Bits & Bytes

Computer Systems Organization

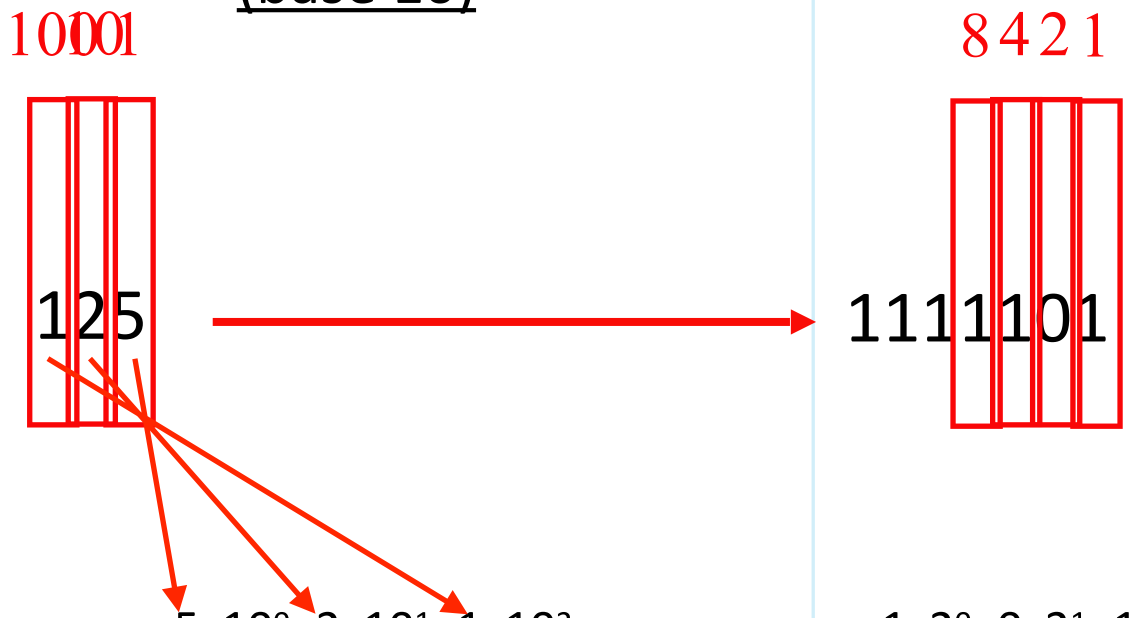
01000111
01100101
01100101
01101011
And if you can read
that you are too.

Why Don’t Computers Use Decimal Numbers?

Decimal representation (base 10): Natural representation used by humans

Decimal representation (base 10)

Binary representation (base 2)



$= 5x10^{0}+2x10^{1}+1x10^{2}$

$= 1x2^{0}+0x2^{1}+1x2^{2}+1x2^{3}+1x2^{4}+1x2^{5}+1x2^{6}+1x2^{7}$

Why Don’t Computers Use Base 10?

