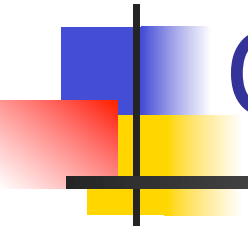


How Numbers are Stored in Computers

Heavily borrowed from:
Foundations of Computer Science
(Cengage Learning)

Computer Architecture

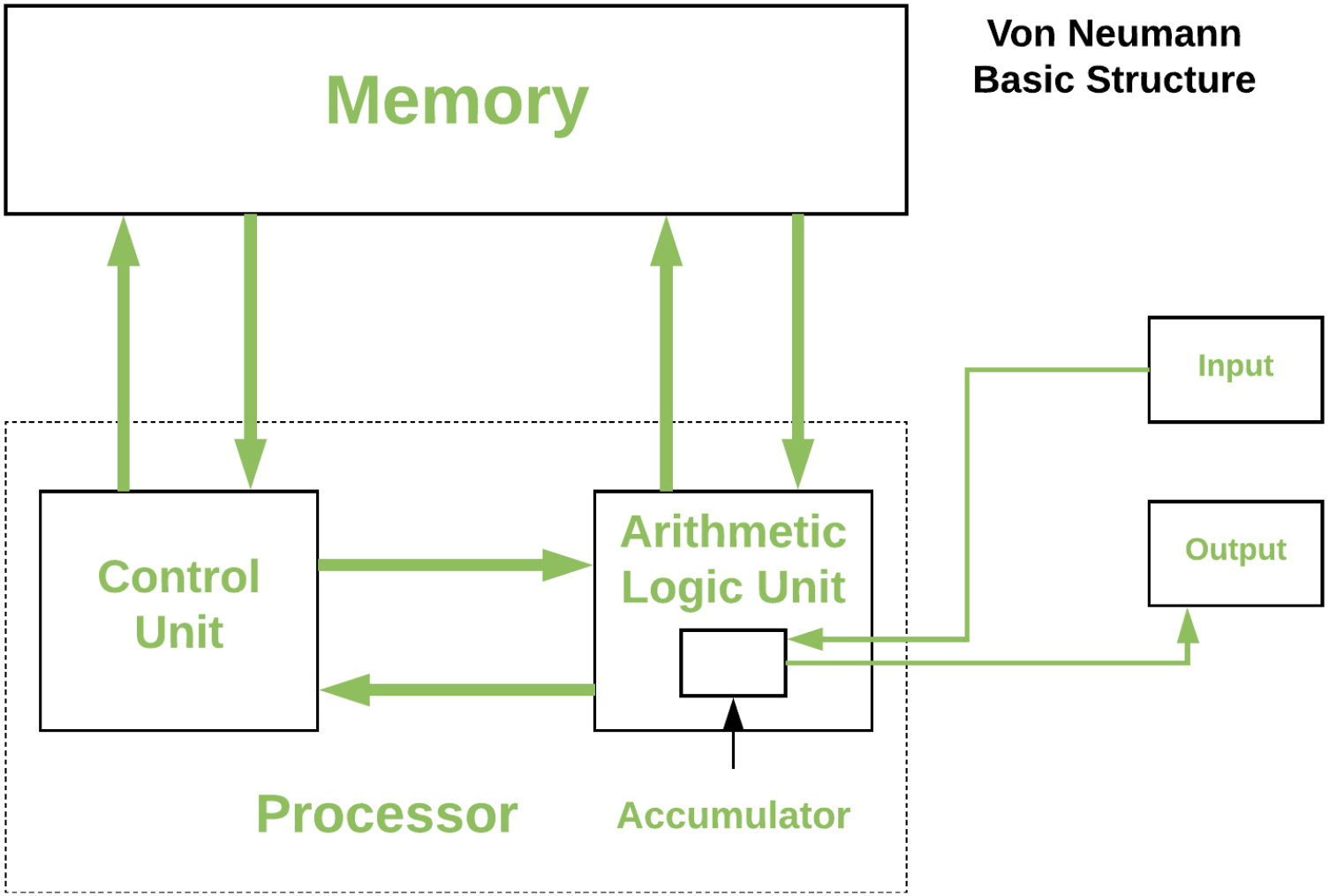


TABLE 4-3
A *machine code* program for adding 1234 and 4321. This is the lowest level of programming: direct manipulation of the digital electronics. (The right column is a continuation of the left column).

```
10111001        00000000
11010010        10100001
00000100        00000000
10001001        00000000
00001110        10001011
00000000        00011110
00000000        00000010
10111001        00000000
11100001        00000011
00010000        11000011
10001001        10100011
00001110        00000100
00000010        00000000
```

