2.1 Number Systems

Decimal	Base = 10	Digit Set = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
Binary	Base = 2	Digit Set = {0, 1}
Octal	Base = 8 = 2^3	Digit Set = {0, 1, 2, 3, 4, 5, 6, 7}
Hexadecimal	Base = 16 = 2^4	Digit Set = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F}

Decimal	Binary (base 2)	Octal (base 8)	Hexadecimal (base 16)
0	0000	00	0
1	0001	01	1
2	0010	02	2
3	0011	03	3
4	0100	04	4
5	0101	05	5
6	0110	06	6
7	0111	07	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F
16	10000	20	10

Note that conversions from hexadecimal to binary can be done one digit at a time, thus DE = 11011110, as D = 1101 and E = 1110. We shall normally denote this as DE = 1101 1110 with a space to facilitate reading the binary.

Conversion from binary to hexadecimal is also quite easy. Group the bits four at a time and convert each set of four. Thus 10111101, written 1011 1101 for clarity is BD because 1011 = B and 1101 = D.

Character Codes: ASCII

The figure below shows the ASCII code. Only the first 128 characters (Codes 00 – 7F in hexadecimal) are standard. There are several interesting facts.

Last Digit \ First Digit	0	1	2	3	4	5	6	7
0	NUL	DLE	SP	0	@	P	`	p
1	SOH	DC1	!	1	A	Q	a	q
2	STX	DC2	“	2	B	R	b	r
3	ETX	DC3	#	3	C	S	c	s
4	EOT	DC4	$	4	D	T	d	t
5	ENQ	NAK	%	5	E	U	e	u
6	ACK	SYN	&	6	F	V	f	v
7	BEL	ETB	`	7	G	W	g	w
8	BS	CAN	(	8	H	X	h	x
9	HT	EM	)	9	I	Y	i	y
A	LF	SUB	*	:	J	Z	j	z
B	VT	ESC	+	;	K	[	k	{
C	FF	FS	‘	<	L	\	l	|
D	CR	GS	-	=	M	]	m	}
E	SO	RS	.	>	N	^	n	~
F	SI	US	/	?	O	_	o	DEL

As ASCII is designed to be an 8–bit code, several manufacturers have defined extended code sets, which make use of the codes 0x80 through 0xFF (128 through 255). One of the more popular was defined by IBM. None are standard; most books ignore them. So do we.

Let X be the ASCII code for a digit. Then X – '0' = X – 30 is the value of the digit.
For example ASCII('7') = 37, with value 37 – 30 = 7.

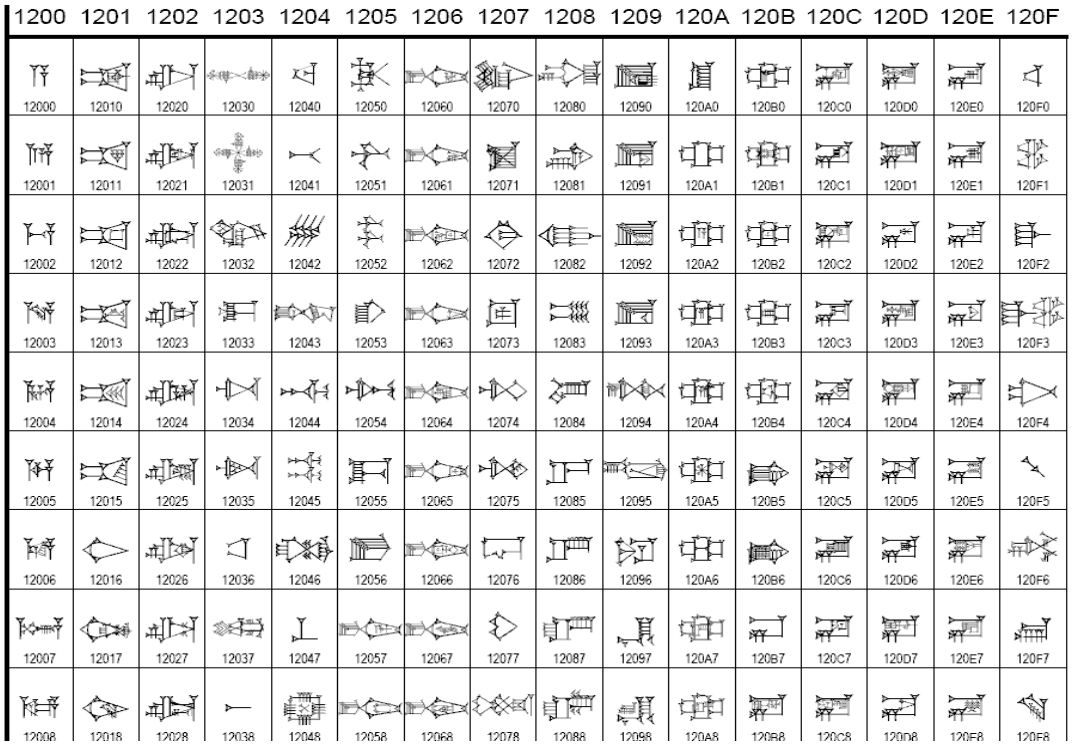
Let X be an ASCII code. If ASCII('A') ≤ X ≤ ASCII('Z') then X is an upper case letter.
If ASCII('a') ≤ X ≤ ASCII('z') then X is an lower case letter.
If ASCII('0') ≤ X ≤ ASCII('9') then X is a decimal digit.

Let X be an upper-case letter. Then ASCII (lower_case(X)) = ASCII (X) + 32
Let X be a lower case letter. Then ASCII (UPPER_CASE(X)) = ASCII (X) – 32.
The expressions are ASCII (X) + 20 and ASCII (X) – 20 in hexadecimal.

