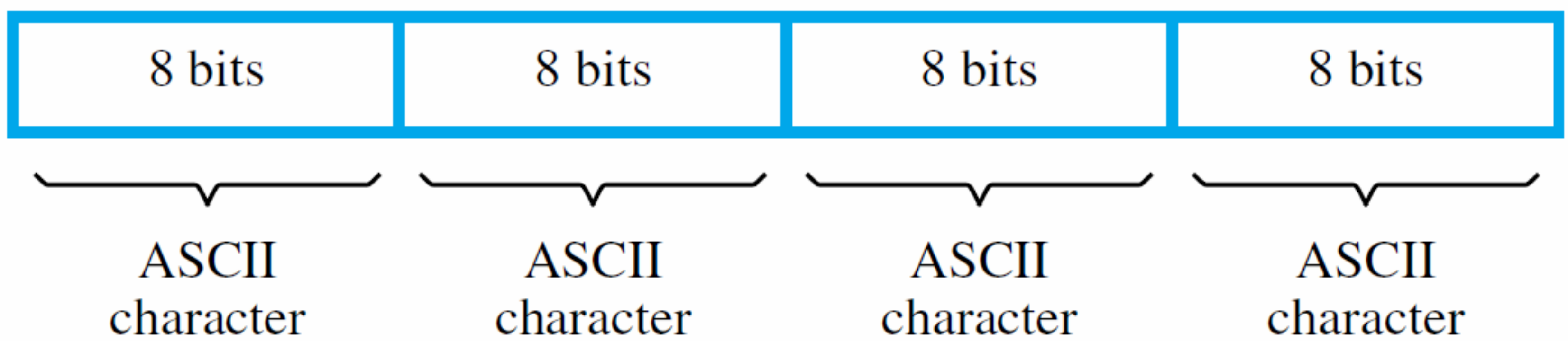
- Memory consists of many millions of cells
- Each cell holds a bit of information, 0 or 1
- Information is usually handled in larger units
- A word is a group of *n* bits
- Word length can be 16 to 64 bits
- Memory is a collection of consecutive words of the size specified by the word length

Word and Byte Encoding

- A common word length is 32 bits
- Such a word can store a 32-bit signed integer or four 8-bit bytes (e.g., ASCII characters)
- For 32-bit integer encoding, bit b_{31} is sign bit
- Words in memory may store data or machine instructions for a program
- Each machine instruction may require one (or more consecutive words for encoding)