TABLE 4-4
An *assembly* program for adding 1234 and 4321. An *assembler* is a program that converts an assembly program into machine code.

```
MOV CX,1234         ;store 1234 in register CX, and then
MOV DS:[0],CX       ;transfer it to memory location DS:[0]

MOV CX,4321         ;store 4321 in register CX, and then
MOV DS:[2],CX       ;transfer it to memory location DS:[2]

MOV AX,DS:[0]       ;move variables stored in memory at
MOV BX,DS:[2]       ;DS:[0] and DS:[2] into AX & BX

ADD AX,BX           ;add AX and BX, store sum in AX

MOV DS:[4],AX       ;move the sum into memory at DS:[4]
```

TABLE 4-5
A *BASIC* program for adding 1234 and 4321. A *compiler* is a program that converts this type of high-level source code into machine code.

```
100  A = 1234
110  B = 4321
120  C = A+B
130  END
```

Low Level Languages

- Machine level language: Just above working with actual electronic circuits.
- Registers: All microprocessors are based around a group of flip-flops that can store a series of ones and zeros.
- The 8088 microprocessor has four general purpose registers, each consisting of 16 bits. These are identified by the names: AX, BX, CX, and DX.
- There are also nine additional registers with special purposes, called: SI, DI, SP, BP, CS, DS, SS, ES, and IP.
- IP, the Instruction Pointer, keeps track of where in memory the next instruction resides.
- Assembly level language patterns of ones and zeros are assigned names according to the function they perform.

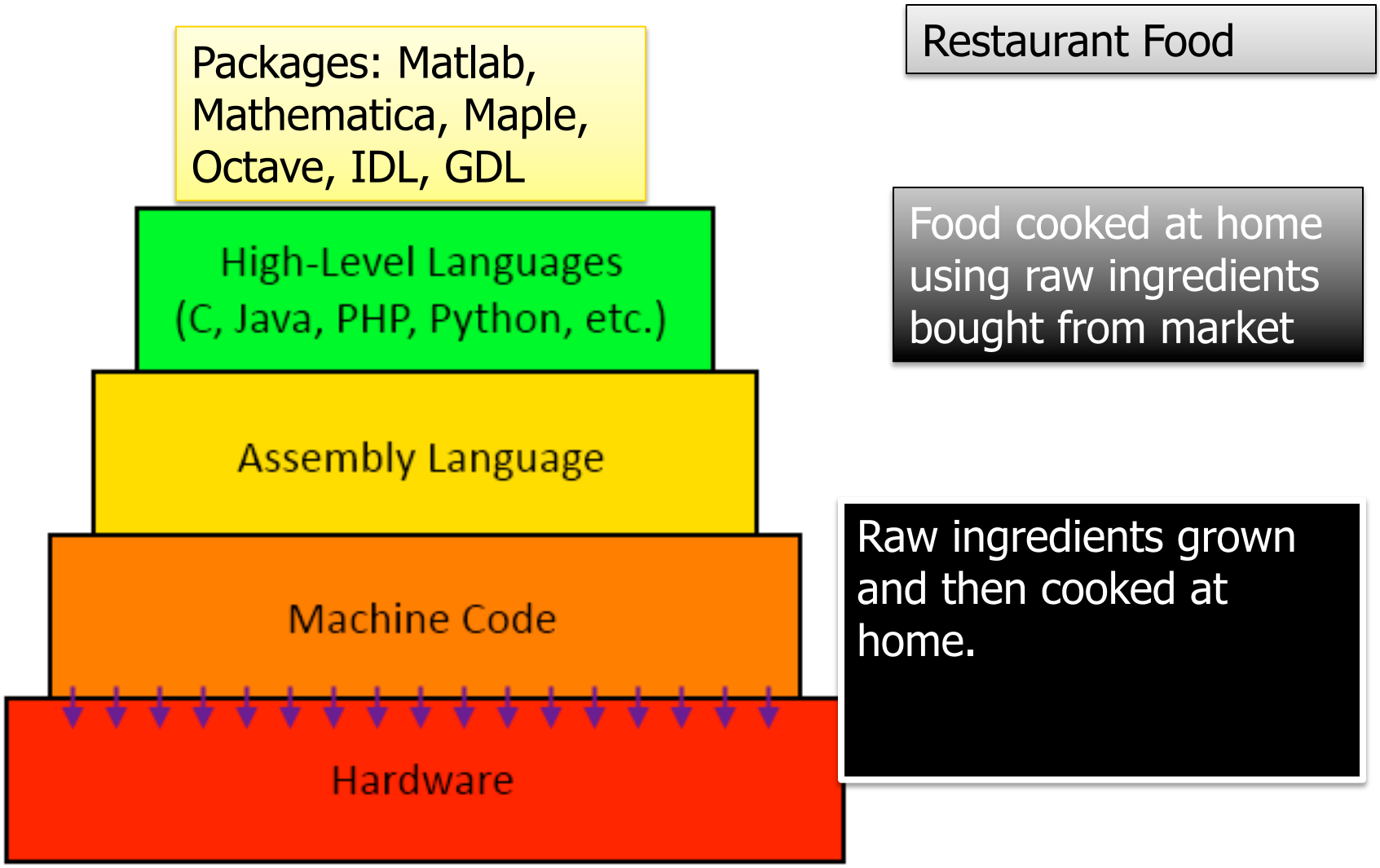
- The instruction below tells an x86/IA-32 processor to move an immediate 8-bit value into a register.
- The binary code for this instruction is 10110 followed by a 3-bit identifier for which register to use. The identifier for the AL register is 000, so the following machine code loads the AL register with the data 01100001.

```
10110000 01100001 ; Load AL with 97
```