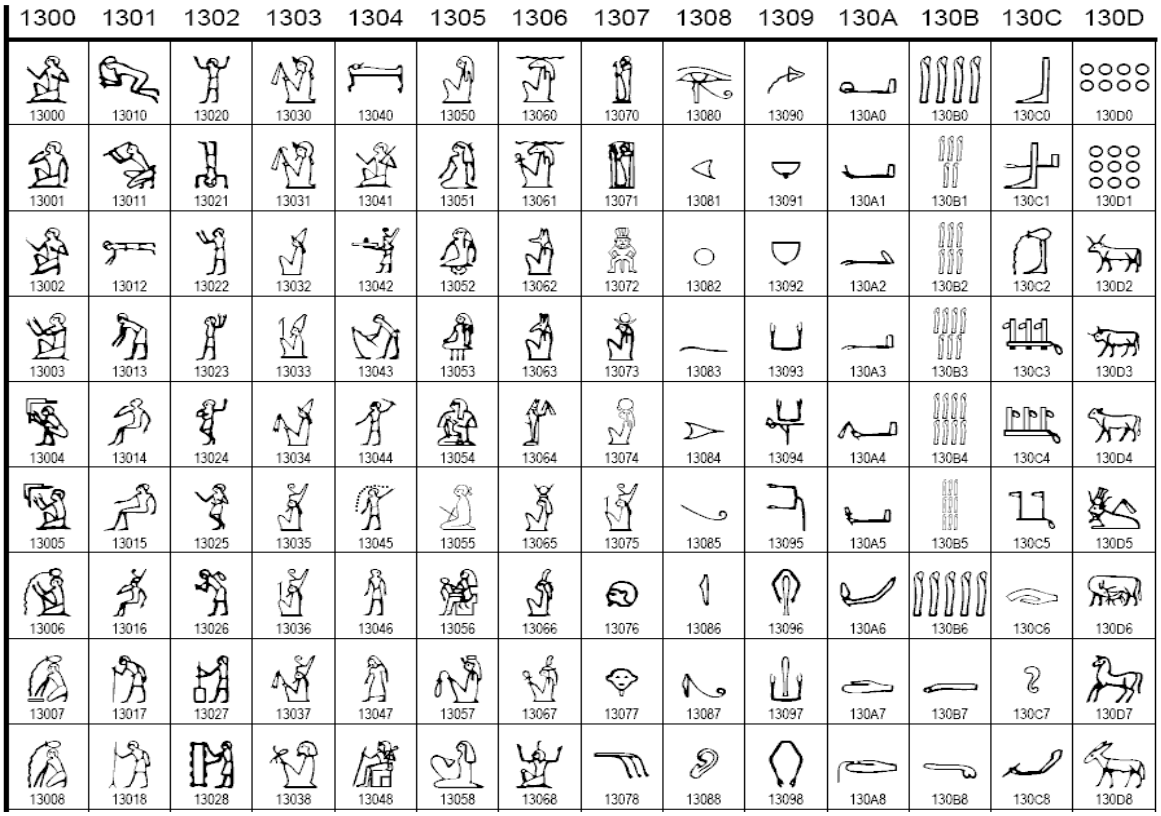
Here is a bit of the Greek alphabe.

0x0391¤	913¤	GREEK·CAPITAL·LETTER·ALPHA¤	Α¤
0x0392¤	914¤	GREEK·CAPITAL·LETTER·BETA¤	Β¤
0x0393¤	915¤	GREEK·CAPITAL·LETTER·GAMMA¤	Γ¤
0x0394¤	916¤	GREEK·CAPITAL·LETTER·DELTA¤	Δ¤
0x0395¤	917¤	GREEK·CAPITAL·LETTER·EPSILON¤	Ε¤
0x0396¤	918¤	GREEK·CAPITAL·LETTER·ZETA¤	Ζ¤
0x0397¤	919¤	GREEK·CAPITAL·LETTER·ETA¤	Η¤
0x0398¤	920¤	GREEK·CAPITAL·LETTER·THETA¤	Θ¤
0x0399¤	921¤	GREEK·CAPITAL·LETTER·IOTA¤	Ι¤
0x039A¤	922¤	GREEK·CAPITAL·LETTER·KAPPA¤	Κ¤
0x039B¤	923¤	GREEK·CAPITAL·LETTER·LAMDA¤	Λ¤
0x039C¤	924¤	GREEK·CAPITAL·LETTER·MU¤	Μ¤
0x039D¤	925¤	GREEK·CAPITAL·LETTER·NU¤	Ν¤
0x039E¤	926¤	GREEK·CAPITAL·LETTER·XI¤	Ξ¤
0x03A0¤	928¤	GREEK·CAPITAL·LETTER·PI¤	Π¤
0x03A1¤	929¤	GREEK·CAPITAL·LETTER·RHO¤	Ρ¤
0x03A3¤	931¤	GREEK·CAPITAL·LETTER·SIGMA¤	Σ¤
0x03A4¤	932¤	GREEK·CAPITAL·LETTER·TAU¤	Τ¤
0x03A5¤	933¤	GREEK·CAPITAL·LETTER·UPSILON¤	Υ¤
0x03A6¤	934¤	GREEK·CAPITAL·LETTER·PHI¤	Φ¤
0x03A7¤	935¤	GREEK·CAPITAL·LETTER·CHI¤	Χ¤
0x03A8¤	936¤	GREEK·CAPITAL·LETTER·PSI¤	Ψ¤
0x03A9¤	937¤	GREEK·CAPITAL·LETTER·OMEGA¤	Ω¤

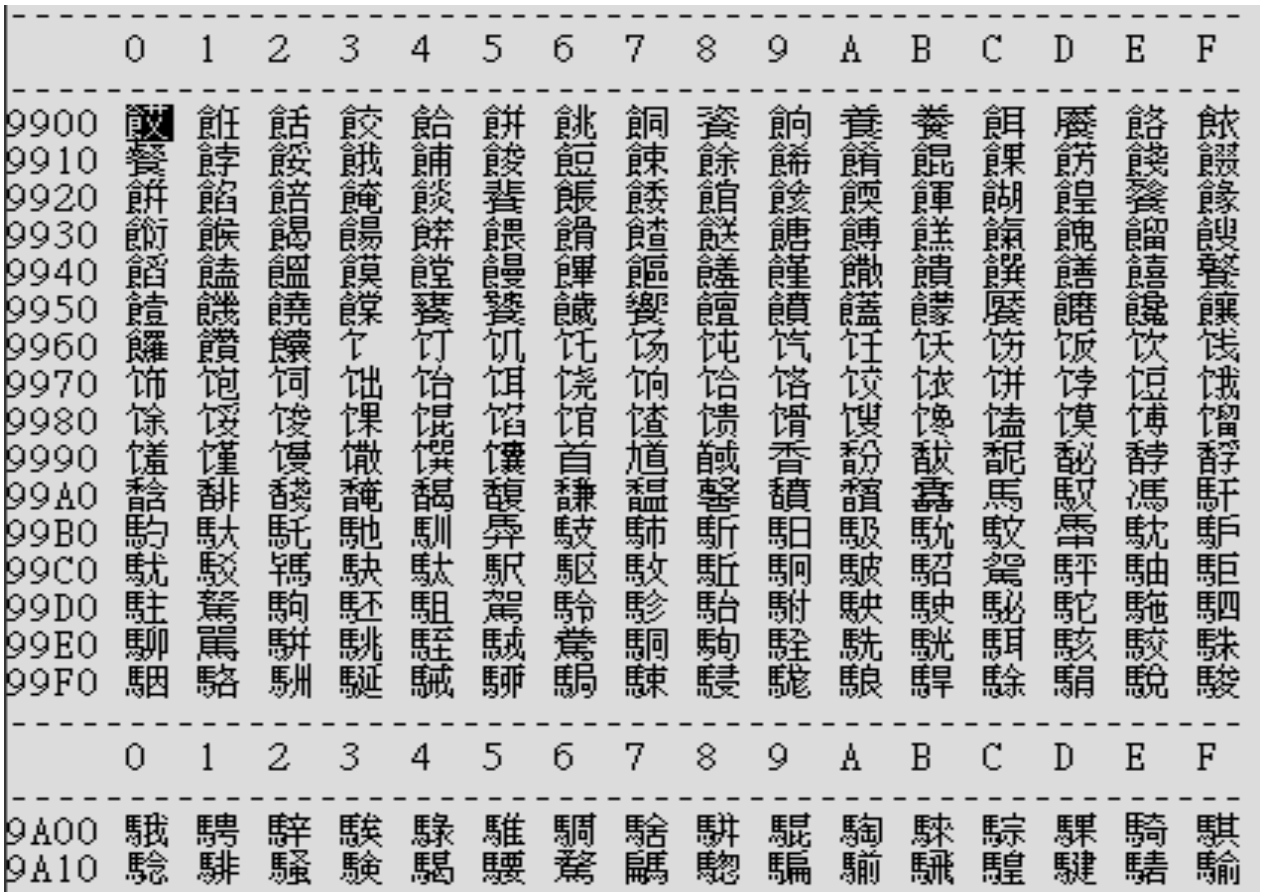
For those with more esoteric tastes, here is a small sample of Cuneiform in 32–bit Unicode.