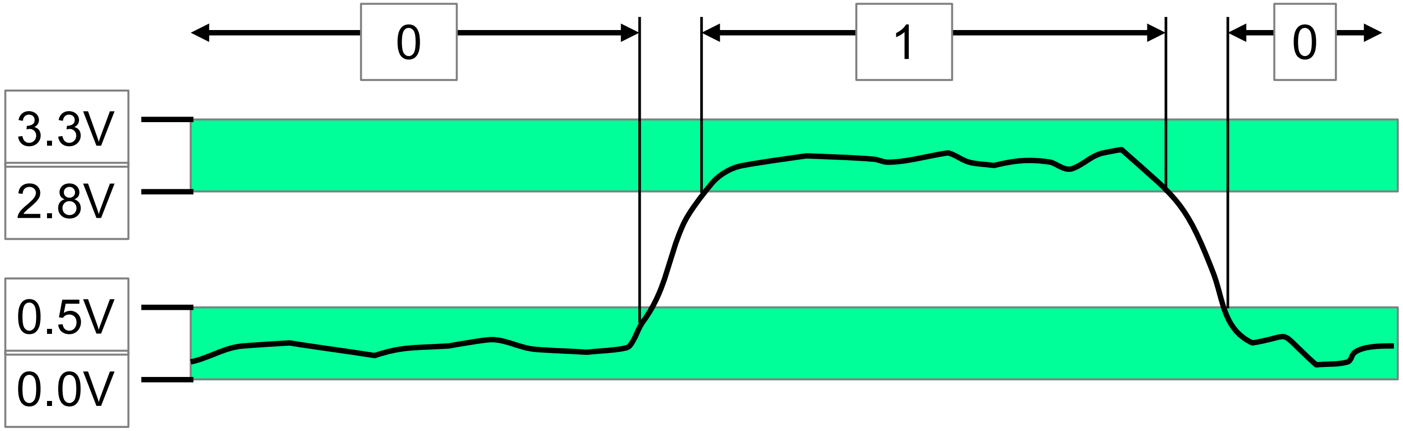
Question: why don’t computers use decimal numbers?

Implementing electronically

- Hard to represent the signal: need high precision to encode 10 signal levels on single wire

Solution

- Use a binary (0/1) representation of numbers/data



How to Calculate Binary Representations?

Binary representation of 75

Subtraction method (or division by 2 method with the last reminder most significant)

- What is the biggest multiple of two that is less or equal than 75?
- 75 - 64 = 11
- What is the biggest multiple of two that is less or equal than 11?
- 11 - 8 = 3
- 3 - 2 = 1
- 1 - 1 = 0

Keep the multiples of 2 in mind

256	128	$\boxed{64}$	32	16	$\boxed{8}$	4	$\boxed{2}$	$\boxed{1}$
0	0	1	0	0	1	0	1	1

How to Calculate Binary Representations?

Binary representation of 339

Subtraction method

- 339 - 256 = 83
- 83 - 64 = 19
- 19 - 16 = 3
- 3 - 2 = 1
- 1 - 1 = 0

Multiples of 2

512	$\boxed{256}$	128	$\boxed{64}$	32	$\boxed{16}$	8	4	$\boxed{2}$	$\boxed{1}$
	1	0	1	0	1	0	0	1	1

Small Binary Numbers

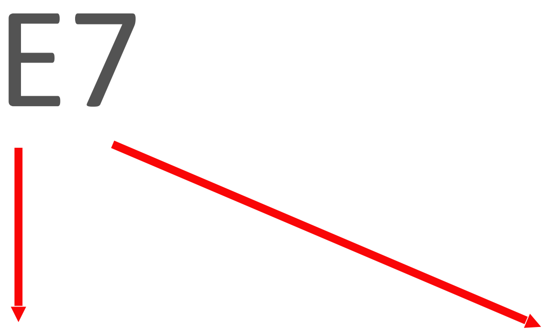
Useful to keep in mind

Decimal	Binary
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
10	1010
11	1011
12	1100
13	1101
14	1110
15	1111

Hexadecimal Numbering System (Base 16)

- Binary (base 2): 0 1
- Decimal (base 10): 0 1 2 3 4 5 6 7 8 9
- Hexadecimal (base 16): 0 1 2 3 4 5 6 7 8 9 A B C D E F

  10 11 12 13 14 15



E7

E x 16^1

7 x 16^0 = 7

14 x 16^1 = 224

E7 = 224 + 7 = 231

You can write this hexadecimal number in C as 0xE7 Or 0xe7

Hex	Decimal
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
A	10
B	11
C	12
D	13
E	14
F	15

Hexadecimal <-> Binary

- Hexadecimal -> Binary
  - Expand each hexadecimal digit to its binary equivalent

1 7 3 A 4 C

- Binary -> Hexadecimal
  - Split it into groups of 4 bits each, then convert each group to the hexadecimal equivalent