- Assembly language for the 8086 family provides the mnemonic MOV (an abbreviation of move) for instructions such as this, so the machine code above can be written as follows in assembly language, which is much easier to read and to remember.
- `MOV AL, 61h ; Load AL with 97`

Higher Level Languages

- High-level languages isolate the programmer from the hardware.
- The source code may be transported between different types of microprocessors.
- Programmer who uses a compiled language needs to know *nothing* about the internal workings of the computer. Another programmer has assumed this responsibility, the one who wrote the compiler.
- In a high-level language, or a package, you are relying on the programmer who wrote the compiler to understand the best techniques for hardware manipulation.
- These programmers have never seen the particular problem you are dealing with. Therefore, they cannot always provide you with an optimal solution.


Packages: Matlab, Mathematica, Maple, Octave, IDL, GDL
High-Level Languages (C, Java, PHP, Python, etc.)
Assembly Language
Machine Code
Hardware
Restaurant Food
Food cooked at home using raw ingredients bought from market
Raw ingredients grown and then cooked at home.

Data inside the computer

All data types are transformed into a uniform representation when they are stored in a computer and transformed back to their original form when retrieved. This universal representation is called a **bit pattern**.

Figure: **A bit pattern**

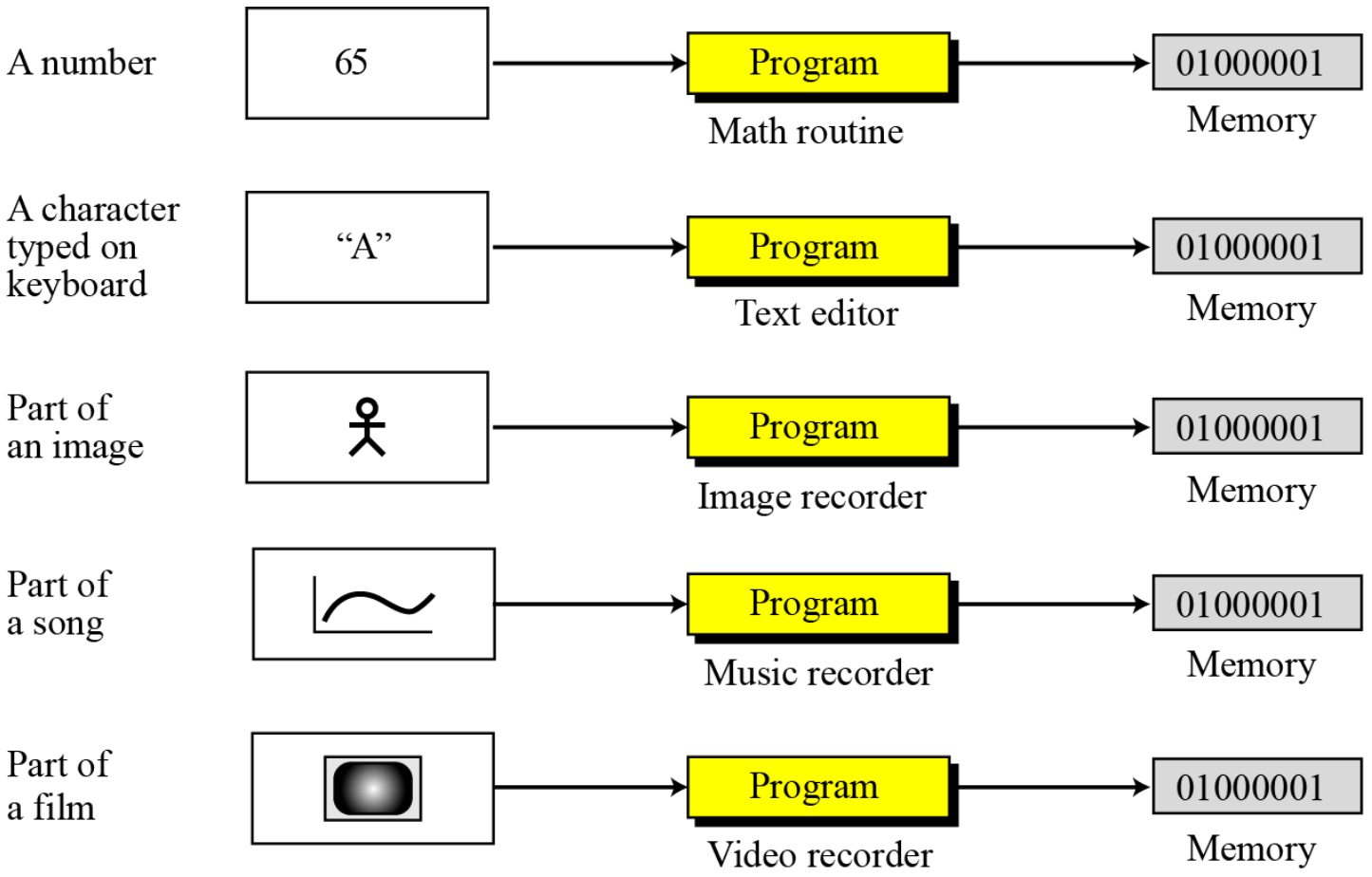