Now we see some Egyptian hieroglyphics, also with the 32–bit Unicode encoding.



.Unicode supports Chinese and Japanese symbols.



Memory as a Linear Array

Consider a byte-addressable memory with N bytes of memory. As stated above, such a memory can be considered to be the logical equivalent of a C++ array, declared as

```
byte memory [N] ;  // Address ranges from 0 through (N - 1)
```

The computer on which these notes were written has 384 MB of main memory, now only an average size but once unimaginably large. 384 MB = 384•2^{20} bytes and the memory is byte-addressable, so N = 384•1048576 = 402,653,184. Quite often the memory size will either be a power of two or the sum of two powers of two; 384 MB = (256 + 128)•2^{20} = 2^{28} + 2^{27}.

Early versions of the IBM S/360 provided addressability of up to 2^{24} = 16,777,216 bytes of memory, or 4,194,304 32–bit words. All addresses in the S/360 series are byte addresses.

The term "**random access**" used when discussing computer memory implies that memory can be accessed at random with no performance penalty. While this may not be exactly true in these days of virtual memory, the key idea is simple – that the time to access an item in memory does not depend on the address given. In this regard, it is similar to an array in which the time to access an entry does not depend on the index. A magnetic tape is a typical **sequential access device** – in order to get to an entry one must read over all pervious entries.

There are two major types of random-access computer memory. These are:

RAM Read-Write Memory
ROM Read-Only Memory

The usage of the term "RAM" for the type of random access memory that might well be called "RWM" has a long history and will be continued in this course. The basic reason is probably that the terms "RAM" and "ROM" can easily be pronounced; try pronouncing "RWM". Keep in mind that both RAM and ROM are random access memory.

Of course, there is no such thing as a pure Read–Only memory; at some time it must be possible to put data in the memory by writing to it, otherwise there will be no data in the memory to be read. The term "Read-Only" usually refers to the method for access by the CPU. All variants of ROM share the feature that their contents cannot be changed by normal CPU write operations. All variants of RAM (really Read-Write Memory) share the feature that their contents can be changed by normal CPU write operations. Some forms of ROM have their contents set at time of manufacture, other types called **PROM** (Programmable ROM), can have contents changed by special devices called PROM Programmers.

The Idea of Address Space

We now must distinguish between the idea of **address space** and **physical memory**. The address space defines the range of addresses (indices into the memory array) that can be generated. The size of the physical memory is usually somewhat smaller, this may be by design (see the discussion of memory-mapped I/O below) or just by accident.

The memory address is specified by a binary number placed in the Memory Address Register (MAR). The number of bits in the MAR determines the range of addresses that can be generated. N address lines can be used to specify 2^N distinct addresses, numbered 0 through $2^N - 1$. This is called the **address space** of the computer. The early IBM S/360 had a 24–bit MAR, corresponding to an address space of 0 through $2^{24} - 1$, or 0 through 4,194,303.

For example, we show some MAR sizes.

Computer	MAR bits	Address Range
PDP-11/20	16	0 to 65,535
Intel 8086	20	0 to 1,048,575
IBM 360	24	0 to 4,194,303
S/370–XA	31	0 to 2,147,483,647
Pentium	32	0 to 4,294,967,295
z/Series	64	A very big number.

The PDP-11/20 was an elegant small machine made by the now defunct Digital Equipment Corporation. As soon as it was built, people realized that its address range was too small.

In general, the address space is much larger than the physical memory available. For example, my personal computer has an address space of 2^{32} (as do all Pentiums), but only 384MB = $2^{28} + 2^{27}$ bytes. Until recently the 32–bit address space would have been much larger than any possible amount of physical memory. At present one can go to a number of companies and order a computer with a fully populated address space; i.e., 4 GB of physical memory. Most high-end personal computers are shipped with 1GB of memory.

Word Addresses in a Byte-Addressable Machine

Most computers today, including all of those in the IBM S/360 series, have memories that are byte-addressable**; thus each byte in the memory has a unique address that can be used to address it. Under this addressing scheme, a word corresponds to a number of addresses.**

A 16-bit word at address Z contains bytes at addresses Z and Z + 1.
A 32-bit word at address Z contains bytes at addresses Z, Z + 1, Z + 2, and Z + 3.

In many computers with byte addressing, there are constraints on word addresses.

A 16-bit word must have an even address
A 32-bit word must have an address that is a multiple of 4.

This is true of the IBM S/360 series in which 16–bit words are called "halfwords", 32–bit words are called "words", and 64–bit words are called "double words". A halfword must have an even address, a word must have an address that is a multiple of 4 and a double word (64 bits) an address that is a multiple of 8.

Even in computers that do not enforce this requirement, it is a good idea to observe these word boundaries. Most compilers will do so automatically.

Suppose a byte-addressable computer with a 24–bit address space. The highest byte address is $2^{24} - 1$. From this fact and the address allocation to multi-byte words, we conclude

the highest address for a 16-bit word is $(2^{24} - 2)$, and

the highest address for a 32-bit word is $(2^{24} - 4)$, because the 32–bit word addressed at $(2^{24} - 4)$ comprises bytes at addresses $(2^{24} - 4)$, $(2^{24} - 3)$, $(2^{24} - 2)$, and $(2^{24} - 1)$.