Hex	Decimal	Binary
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
A	10	1010
B	11	1011
C	12	1100
D	13	1101
E	14	1110
F	15	1111

Try to memorize this table

Practice

- Calculate the binary representation of

5 3 A 1 F E

Hex	Decimal	Binary
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
A	10	1010
B	11	1011
C	12	1100
D	13	1101
E	14	1110
F	15	1111

Decimal <—> hexadecimal

Decimal to Hexadecimal (division by 16 with the last reminder as most significant):

314,156 = 19,634 x 16 + 12 (C)
19,634 = 1,227 x 16 + 2 (2)
1,227 = 76 x 16 + 11 (B)
76 = 4 x 16 + 12 (C)
4 = 0 x 16 + 4 (4)
0x4CB2C

Hexadecimal to Decimal:

0x7AF
7 x 16pow2 + 10 x 16pow1 + 15 x 16pow0 = 7 x 256 + 10 x 16 + 15 = 1,792 + 160 + 15 = 1,967

Bits, Bytes and Words

Bit = 0 or 1

Byte = 8 bits

Binary	00000000_2	to	11111111_2
Decimal:	0_{10}	to	255_{10}
Hexadecimal	00_{16}	to	FF_{16}

Word = multiple bytes

- E.g., 4 bytes (32 bits) or 8 bytes (64 bits)
- Size of pointer data usually determines the number of bytes in a word
- W-bit word size
  - => The virtual addresses can range from 0 to $2^w - 1$ bytes
- Older machines are 32 bits (4 bytes)
  - Limits addresses to 4GB
  - Becoming too small for memory-intensive applications
- Today systems are 64 bits (8 bytes)
  - Can address 1.84 X 10^{19} bytes

Data Representations

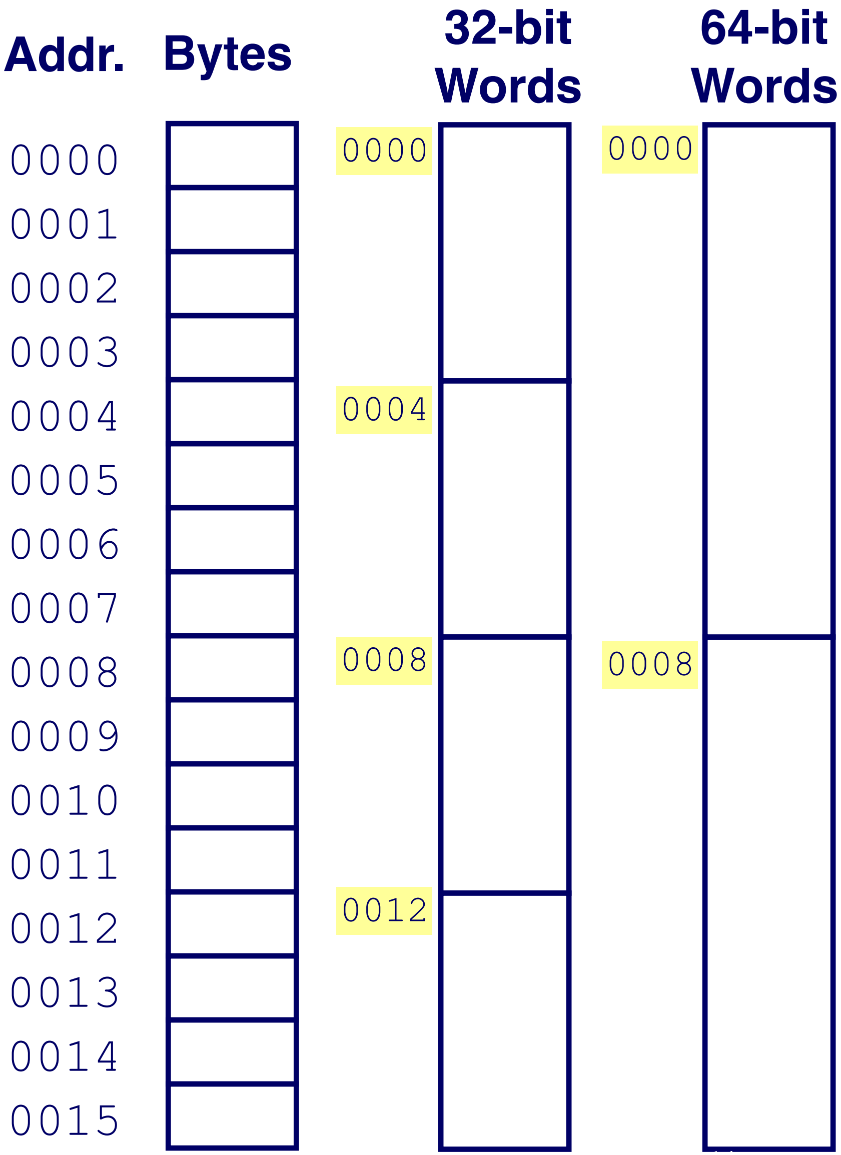
Sizes of C objects (in Bytes)