Figure **Storage of different data types**

ASCII Table

Dec	Hex	Oct	Char	Dec	Hex	Oct	Char	Dec	Hex	Oct	Char	Dec	Hex	Oct	Char
0	0	0		32	20	40	[space]	64	40	100	@	96	60	140	`
1	1	1		33	21	41	!	65	41	101	A	97	61	141	a
2	2	2		34	22	42	"	66	42	102	B	98	62	142	b
3	3	3		35	23	43	#	67	43	103	C	99	63	143	c
4	4	4		36	24	44	$	68	44	104	D	100	64	144	d
5	5	5		37	25	45	%	69	45	105	E	101	65	145	e
6	6	6		38	26	46	&	70	46	106	F	102	66	146	f
7	7	7		39	27	47	'	71	47	107	G	103	67	147	g
8	8	10		40	28	50	(	72	48	110	H	104	68	150	h
9	9	11		41	29	51	)	73	49	111	I	105	69	151	i
10	A	12		42	2A	52	*	74	4A	112	J	106	6A	152	j
11	B	13		43	2B	53	+	75	4B	113	K	107	6B	153	k
12	C	14		44	2C	54	,	76	4C	114	L	108	6C	154	l
13	D	15		45	2D	55	-	77	4D	115	M	109	6D	155	m
14	E	16		46	2E	56	.	78	4E	116	N	110	6E	156	n
15	F	17		47	2F	57	/	79	4F	117	O	111	6F	157	o
16	10	20		48	30	60	0	80	50	120	P	112	70	160	p
17	11	21		49	31	61	1	81	51	121	Q	113	71	161	q
18	12	22		50	32	62	2	82	52	122	R	114	72	162	r
19	13	23		51	33	63	3	83	53	123	S	115	73	163	s
20	14	24		52	34	64	4	84	54	124	T	116	74	164	t
21	15	25		53	35	65	5	85	55	125	U	117	75	165	u
22	16	26		54	36	66	6	86	56	126	V	118	76	166	v
23	17	27		55	37	67	7	87	57	127	W	119	77	167	w
24	18	30		56	38	70	8	88	58	130	X	120	78	170	x
25	19	31		57	39	71	9	89	59	131	Y	121	79	171	y
26	1A	32		58	3A	72	:	90	5A	132	Z	122	7A	172	z
27	1B	33		59	3B	73	;	91	5B	133	[	123	7B	173	{
28	1C	34		60	3C	74	<	92	5C	134	\	124	7C	174	|
29	1D	35		61	3D	75	=	93	5D	135	]	125	7D	175	}
30	1E	36		62	3E	76	>	94	5E	136	^	126	7E	176	~
31	1F	37		63	3F	77	?	95	5F	137	_	127	7F	177	