Byte Addressing vs. Word Addressing

We have noted above that N address lines can be used to specify 2^N distinct addresses, numbered 0 through $2^N - 1$. We now ask about the size of the addressable items. As a simple example, consider a computer with a 24–bit address space. The machine would have 16,777,216 (16M) addressable entities. In a byte–addressable machine, such as the IBM S/360, this would correspond to:

16 M Bytes	16,777,216	bytes, or
8 M halfwords	8,388,608	16-bit halfwords, or
4 M words	4,194,304	32–bit fullwords.

The advantages of byte-addressability are clear when we consider applications that process data one byte at a time. Access of a single byte in a byte-addressable system requires only the issuing of a single address. In a 16–bit word addressable system, it is necessary first to compute the address of the word containing the byte, fetch that word, and then extract the byte from the two–byte word. Although the processes for byte extraction are well understood, they are less efficient than directly accessing the byte. For this reason, many modern machines are byte addressable.

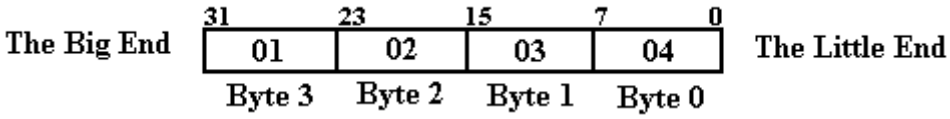
Big–Endian and Little–Endian

The reference here is to a story in *Gulliver's Travels* written by Jonathan Swift in which two groups went to war over which end of a boiled egg should be broken – the big end or the little end. The student should be aware that Swift did not write pretty stories for children but focused on biting satire; his work *A Modest Proposal* is an excellent example.

Consider the 32-bit number represented by the eight–digit hexadecimal number 0x01020304, stored at location Z in memory. In all byte–addressable memory locations, this number will be stored in the four consecutive addresses Z, (Z + 1), (Z + 2), and (Z + 3). The difference between big-endian and little-endian addresses is where each of the four bytes is stored.

In our example	0x01 represents bits	31	– 24,	
	0x02 represents bits	23	– 16,	
	0x03 represents bits	15	– 8,	and
	0x04 represents bits	7	– 0.	

As a 32-bit signed integer, the number represents $01 \bullet (256)^3 + 02 \bullet (256)^2 + 03 \bullet (256) + 04$ or $0 \bullet 16^7 + 1 \bullet 16^6 + 0 \bullet 16^5 + 2 \bullet 16^4 + 0 \bullet 16^3 + 3 \bullet 16^2 + 0 \bullet 16^1 + 4 \bullet 16^0$, which evaluates to $1 \bullet 16777216 + 2 \bullet 65536 + 3 \bullet 256 + 4 \bullet 1 = 16777216 + 131072 + 768 + 4 = 16909060$. Note that the number can be viewed as having a "big end" and a "little end", as in the next figure.

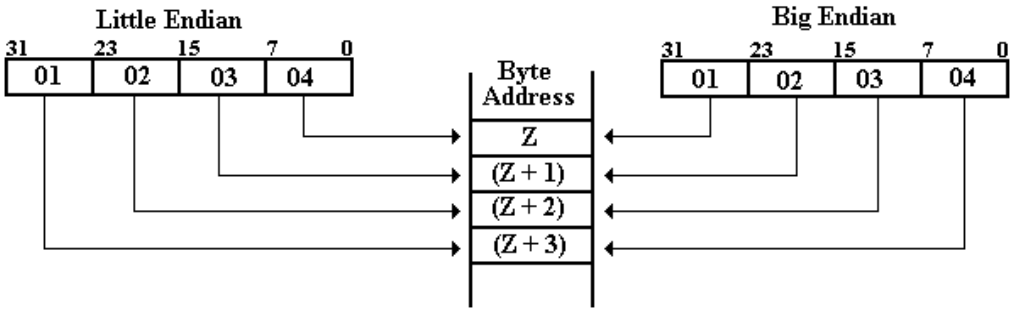


The "big end" contains the most significant digits of the number and the "little end" contains the least significant digits of the number. We now consider how these bytes are stored in a byte-addressable memory. Recall that each byte, comprising two hexadecimal digits, has a unique address in a byte-addressable memory, and that a 32-bit (four-byte) entry at address Z occupies the bytes at addresses Z, (Z + 1), (Z + 2), and (Z + 3).

The hexadecimal values stored in these four byte addresses are shown below.

Address	Big-Endian	Little-Endian
Z	01	04
Z + 1	02	03
Z + 2	03	02
Z + 3	04	01

The figure below shows a graphical way to view these two options for ordering the bytes copied from a register into memory. We suppose a 32-bit register with bits numbered from 31 through 0. Which end is placed first in the memory – at address Z? For big-endian, the "big end" or most significant byte is first written. For little-endian, the "little end" or least significant byte is written first.



Just to be complete, consider the 16-bit number represented by the four hex digits 0A0B, with decimal value 10•256 + 11 = 2571. Suppose that the 16-bit word is at location W; i.e., its bytes are at locations W and (W + 1). The most significant byte is 0x0A and the least significant byte is 0x0B. The values in the two addresses are shown below.

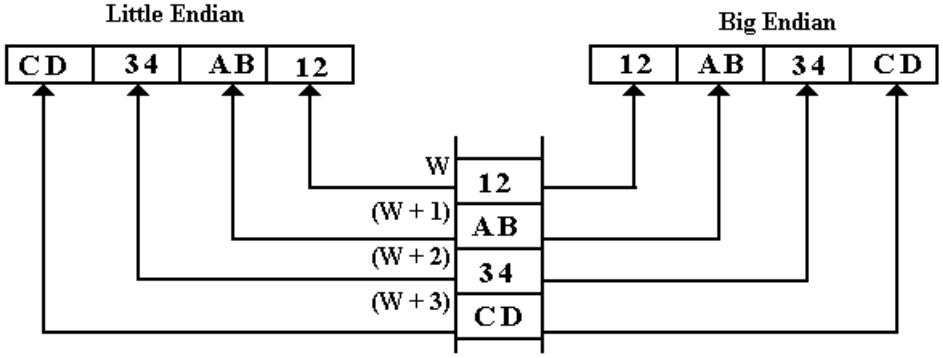
Address	Big-Endian	Little-Endian
W	0A	0B
W + 1	0B	0A

Here we should note that the IBM S/360 is a "Big Endean" machine.