C Data Types	32-bit machine	64-bit
char	1	1
short	2	2
int	4	4
long int	4	8
int32_t	4	4
int64_t	8	8
float	4	4
double	8	8
Pointer	4	8

Word-Oriented Memory Organization

Memory is a sequence of bytes

- The byte is the smallest addressable unit in memory
- Each byte has an address
- Addresses of successive words differ by 4 (32-bit) or 8 (64-bit)

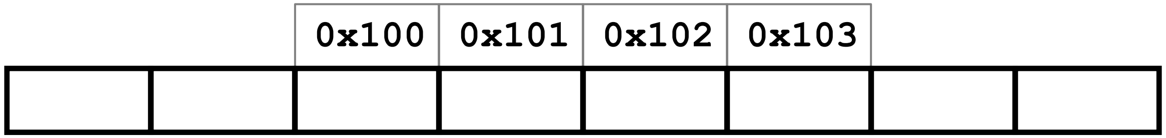


Byte Ordering

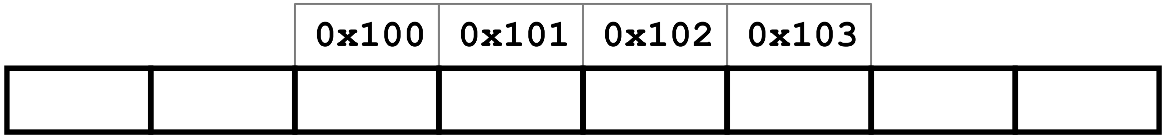
Example

- Variable `x` has 4-byte representation `0x01234567`
- Address given by `&x` is `0x100`

How should x be stored in memory?