Amount of Data

ASCII: The American Standard Code for Information Interchange

Units

1 Byte (B) = 8 bits

1 KiB (Kibibyte) or 1 KB =1024 bytes = 2^{10}

1 kB (kilobyte) = 1000 bytes

1 MB (mebibyte) = 1048576 bytes = 2^{20}

1 mB (megabyte)=1000000 bytes

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	00	Null	32	20	Space	64	40	@	96	60	`
1	01	Start of heading	33	21	!	65	41	A	97	61	a
2	02	Start of text	34	22	"	66	42	B	98	62	b
3	03	End of text	35	23	#	67	43	C	99	63	c
4	04	End of transmit	36	24	$	68	44	D	100	64	d
5	05	Enquiry	37	25	%	69	45	E	101	65	e
6	06	Acknowledge	38	26	&	70	46	F	102	66	f
7	07	Audible bell	39	27	'	71	47	G	103	67	g
8	08	Backspace	40	28	(	72	48	H	104	68	h
9	09	Horizontal tab	41	29	)	73	49	I	105	69	i
10	0A	Line feed	42	2A	*	74	4A	J	106	6A	j
11	0B	Vertical tab	43	2B	+	75	4B	K	107	6B	k
12	0C	Form feed	44	2C	,	76	4C	L	108	6C	l
13	0D	Carriage return	45	2D	-	77	4D	M	109	6D	m
14	0E	Shift out	46	2E	.	78	4E	N	110	6E	n
15	0F	Shift in	47	2F	/	79	4F	O	111	6F	o
16	10	Data link escape	48	30	0	80	50	P	112	70	p
17	11	Device control 1	49	31	1	81	51	Q	113	71	q
18	12	Device control 2	50	32	2	82	52	R	114	72	r
19	13	Device control 3	51	33	3	83	53	S	115	73	s
20	14	Device control 4	52	34	4	84	54	T	116	74	t
21	15	Neg. acknowledge	53	35	5	85	55	U	117	75	u
22	16	Synchronous idle	54	36	6	86	56	V	118	76	v
23	17	End trans. block	55	37	7	87	57	W	119	77	w
24	18	Cancel	56	38	8	88	58	X	120	78	x
25	19	End of medium	57	39	9	89	59	Y	121	79	y
26	1A	Substitution	58	3A	:	90	5A	Z	122	7A	z
27	1B	Escape	59	3B	;	91	5B	[	123	7B	{
28	1C	File separator	60	3C	<	92	5C	\	124	7C	|
29	1D	Group separator	61	3D	=	93	5D	]	125	7D	}
30	1E	Record separator	62	3E	>	94	5E	^	126	7E	~
31	1F	Unit separator	63	3F	?	95	5F	_	127	7F	□

Information Object	How many bytes
A binary decision	1 bit
A single text character	1 byte
A typical text word	10 bytes
A typewritten page	2 kilobyte s (KB s)
A low-resolution photograph	100 kilobytes
A short novel	**1 megabyte (MB)**
The contents of a 3.5 inch floppy disk	1.44 megabytes
A high-resolution photograph	2 megabytes
The complete works of Shakespeare	**5 megabytes**
A minute of high-fidelity sound	10 megabytes
One meter (or close to a yard) of shelved books	100 megabytes
The contents of a CD-ROM	500 megabytes
A pickup truck filled with books	1 gigabyte GB)
The contents of a DVD	**17 gigabyte s**
A collection of the works of Beethoven	**20 gigabytes**
A library floor of academic journals	100 gigabytes
50,000 trees made into paper and printed	**1 terabyte (TB)**

An academic research library	2 terabytes
The print collections of the U.S. Library of Congress	10 terabytes
The National Climactic Data Center database	400 terabytes
Three years' of EOS data (2001)	1 petabyte (PB)
All U.S. academic research libraries	**2 petabytes**
All hard disk capacity developed in 1995	20 petabytes
All printed material in the world	**200 petabytes**
Total volume of information generated in 1999	**2 exabytes (EB s)**
All words ever spoken by human beings	**5 exabytes**

Tera 10^{12}
Peta 10^{15}
Exa 10^{18}

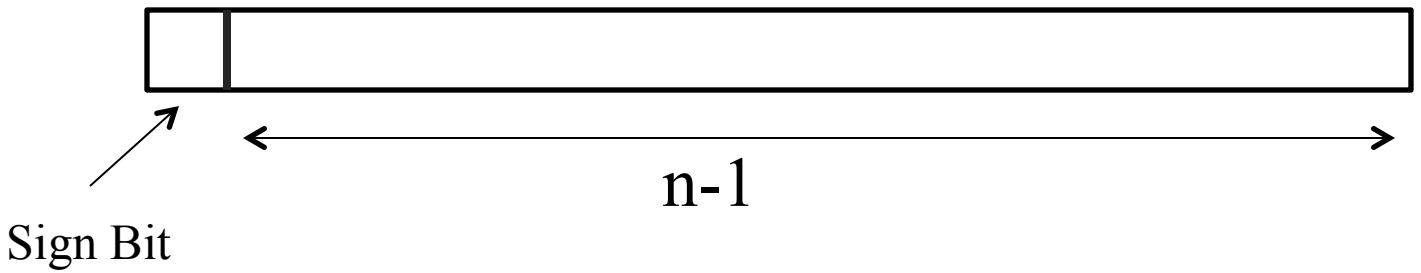
Representing a number on a computer

We can think of infinite number of numbers in our mind. If we are given a number, we can always come up with a number larger than that. If we are given two numbers we can always come up with a number in between those two.