As an example of a typical problem, let's examine the following memory map, with byte addresses centered on address W. Note the contents are listed as hexadecimal numbers. Each byte is an 8-bit entry, so that it can store unsigned numbers between 0 and 255, inclusive. These are written in hexadecimal as 0x00 through 0xFF inclusive.

Address	(W – 2)	(W – 1)	W	(W + 1)	(W + 2)	(W + 3)	(W + 4)
Contents	0B	AD	12	AB	34	CD	EF

We first ask what 32-bit integers are stored at address W. Recalling that the value of the number stored depends on whether the format is big-endian or little-endian, we draw the memory map in a form that is more useful.



This figure should illustrate one obvious point: the entries (W – 2), (W – 1), and (W + 4) are "red herrings", data that have nothing to do with the problem at hand. We now consider the conversion of the number in big-endian format. As a decimal number, this evaluates to

$1\bullet16^7 + 2\bullet16^6 + A\bullet16^5 + B\bullet16^4 + 3\bullet16^3 + 4\bullet16^2 + C\bullet16^1 + D$, or
$1\bullet16^7 + 2\bullet16^6 + 10\bullet16^5 + 11\bullet16^4 + 3\bullet16^3 + 4\bullet16^2 + 12\bullet16^1 + 13$, or
$1\bullet268435456 + 2\bullet16777216 + 10\bullet1048576 + 11\bullet65536 + 3\bullet4096 + 4\bullet256 + 12\bullet16 + 13$, or
$268435456 + 33554432 + 10485760 + 720896 + 12288 + 1024 + 192 + 13$, or
313210061.

The evaluation of the number as a little–endian quantity is complicated by the fact that the number is negative. In order to maintain continuity, we convert to binary (recalling that A = 1010, B = 1011, C = 1100, and D = 1101) and take the two's-complement.

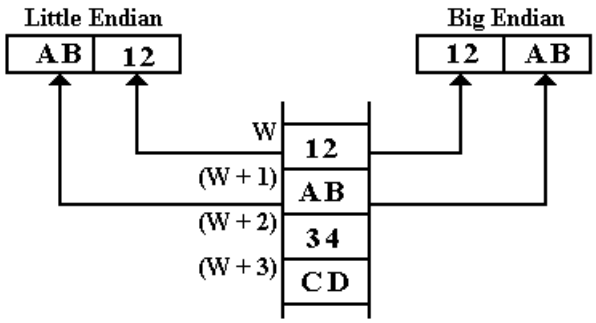
Hexadecimal	CD	34	AB	12
Binary	11001101	00110100	10101011	00010010
One's Comp	00110010	11001011	01010100	11101101
Two's Comp	00110010	11001011	01010100	11101110
Hexadecimal	32	AB	54	EE

Converting this to decimal, we have the following

$3\bullet16^7 + 2\bullet16^6 + A\bullet16^5 + B\bullet16^4 + 5\bullet16^3 + 4\bullet16^2 + E\bullet16^1 + E$, or
$3\bullet16^7 + 2\bullet16^6 + 10\bullet16^5 + 11\bullet16^4 + 5\bullet16^3 + 4\bullet16^2 + 14\bullet16^1 + 14$, or
$3\bullet268435456 + 2\bullet16777216 + 10\bullet1048576 + 11\bullet65536 + 5\bullet4096 + 4\bullet256 + 14\bullet16 + 14$, or
$805306368 + 33554432 + 10485760 + 720896 + 20480 + 1024 + 224 + 14$, or
850089198

The number represented in little-endian form is – 850,089,198.

We now consider the next question: what 16-bit integer is stored at address W? We begin our answer by producing the drawing for the 16-bit big-endian and little-endian numbers.



The evaluation of the number as a 16-bit big-endian number is again the simpler choice. The decimal value is $1\bullet16^3 + 2\bullet16^2 + 10\bullet16 + 11 = 4096 + 512 + 160 + 11 = 4779$.

The evaluation of the number as a little–endian quantity is complicated by the fact that the number is negative. We again take the two's-complement to convert this to positive.

Hexadecimal	AB	12
Binary	10101011	00010010
One's Comp	01010100	11101101
Two's Comp	01010100	11101110
Hexadecimal	54	EE

The magnitude of this number is $5\bullet16^3 + 4\bullet16^2 + 14\bullet16 + 14 = 20480 + 1024 + 224 + 14$, or 21742. The original number is thus the negative number – 21742.

One might ask similar questions about real numbers and strings of characters stored at specific locations. For a string constant, the value depends on the format used to store strings and might include such things as /0 termination for C and C++ strings. A typical question on real number storage would be to consider the following:

A real number is stored in byte-addressable memory in little-endian form.
The real number is stored in IEEE-754 single-precision format.

Address	W	(W + 1)	(W + 2)	(W + 3)
Contents	00	00	E8	42

The trick here is to notice that the number written in its proper form, with the "big end" on the left hand side is 0x42E80000, which we have seen represents the number 116.00. Were the number stored in big-endian form, it would be a denormalized number, about $8.32 \bullet 10^{-41}$.

There seems to be no advantage of one system over the other. Big-endian seems more natural to most people and facilitates reading hex dumps (listings of a sequence of memory locations), although a good debugger will remove that burden from all but the unlucky.

Big-endian computers include the IBM 360 series, Motorola 68xxx, and SPARC by Sun.

Little-endian computers include the Intel Pentium and related computers.

Solved Problems

As is obvious from the variety of fonts, these problems have been assembled from a variety of sources. All of these problems have been used in a teaching context.

1. What range of integers can be stored in an 16–bit word if

   a) the number is stored as an unsigned integer?

   b) the number is stored in two's–complement form?

Answer: a) 0 through 65,535 inclusive, or 0 through $2^{16} - 1$.
b) –32768 through 32767 inclusive, or $-(2^{15})$ through $(2^{15}) - 1$

2 You are given the 16–bit value, represented as four hexadecimal digits, and stored in two bytes. The value is 0x812D.

   a) What is the decimal value stored here, if interpreted as a packed decimal number?

   b) What is the decimal value stored, if interpreted as a 16–bit two's–complement integer?

   c) What is the decimal value stored here, if interpreted as a 16–bit unsigned integer?

ANSWER: The answers are found in the lectures for January 13 and January 20.

   a) For a packed decimal number, the absolute value is 812 and the value is negative. The answer is **–812**.

   b) To render this as a two's–complement integer, one first has to convert to binary.
   Hexadecimal **`812D`** converts to **`1000 0001 0010 1101`**. This is negative.
   Take the one's complement to get **`0111 1110 1101 0010`**.
   Add 1 to get the positive value **`0111 1110 1101 0011`**.
   In hexadecimal, this is **`7ED3`**, which converts to $7 \bullet 16^3 + 14 \bullet 16^2 + 13 \bullet 16 + 3$,
   or $7 \bullet 4096 + 14 \bullet 256 + 13 \bullet 16 + 3 = 28672 + 3584 + 208 + 3 = 32{,}467$
   The answer is **–32,467**.

   c) As an unsigned binary number the value is obtained by direct conversion from the hexadecimal value. The value is $8 \bullet 16^3 + 1 \bullet 16^2 + 2 \bullet 16 + 13$,
   or $8 \bullet 4096 + 1 \bullet 256 + 2 \bullet 16 + 13 = 32768 + 256 + 32 + 13 =$ **33069**.