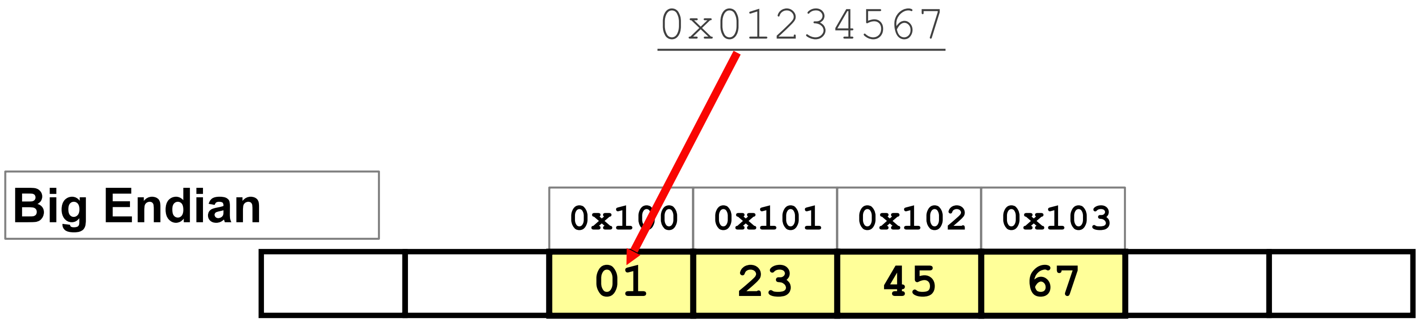


0x100 0x101 0x102 0x103



0x100 0x101 0x102 0x103

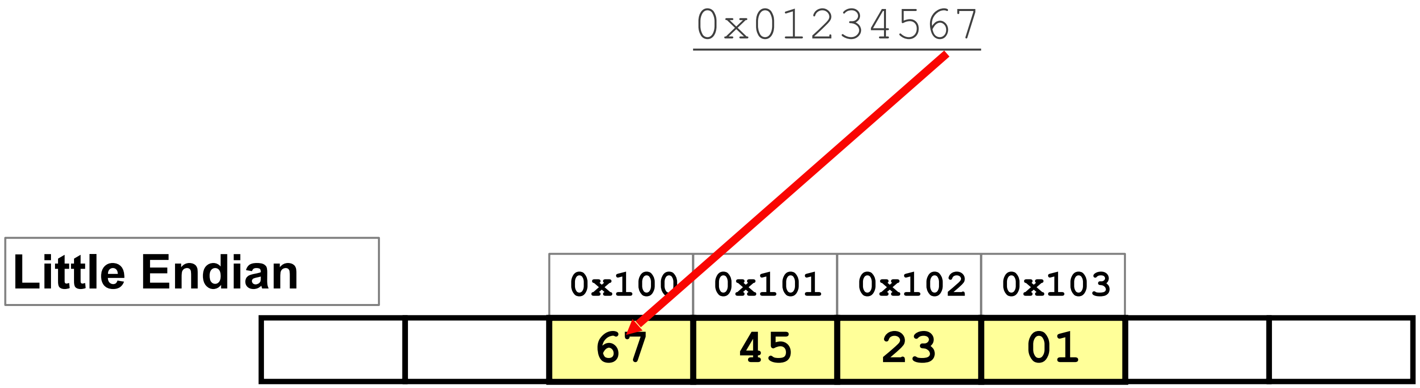
Byte Ordering



0x01234567

Big Endian

		0x100	0x101	0x102	0x103		
		01	23	45	67		



0x01234567

Little Endian

		0x100	0x101	0x102	0x103		
		67	45	23	01		

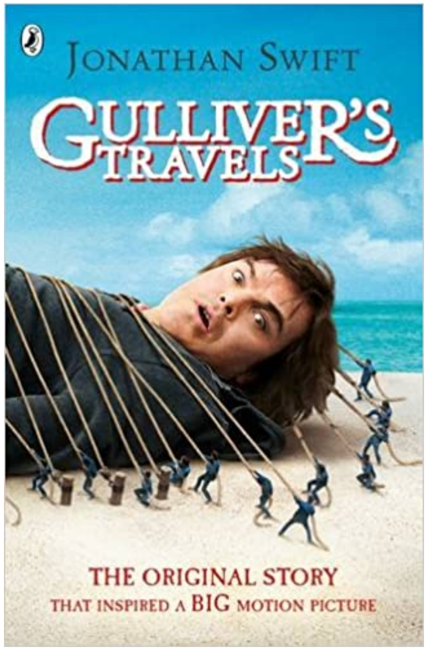
Byte Ordering Example

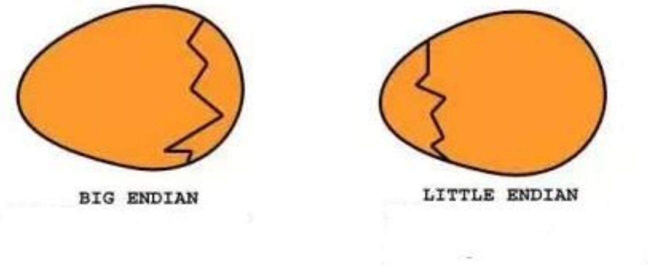
Big Endian

- Most significant byte has lowest address
- Sun’s, Mac’s (PowerPC)