But computer memory is a finite space. There is a limit on how large or how small a number or how close to each other two numbers could be handled on a computer.

This depends on how a computer stores numbers.

Representation of an Integer



bits	integer
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	−8
1001	−7
1010	−6
1011	−5
1100	−4
1101	−3
1110	−2
1111	−1

List of All Possible Integers Represented by 4 bits

$1111_2 = 15-8=7$

$1100_2 = 12-8 = 4$

$0100_2 = 4-8=-4$

$0000_2 = 0-8=-8$

$23.125_{10} = 10111.001_2$

$=1.0111001 \times 2^4$

S=0

$E=4+127=131_{10}=10000011$

M=0111001 ⋯sixteen zeros

0 10000011 0111001⋯sixteen zeros

E=8 bits

Max=255

Excess 127 represntation

255-127=128

$$_2^{128} = 10^{38}$$

M = 23 bits

000000···0001

1.00000···0001

1.00000···0010

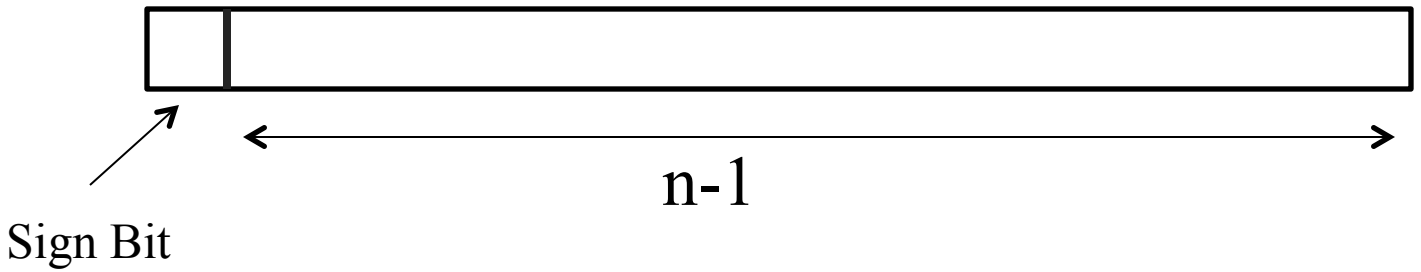
$$2^{-23} = 10^{-7}$$

A=1.0000000

B=1.0000001

A-B=0.0000001

Representation of an Integer



$$-2^{n-1} < i < 2^{n-1} - 1$$

Try finding 2*n+1 for n=0, 1, ..., N
Check where you stop getting sensible result.

Example 3.5

Store −28 in an 8-bit memory location using sign-and-magnitude representation.

Solution

The integer is changed to 7-bit binary. The leftmost bit is set to 1. The 8-bit number is stored.

Change 28 to 7-bit binary		0	0	1	1	1	0	0
Add the sign and store	1	0	0	1	1	1	0	0

Example 3.6

Retrieve the integer that is stored as 01001101 in sign-and-magnitude representation.

Solution

Since the leftmost bit is 0, the sign is positive. The rest of the bits (1001101) are changed to decimal as 77. After adding the sign, the integer is +77.

Example

Show the number

$$(10100100000000000000000000000000.00)_2$$

in floating-point representation.

Solution

We use the same idea, keeping only one digit to the left of the decimal point.

Actual number	→	+	$(10100100000000000000000000000000.00)_2$
Scientific notation	→	+	1.01001×2^{32}

Example

Show the number

$$-(0.00000000000000000000000101)_2$$

in floating-point representation.

Solution

We use the same idea, keeping only one digit to the left of the decimal point.

Actual number	→	–	$(0.00000000000000000000000101)_2$
Scientific notation	→	–	1.01×2^{-24}

Representation of a Real number (Single Precision)

	s	e	f
Bit position:	31	30 23	22 0

Number name	Values of s, e, and f	Value of single
Normal	$0 < e < 255$	$(-1)^s \times 2^{e-127} \times 1.f$
Subnormal	$e = 0,\ f \neq 0$	$(-1)^s \times 2^{-126} \times 0.f$
Signed Zero (± 0)	$e = 0,\ f = 0$	$(-1)^s \times 0.0$
$+\infty$	$s = 0,\ e = 255,\ f = 0$	+INF
$-\infty$	$s = 1,\ e = 255,\ f = 0$	−INF
Not a number	$s = u,\ e = 255,\ f \neq 0$	NaN

Representation of a Real number (Double Precision)

	s	e	f	f (cont)
Bit position:	63	62 52	51 32	31 0

Number name	Values of s, e, and f	Value of double
Normal	$0 \leq e \leq 2047$	$(-1)^s \times 2^{e-1023} \times 1.f$
Subnormal	$e = 0,\ f \neq 0$	$(-1)^s \times 2^{-1022} \times 0.f$
Signed zero	$e = 0,\ f = 0$	$(-1)^s \times 0.0$
$+\infty$	$s = 0,\ e = 2047,\ f = 0$	`+INF`
$-\infty$	$s = 1,\ e = 2047,\ f = 0$	`-INF`
Not a number	$s = u,\ e = 2047,\ f \neq 0$	`NaN`

Example 3.20

Show the number

$$(10100100000000000000000000000000.00)_2$$

in floating-point representation.