3. Give the 8–bit two's complement representation of the number – 98.

Answer: 98 = 96 + 2 = 64 + 32 + 2, so its binary representation is 0110 0010.

8–bit representation of + 98	**0110 0010**	
One's complement	**1001 1101**	
Add 1 to get	**1001 1110**	**9E.**

4. Give the 16–bit two's complement representation of the number – 98.

Answer:	**The 8–bit representation of – 98 is**	**1001 1110**
	Sign extend to 16 bits	**1111 1111 1001 1110**

5 Convert the following decimal numbers to binary.

a) 37.375 b) 93.40625

ANSWER: Recall that the integer part and fractional part are converted separately.

a) 37.375

37 / 2	= 18	rem 1	
18 / 2	= 9	rem 0	
9 / 2	= 4	rem 1	
4 / 2	= 2	rem 0	
2 / 2	= 1	rem 0	
1 / 2	= 0	rem 1	Answer: 100101.

0.375 • 2	= 0.75		
0.75 • 2	= 1.50		
0.50 • 2	= 1.00		
0.00 • 2	= 0.00	Answer 0.011	**100101.011**

b) 93.40625

93 / 2	= 46	rem 1	
46 / 2	= 23	rem 0	
23 / 2	= 11	rem 1	
11 / 2	= 5	rem 1	
5 / 2	= 2	rem 1	
2 / 2	= 1	rem 0	
1 / 2	= 0	rem 1	Answer: 1011101

0.40625 • 2	= 0.8125		
0.8125 • 2	= 1.6250		
0.625 • 2	= 1.2500		
0.25 • 2	= 0.5000		
0.50 • 2	= 1.0000		
0.00 • 2	= 0.0000	Answer: 0.01101	**1011101.01101**

6 Convert the following hexadecimal number to decimal numbers.
The numbers are unsigned. Use as many digits as necessary
a) 0x022 b) 0x0BAD c) 0x0EF

ANSWER: A = 10, B = 11, C = 12, D = 13, E = 14, F = 15
$16^0 = 1$, $16^1 = 16$, $16^2 = 256$, $16^3 = 4096$, $16^4 = 65536$

a) 0x022 = 2•16 + 2 = 32 + 2 = 34 **34**

b) 0x0BAD = $11 \bullet 16^2 + 10 \bullet 16 + 13$ = 11•256 + 10•16 + 13
= 2816 + 160 + 13 = 2989 **2989**

c) 0x0EF = 14•16 + 15 = 224 + 15 = 239 **239**

7 Show the IEEE–754 single precision representation of the following real numbers. Show all eight hexadecimal digits associated with each representation.
a) 0.0 b) – 1.0 c) 7.625 d) – 8.75

ANSWER:

a) 0.0 = 0x0000 0000 **0x0000 0000**

b) – 1.0 this is negative, so the sign bit is S = 1
1.0 = $1.0 \bullet 2^0$
1.M = 1.0 so M = 0000
P = 0 so P + 127 = 127 = $0111\ 1111_2$

Concatenate S | (P + 127) | M	1 0111 1111 0000
Group by 4's from the left	1011 1111 1000 0
Pad out the last to four bits	1011 1111 1000 0000
Convert to hex digits	BF80
Pad out to eight hexadecimal digits	**0xBF80 0000**

c) 7.625 this is non-negative, so the sign bit is S = 0
Convert 7.625 to binary.

7 = 4 + 2 + 1	0111_2
0.625 = 5/8 = 1/2 + 1/8	$.101_2$
7.625	111.101_2

Normalize by moving the binary point two places to the left. $1.11101 \bullet 2^2$
Thus saying that $2^2 \le 7.625 < 2^3$.

1.M = 1.11101 so M = 11101
P = 2 so P + 127 = 129 = 1000 0001

Concatenate S | (P + 127) | M	0 1000 0001 11101
Group by 4's from the left	0100 0000 1111 01
Pad out the last to four bits	0100 0000 1111 0100
Covert to hex digits	40F4 **0x40F4 0000**

d) – 8.75 this is negative, so S = 1
8.75 = 8 + 1/2 + 1/4 1000.11
$1.00011 \bullet 2^3$

1.M = 1.00011 so M = 00011
P = 3 so P + 127 = 130 1000 0010

Concatenate S | (P + 127) | M	1 1000 0010 00011
Group by 4's from the left	1100 0001 0000 11
Pad out the last to four bits	1100 0001 0000 1100
Convert to hex digits	C10C **0xC10C 0000**

8 Give the value of the real number (in standard decimal representation) represented by the following 32-bit words stored as an IEEE standard single precision.

a) 4068 0000 b) 42E8 0000 c) C2E8 0000

ANSWERS:

The first step in solving these problems is to convert the hexadecimal to binary.

a) 4068 0000 = 0100 0000 0110 1000 0000 0000 0000 0000
Regroup to get 0 1000 0000 1101 0000 etc.
Thus s = 0 (not a negative number)
p + 127 = 100000002 = 12810, so p = 1
and m = 1101, so 1.m = 1.1101 and the number is 1.1101•21 = 11.1012.
But 11.1012 = 2 + 1 + 1/2 + 1/8 = 3 + 5/8 = **3.625**.

b) 42E8 0000 = 0100 0010 1110 1000 0000 0000 0000 0000
Regroup to get 0 1000 0101 1101 0000 etc
Thus s = 0 (not a negative number)
p + 127 = 100001012 = 128 + 4 + 1 = 133, hence p = 6
and m = 1101, so 1.m = 1.1101 and the number is 1.1101•26 = 11101002
But 11101002 = 64 + 32 + 16 + 4 = 96 + 20 = 116 = **116.0**

c) C2E80 0000 = 1100 0010 1110 1000 0000 0000 0000 0000
Regroup to get 1 1000 0101 1101 0000 etc.
Thus s = 1 (a negative number) and the rest is the same as b). So **– 116.0**

9 Consider the string of digits "2108".

a) Show the coding of this digit string in EBCDIC.
b) How many bytes does this encoding take?

ANSWER: F2 F1 F0 F8. Four bytes.

10 Consider the positive number 2108.

a) Show the representation of this number in packed decimal format.
b) How many bytes does this representation take?