(a) A signed integer



(b) Four characters

Addresses for Memory Location

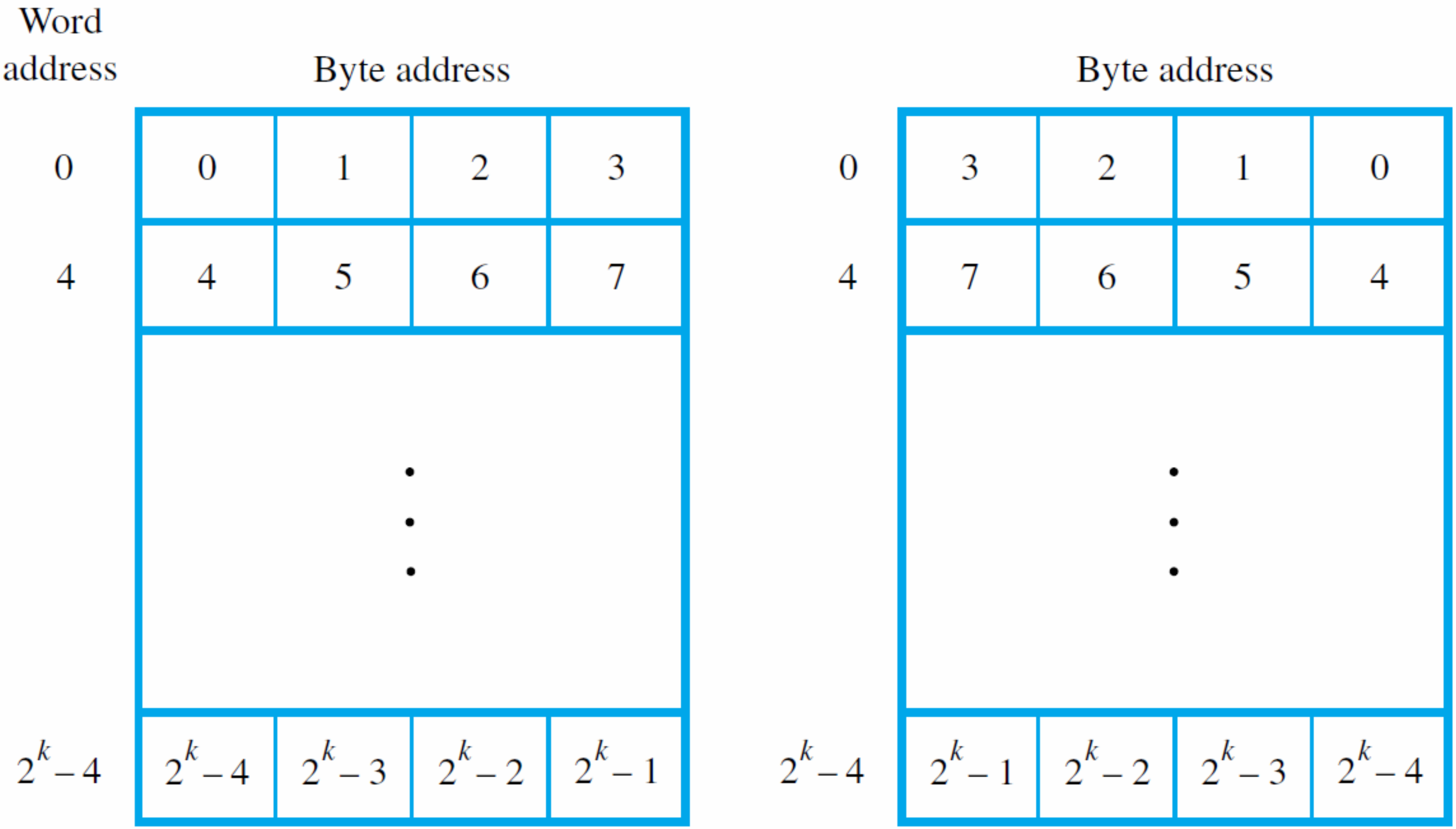
- To store or retrieve items of information, each memory location has a distinct address
- Numbers 0 to $2^k - 1$ are used as addresses for successive locations in the memory
- The 2^k locations constitute the address space
- Memory size set by k (number of address bits)
- Examples: $k = 20 \longrightarrow 2^{20}$ or 1M locations,
  $k = 32 \longrightarrow 2^{32}$ or 4G locations

Byte Addressability

- Byte size is always 8 bits
- But word length may range from 16 to 64 bits
- Impractical to assign an address to each bit
- Instead, provide a byte-addressable memory that assigns an address to each byte
- Byte locations have addresses 0, 1, 2, …
- Assuming that the word length is 32 bits, word locations have addresses 0, 4, 8, …

Big- Little-Endianess

- Two ways to assign byte address across words
- Big-endian addressing assigns lower addresses to more significant (leftmost) bytes of word
- Little-endian addressing uses opposite order
- Commercial computers use either approach, and some can support both approaches
- Addresses for 32-bit words are still 0, 4, 8, …
- Bits in each byte labeled b_7 … b_0, left to right



(a) Big-endian assignment

(b) Little-endian assignment

Word Alignment

- # of bytes per word is normally a power of 2
- Word locations have aligned addresses if they begin at byte addresses that are multiples of the number of bytes in a word
- Examples of aligned addresses:
  2 bytes per word → 0, 2, 4, ...
  8 bytes per word → 0, 8, 16, ...
- Some computers permit unaligned addresses