Solution

We use the same idea, keeping only one digit to the left of the decimal point.

Actual number	→	+	$(10100100000000000000000000000000.00)_2$
Scientific notation	→	+	1.01001×2^{32}

Example 3.21

Show the number

$$-(0.00000000000000000000000101)_2$$

in floating-point representation.

Solution

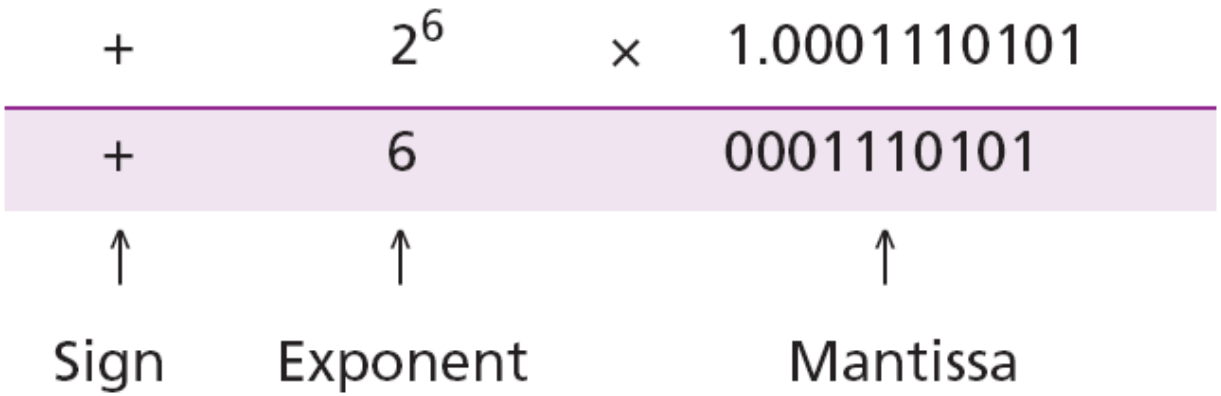
We use the same idea, keeping only one digit to the left of the decimal point.

Actual number	→	–	$(0.00000000000000000000000101)_2$
Scientific notation	→	–	1.01×2^{-24}

Normalization

To make the fixed part of the representation uniform, both the scientific method (for the decimal system) and the floating-point method (for the binary system) use only one non-zero digit on the left of the decimal point. This is called **normalization**. In the decimal system this digit can be 1 to 9, while in the binary system it can only be 1. In the following, *d* is a non-zero digit, *x* is a digit, and *y* is either 0 or 1.

Decimal	→	±	d.xxxxxxxxxxxxxx	Note: *d* is 1 to 9 and each *x* is 0 to 9
Binary	→	±	1.yyyyyyyyyyyyyy	Note: each *y* is 0 or 1



Note that the point and the bit 1 to the left of the fixed-point section are not stored—they are implicit.

The mantissa is a fractional part that, together with the sign, is treated like an integer stored in sign-and-magnitude representation.

Example

Show the Excess_127 (single precision) representation of the decimal number 5.75.

Solution

a. The sign is positive, so S = 0.
b. Decimal to binary transformation: $5.75 = (101.11)_2$.
c. Normalization: $(101.11)_2 = (1.0111)_2 \times 2^2$.
d. $E = 2 + 127 = 129 = (10000001)_2$, M = 0111. We need to add nineteen zeros at the right of M to make it 23 bits.
e. The presentation is shown below:

0	10000001	10110000000000000000
S	E	M

The number is stored in the computer as

01000000110110000000000000000000

Example

Show the Excess_127 (single precision) representation of the decimal number –161.875.

Solution

a. The sign is negative, so S = 1.
b. Decimal to binary transformation: $161.875 = (10100001.111)_2$.
c. Normalization: $(10100001.111)_2 = (1.0100001111)_2 \times 2^7$.
d. $E = 7 + 127 = 134 = (10000110)_2$ and $M = (0100001111)_2$.
e. Representation:

1	10000110	01000011110000000000000
S	E	M

The number is stored in the computer as

11000011010000111100000000000000

Example

Show the Excess_127 (single precision) representation of the decimal number –0.0234375.