ANSWER: To get an odd number of decimal digits, this must be represented with a leading 0, as 02108. 02 10 8C. Three bytes.

11 The following block of bytes contains EBCDIC characters.
Give the English sentence represented.

```
E3 C8 C5 40 C5 D5 C4 4B
```

Answer:

```
E3 C8 C5 40 C5 D5 C4 4B
T  H  E     E  N  D  .  "THE END."
```

12 Perform the following sums assuming that each is a hexadecimal number.
Show the results as 16–bit (four hexadecimal digit) results.

a) 123C + 888C

b) 123C + 99D

ANSWER: a) In hexadecimal **C + C = 18** **(12 + 12 = 24 = 16 + 8).**
3 + 8 + 1 = C **(Decimal 12 is 0xC)**
2 + 8 + 0 = A **(Decimal 10 is 0xA)**
1 + 8 + 0 = 9. **9AC8**

b) In hexadecimal **C + D = 19** **(12 + 13 = 25 = 16 + 9)**
3 + 9 + 1 = D **(Decimal 13 is 0xD)**
2 + 9 + 0 = B **(Decimal 11 is 0xB)**
1 + 0 + 0 = 1 **1BD9**

Each of the previous two problems uses numbers written as 123C and 888C.

The numbers in the problem on packed decimal are **not** the same as those in the problem on hexadecimal. They just look the same.

Interpreted as packed decimal: 123C is interpreted as the positive number 123, and 888C is interpreted as the positive number 888.

If we further specify that each of these is to be read as an integer value, then we have the two numbers +123 and +888.

Interpreted as hexadecimal, 123C is interpreted as the decimal number
$1 \bullet 16^3 + 2 \bullet 16^2 + 3 \bullet 16 + 12 = 4096 + 512 + 48 + 12 = 4{,}668$

Interpreted as hexadecimal, 888C is interpreted as the decimal number
$8 \bullet 16^3 + 8 \bullet 16^2 + 8 \bullet 16 + 12 = 32768 + 2048 + 128 + 12 = 34{,}968$

As for the number 1011, it translates to 0x3F3.

13 The two–byte entry shown below can be interpreted in a number of ways.

```
VALUE      DC X'021D'
```

a) What is its decimal value if it is interpreted as an unsigned binary integer?
b) What is its decimal value if it is interpreted as a packed decimal value?

ANSWER: As a binary integer, its value is $2 \bullet 16^2 + 1 \bullet 16 + 13 = 512 + 16 + 13 = 541$.

As a packed decimal, this has value – 21.

14 A given computer uses byte addressing with the little-endian structure.
The following is a memory map, with all values expressed in hexadecimal.

Address	104	105	106	107	108	109	10A	10B	10C	10D
Value	C2	3F	84	00	00	00	9C	C1	C8	C0

What is the value (as a decimal real number; e.g. 203.75) of the floating point number stored at address 108? Assume IEEE–754 single precision format.

Answer: The first thing to notice is that the memory is byte-addressable. That means that each address takes holds one byte or eight bits. The IEEE format for single precision numbers calls for 32 bits to be stored, so the number takes four bytes of memory.

In byte addressable systems, the 32-bit entry at address 108 is stored in the four bytes with addresses 108, 109, 10A, and 10B. The contents of these are 00, 00, 9C, and C1.

The next thing to do is to get the four bytes of the 32-bit number in order.
This is a **little-endian** memory organization, which means that the LSB is stored at address 108 and the MSB is stored as address 10B. In correct order, the four bytes are C1 9C 00 00. In binary, this becomes 1100 0001 1001 1100 0000 0000 0000 0000. Breaking into fields we get 1**100 0001 1**001 1100 0000 0000 0000 0000, with the exponent field in bold, or
1 1000 0011 0011 1000.

Thus s = 1 a negative number
e + 127 = 1000 0011_2 = 128 + 2 + 1 = 131, so e = 4
m = 00111

So the number's magnitude is $1.00111_2 \bullet 2^4 = 10011.1_2$ = 16 + 2 + 1 + 0.5 = 19.5,
and the answer is **– 19.5.**

15 Consider the 32-bit number represented by the eight hexadecimal digits BEEB 0000. What is the value of the floating point number represented by this bit pattern assuming that the IEEE-754 single-precision standard is used?

ANSWER: First recall the binary equivalents: B = 1011, E = 1110, and 0 = 0000.
Convert the hexadecimal string to binary

```
Hexadecimal:  B    E    E    B    0    0    0    0
Binary:     1011 1110 1110 1011 0000 0000 0000 0000
```

Regroup the binary according to the 1 | 8 | 23 split required by the format.

```
Binary:     1011 1110 1110 1011 0000 0000 0000 0000
Split:      1 011 1110 1 110 1011 0000 0000 0000 0000
Regrouped:  1  0111 1101  1101 0110 0000 0000 0000 000
```

The fields in the expression are now analyzed.
Sign bit: S = 1 this will become a negative number
Exponent:
The field contains 0111 1101, or 64 + 32 + 16 + 8 + 4 + 1 = 96 + 24 + 5 = 125. This number may be more easily derived by noting that 0111 1111 = 127 and this is 2 less.
The exponent is given by P + 127 = 125, or P = – 2. The absolute value of the number being represented should be in the range [0.25, 0.50), or $0.25 \le N < 0.50$.

Mantissa:
The mantissa field is 1101 0110, so 1.M = 1.1101 0110.
In decimal, this equals 1 + 1/2 + 1/4 + 1/16 + 1/64 + 1/128, also written as
(128 + 64 + 32 + 8 + 2 + 1) / 128 = (192 + 40 + 3) / 128 = 235 / 128.
The magnitude of the number equals 235 / 128 • 1/4 = 235 / 512 = 0.458984375.

16 The following are two examples of the hexadecimal representation of floating–point numbers stored in the IBM single–precision format. Give the decimal representation of each. Fractions (e.g., 1/8) are acceptable.

a) C1 64 00 00

b) 3F 50 00 00

ANSWER: a) First look at the sign and exponent byte. This is 0xC1, or 1100 0001.

Bit	0	1	2	3	4	5	6	7
Value	1	1	0	0	0	0	0	1

The sign bit is 1, so this is a negative number.
Stripping the sign bit, the exponent field is 0100 0001 or 0x41 = decimal 65.
We have (Exponent + 64) = 65, so the exponent is 1.

The value is 16^1•F, where F is 0x64, or 6/16 + 4/256.
The magnitude of the number is 16^1•(6/16 + 4/256) = 6 + 4/16 = 6.25. The value is –6.25.

b) First look at the sign and exponent byte. This 0x3F, or 0011 1111

The sign bit is 0, so this is a non–negative number.
The exponent field is 0x3F = 3•16 + 15 = decimal 63 = 64 – 1.