Little Endian

- Least significant byte has lowest address
- PC’s (x86), Mac's (x86), Alphas, ARM with iOS and Android





Examining Data Representations

Code to Print Byte Representation of Data

- Casting pointer to `unsigned char *` creates byte array

```
typedef unsigned char * pointer;

void show_bytes(pointer start, int len)
{
  int i;
  for (i = 0; i < len; i++)
    printf("%p \t %.2x\n",start+i, start[i]);
  printf("\n");
}
```

Printf directives:

%p: Print pointer

%x: Print Hexadecimal

show_bytes Execution Example

```
int a = 15213;
printf("%d is”, a);
printf(“%x in hexadecimal \n”, a);
show_bytes((pointer) &a, sizeof(int));
```

```
15213 is 0x00003b6d in hexadecimal
```

Result (Linux, Intel X86):

```
int a = 15213;
0x11ffffcb8   6d
0x11ffffcb9   3b
0x11ffffcba   00
0x11ffffcbb   00
```

Result (Solaris, Sun SPARC):

```
int a = 15213;
0x11ffffcb8   00
0x11ffffcb9   00
0x11ffffcba   3b
0x11ffffcbb   6d
```

Representing Strings in C

```
char S[6] = "15213";
```

- Represented by array of characters
- Strings end with a null character
  - Final character = 0
- Each character encoded in ASCII format
  - ASCII = American Standard Code for Information Interchange
  - Standard 7-bit encoding of character set
  - Other encodings exist, but uncommon

Representing Strings in C

ASCII TABLE

Hex	Decimal	Binary
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
A	10	1010
B	11	1011
C	12	1100
D	13	1101
E	14	1110
F	15	1111

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
		[ENQUIRY]	37	25	%	69	45	E	101	65	e
		[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
		[BELL]	39	27	'	71	47	G	103	67	g
						72	48	H	104	68	h
						73	49	I	105	69	i
						74	4A	J	106	6A	j
						75	4B	K	107	6B	k
						76	4C	L	108	6C	l
						77	4D	M	109	6D	m
						78	4E	N	110	6E	n
						79	4F	O	111	6F	o
						80	50	P	112	70	p
						81	51	Q	113	71	q
						82	52	R	114	72	r
						83	53	S	115	73	s
						84	54	T	116	74	t
						85	55	U	117	75	u
						86	56	V	118	76	v
						87	57	W	119	77	w
						88	58	X	120	78	x
						89	59	Y	121	79	y
						90	5A	Z	122	7A	z
						91	5B	[	123	7B	{
						92	5C	\	124	7C	|
						93	5D	]	125	7D	}
						94	5E	^	126	7E	~
		[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

Practice

Write a function that takes a character as input and returns whether it is a digit:

isdigit(‘4’) ==> 1

isdigit(‘9’) ==> 1

isdigit(‘a’) ==> 0

isdigit(‘?’) ==> 0

Try to write isdigit()!

```
int isdigit(char c)
{
  if ((c >=‘0') && (c <= ‘9’))
        return 1;
  else
        return 0;
}
```

Machine-Level Code Representation

- A program as a sequence of instructions
- Each instruction is a simple operation
    - Arithmetic operation
    - Read or write memory
    - Conditional branch
- Instructions encoded as bytes

```
int sum(int x, int y)
{
   return x+y;
}
```

- Different instruction encodings for different machines
    - Most code not binary compatible

Programs are Byte Sequences Too!

PC sum
55
89
E5
8B
45
0C
03
45
08
89
EC
5D
C3

Practice

Write a C function that finds the length of a string

```c
int strlen(char * text)
{
    int index= 0;

    while(text[index] != '\0')
    {
        index++;
    }

    return index;
}
```