Solution

a. S = 1 (the number is negative).
b. Decimal to binary transformation: $0.0234375 = (0.0000011)_2$.
c. Normalization: $(0.0000011)_2 = (1.1)_2 \times 2^{-6}$.
d. $E = -6 + 127 = 121 = (01111001)_2$ and $M = (1)_2$.
e. Representation:

1	01111001	10000000000000000000000
S	E	M

The number is stored in the computer as

10111100110000000000000000000000

Example

The bit pattern $(11001010000000000111000100001111)_2$ is stored in Excess_127 format. Show the value in decimal.

Solution

a. The first bit represents S, the next eight bits, E and the remaining 23 bits, M.

S	E	M
1	10010100	00000000111000100001111

b. The sign is negative.
c. The shifter = E − 127 = 148 − 127 = 21.
d. This gives us $(1.00000000111000100001111)_2 \times 2^{21}$.
e. The binary number is $(100000000111000100011.11)_2$.
f. The absolute value is 2,104,378.75.
g. The number is **−2,104,378.75**.

	single precision	double precision
largest number	$\simeq 10^{38}$	$\simeq 10^{308}$
smallest number	$\simeq 10^{-38}$	$\simeq 10^{-308}$
precision	$\simeq 10^{-7}$	$\simeq 10^{-16}$

Overflow and Underflow

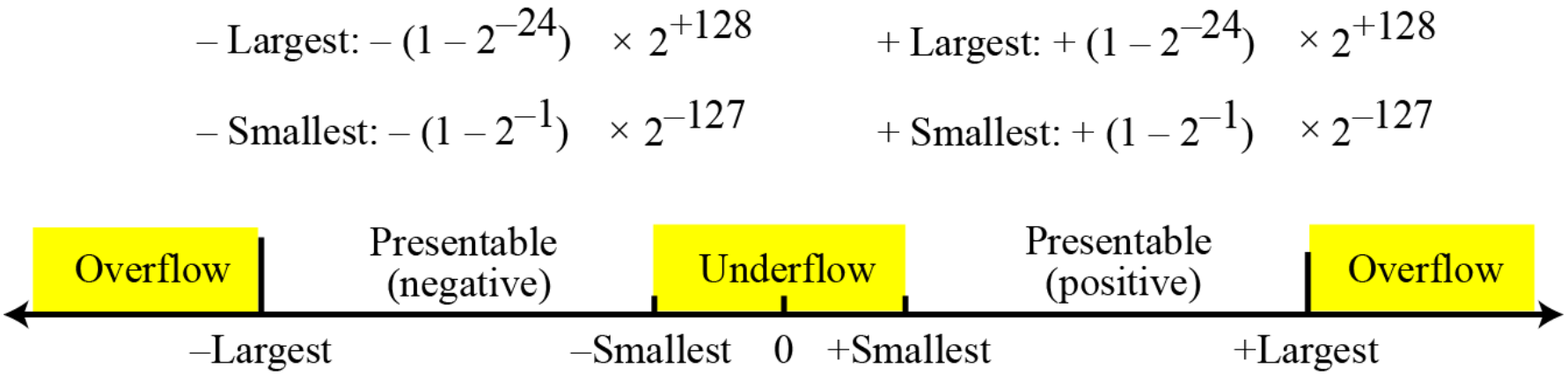


Figure 3.12 **Overflow and underflow in floating-point representation of reals**

Zero: A real number with an integral part and the fractional part set to zero, that is, 0.0, cannot be stored using the steps discussed above. To handle this special case, it is agreed that in this case the sign, exponent and the mantissa are set to 0s.

Infinity: Every bit of the exponent is 1 and mantissa is all 0.

NaN: Every bit of the exponent is 1 and at least one mantissa bit is 1

Numerical Precision

- The difference between 1 and the closest number to 1 that is distinct from 1.
- With a 23-bit mantissa the precision is $2^{-23} \simeq 1.2 \times 10^{-7}$.
- The precision is also a typical value of the relative error of a number that is not represented exactly by 23 bits of mantissa. The relative error is defined by

Relative error = (estimated value − exact value)/exact value

In base-10 the number 1/2 has a terminating expansion (0.5) while the number 1/3 does not (0.333...). In base-2 only rationals with denominators that are powers of 2 (such as 1/2 or 3/16) are terminating. Any rational with a denominator that has a prime factor other than 2 will have an infinite binary expansion.

This means that numbers which appear to be short and exact when written in decimal format may need to be approximated when converted to binary floating-point. For example, the decimal number 0.1 is not representable in binary floating-point of any finite precision; the exact binary representation would have a "1100" sequence continuing endlessly:

e = −4; f = 11001100110011001100110011001100110011...
where, as previously, f is the significand and e is the exponent.
When rounded to 24 bits this becomes
e = −4; f = 11001100110011001100110,
which is actually 0.100000001490116119384765625 in decimal.

---Wikipedia