The value is 16^{-1}•F, where F is 0x50, or 5/16.
The magnitude of the number is 16^{-1}•(5/16) = 5/256 = 0.01953125.

17 These questions refer to the IBM Packed Decimal Format.
a) How many bytes are required to represent a 3–digit integer?
b) Give the Packed Decimal representation of the positive integer 123.
c) Give the Packed Decimal representation of the negative integer –107.

ANSWER: Recall that each decimal digit is stored as a hexadecimal digit, and that the form calls for one hexadecimal digit to represent the sign.

a) One needs four hexadecimal digits, or two bytes, to represent three decimal digits.

b) `12 3C` c) `10 7D`

18 These questions also refer to the IBM Packed Decimal Format.

a) How many decimal digits can be represented in Packed Decimal form if three bytes (8 bits each) are allocated to store the number?

b) What is the Packed Decimal representation of the largest integer stored in 3 bytes?

ANSWER: Recall that N bytes will store 2•N hexadecimal digits. With one of these reserved for the sign, this is (2•N – 1) decimal digits.

a) 3 bytes can store **five decimal digits**.

b) The largest integer is 99,999. It is represented as **`99 99 9C`**.

19 Convert the following numbers to their representation IBM Single Precision floating point and give the answers as hexadecimal digits.

a) 123.75

b) –123.75

ANSWER:

a) First convert the number to hexadecimal.

The whole number conversion: 123 / 16 = 7 with remainder = 11 (B)
7 / 16 = 0 with remainder 7. 123 = 7B.

The fractional part conversion: .75•16 = 12 (C). The number is 7B.C

The number can be represented as 16^2 • 0.7BC; the exponent is 2.

The exponent stored with excess 64, thus it is 66 or X'42'.

Appending the fractional part, we get X'427BC'.

Add three hexadecimal zeroes to pad out the answer to X'**`427B C000`**'

b) The only change here is to add the sign bit as the leftmost bit.

In the positive number, the leftmost byte was X'**`42`**', which in binary would be

Bit	0	1	2	3	4	5	6	7
Value	0	1	0	0	0	0	1	0

Just flip the bit in position 0 to get the answer for the leftmost byte.

Bit	0	1	2	3	4	5	6	7
Value	1	1	0	0	0	0	1	0

This is X'**`C2`**'. The answer to this part is X'**`C27B C000`**'

Convert the following numbers to their representation in packed decimal.
Give the hexadecimal representation with the proper number of hexadecimal digits.

a) 123.75

b) –123.7

ANSWER: **a)** 12375 has five digits. It is represented as **`12 37 5C`**.

b) 1237 has four digits. Expand to 01237 and represent as **`01 23 7D`**.

20 Give the correct Packed Decimal representation of the following numbers.

a) 31.41 b) –102.345 c) 1.02345

ANSWER: Recall that the decimal is not stored, and that we need to have an odd count of decimal digits.

a) This becomes 3141, or 03141. 03141C

b) This becomes 102345, or 0102345 0102345D

c) This also becomes 102345, or 0102345 0102345C.

21 Perform the following sums of numbers in Packed Decimal format. Convert to standard integer and show your math. Use Packed Decimal for the answers.

a) **025C** + **085C**

b) **032C** + **027D**

c) **10003C** + **09999D**

d) **666D** + **444D**

e) **091D** + **0C**

ANSWER: Just do the math required and convert back to standard Packed Decimal format.

a) **025C** + **085C** represents 25 +85 = 110. This is represented as **110C**.

b) **032C** + **027D** represents 32 –27 = 5. This is represented as **5C**.

c) **10003C** + **09999D** represents 10003 –9999 = 4. This is represented as **4C**.

d) **666D** + **444D** represents –666 –444 = –1110. This is represented as **01 11 0D**.

e) **091D** + **0C** represents –91 +0 = –91 This is represented as **091D**.

22 These questions concern 10–bit integers, which are not common.

a) What is the range of integers storable in 10–bit unsigned binary form?

b) What is the range of integers storable in 10–bit two's–complement form?

c) Represent the positive number 366 in 10–bit two's–complement binary form.

d) Represent the negative number –172 in 10–bit two's–complement binary form.

e) Represent the number 0 in 10–bit two's–complement binary form.

ANSWER: Recall that an N–bit scheme can store 2^N distinct representations.
For unsigned integers, this is the set of integers from 0 through $2^N - 1$.
For 2's–complement, this is the set from $-(2^{N-1})$ through $2^{N-1} - 1$.

a) For 10–bit unsigned the range is 0 though $2^{10} - 1$, or 0 through 1023.

b) For 10–bit 2's–complement, this is $-(2^9)$ through $2^9 - 1$, or – 512 through 511.

c)

366 / 2	= 183	remainder = 0
183 / 2	= 91	remainder = 1
91 / 2	= 45	remainder = 1
45 / 2	= 22	remainder = 1
22 / 2	= 11	remainder = 0
11 / 2	= 5	remainder = 1
5 / 2	= 2	remainder = 1
2 / 2	= 1	remainder = 0
1 / 2	= 0	remainder = 1. **READ BOTTOM TO TOP!**

The answer is 1 0110 1110, or **`01 0110 1110`**, which equals **`0x16E`**.
`0x16E` = 1•256 + 6•16 + 14 = 256 + 96 + 14 = 256 + 110 = 366.

The number is not negative, so we stop here. Do not take the two's complement unless the number is negative.

d)

172 / 2	= 86	remainder = 0
86 / 2	= 43	remainder = 0
43 / 2	= 21	remainder = 1
21 / 2	= 10	remainder = 1
10 / 2	= 5	remainder = 0
5 / 2	= 2	remainder = 1
2 / 2	= 1	remainder = 0
1 / 2	= 0	remainder = 1. **READ BOTTOM TO TOP!**

This number is 1010 1100, or **`00 1010 1100`**, which equals **`0x0AC`**.
`0xAC` = 10•16 + 12 = 160 + 12 = 172.

The absolute value:	**`00 1010 1100`**
Take the one's complement:	**`11 0101 0011`**
Add one:	**`1`**
The answer is:	**`11 0101 0100`** or **`0x354.`**

e) The answer is **`00 0000 0000`**.
You should just know this one.

23 These questions IBM Packed Decimal Form.

a) Represent the positive number 366 as a packed decimal with fewest digits.

d) Represent the negative number –172 as a packed decimal with fewest digits.

e) Represent the number 0 as a packed decimal with fewest digits.

ANSWER: a) **`366C`**
b) **`172D`**
c) **`0C`** (not **`0D`**, which is incorrect)