IV

Data And Program Representation

Digital Logic

- Built on two-valued logic system
- Can be interpreted as
  - *Five volts* and *zero volts*
  - *High* and *low*
  - *True* and *false*

Data Representation

- Builds on digital logic
- Applies familiar abstractions
- Interprets sets of Boolean values as
  - Numbers
  - Characters
  - Addresses
- Underneath, it’s all just bits...

Bit (Binary Digit)

- Direct representation of digital logic values
- Assigned mathematical interpretation
  - *0* and *1*
- Multiple bits used to represent complex data item
- The the same underlying hardware can represent bits of an integer and bits of a character

Byte

- Set of multiple bits
- Size depends on computer
- Examples of byte sizes
  - CDC: 6-bit byte
  - BBN: 10-bit byte
  - IBM: 8-bit byte
- On most computers, smallest addressable unit of storage
- Note: following most modern computers, we will assume an 8-bit byte

Byte Size And Values

- Number of bits per byte determines range of values that can be stored
- Byte of k bits can store 2^k values
- Examples
  - Six-bit byte can store 64 possible values
  - Eight-bit byte can store 256 possible values

Binary Representation

000	010	100	110
001	011	101	111

- All possible combinations of three bits

Meaning Of Bits

- Bits themselves have no intrinsic meaning
- Byte merely stores string of 0’s and 1’s
- All interpretation determined by use

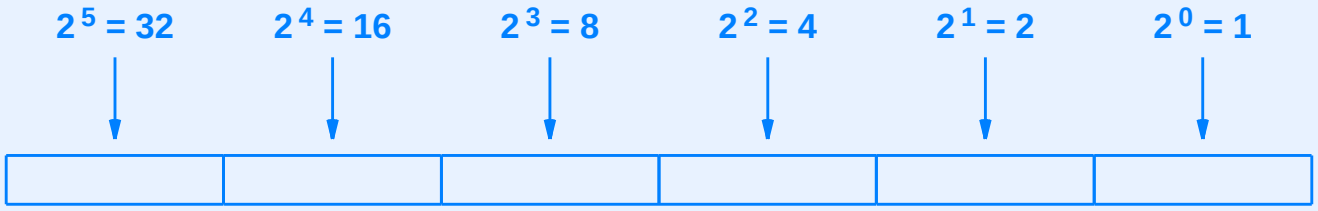
Example Of Interpretation

- Assume three bits used for status of peripheral devices
  - First bit has the value 1 if a disk is connected
  - Second bit has the value 1 if a printer is connected
  - Third bit has the value 1 if a keyboard is connected

Arithmetic Values

- Combination of bits interpreted as an integer
- Positional representation uses base 2
- Note: interpretation must specify order of bits

Illustration Of Positional Interpretation



- Example:

$$0\ 1\ 0\ 1\ 0\ 1$$

is interpreted as:

$$0 \times 2^5 + 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 21$$

The Range Of Values

A set of k bits can be interpreted to represent a binary integer. When conventional positional notation is used, the values that can be represented with k bits range from 0 through $2^k - 1$.

Hexadecimal Notation

- Convenient way to represent binary data
- Uses base 16
- Each hex digit encodes four bits

Hexadecimal Digits

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

Hexadecimal Constants

- Supported in some programming languages
- Typical syntax: constant begins with *0x*
- Example:

```
0xDEC90949
```

Character Sets

- Symbols for upper and lower case letters, digits, and punctuation marks
- Set of symbols defined by computer system
- Each symbol assigned unique bit pattern
- Typically, character set size determined by byte size

Example Character Encodings

- EBCDIC
- ASCII
- Unicode

EBCDIC

- Extended Binary Coded Decimal Interchange Code
- Defined by IBM
- Popular in 1960s
- Still used on IBM mainframe computers
- Example encoding: lower case letter *a* assigned binary value

10000001

ASCII

- American Standard Code for Information Interchange
- Vendor independent: defined by American National Standards Institute (ANSI)
- Adopted by PC manufacturers
- Specifies 128 characters
- Example encoding: lower case letter *a* assigned binary value

01100001

Full ASCII Character Set

00	nul	01	soh	02	stx	03	etx	04	eot	05	enq	06	ack	07	bel
08	bs	09	ht	0A	lf	0B	vt	0C	np	0D	cr	0E	so	0F	si
10	dle	11	dc1	12	dc2	13	dc3	14	dc4	15	nak	16	syn	17	etb
18	can	19	em	1A	sub	1B	esc	1C	fs	1D	gs	1e	rs	1F	us
20	sp	21	!	22	"	23	#	24	$	25	%	26	&	27	’
28	(	29	)	2A	*	2B	+	2C	,	2D	–	2E	.	2F	/
30	0	31	1	32	2	33	3	34	4	35	5	36	6	37	7
38	8	39	9	3A	:	3B	;	3C	<	3D	=	3E	>	3F	?
40	@	41	A	42	B	43	C	44	D	45	E	46	F	47	G
48	H	49	I	4A	J	4B	K	4C	L	4D	M	4E	N	4F	O
50	P	51	Q	52	R	53	S	54	T	55	U	56	V	57	W
58	X	59	Y	5A	Z	5B	[	5C	\	5D	]	5E	^	5F	_
60	‘	61	a	62	b	63	c	64	d	65	e	66	f	67	g
68	h	69	i	6A	j	6B	k	6C	l	6D	m	6E	n	6F	o
70	p	71	q	72	r	73	s	74	t	75	u	76	v	77	w
78	x	79	y	7A	z	7B	{	7C	|	7D	}	7E	~	7F	del

Unicode

- Each character is 16 bits long
- Can represent larger set of characters
- Motivation: accommodate languages such as Chinese

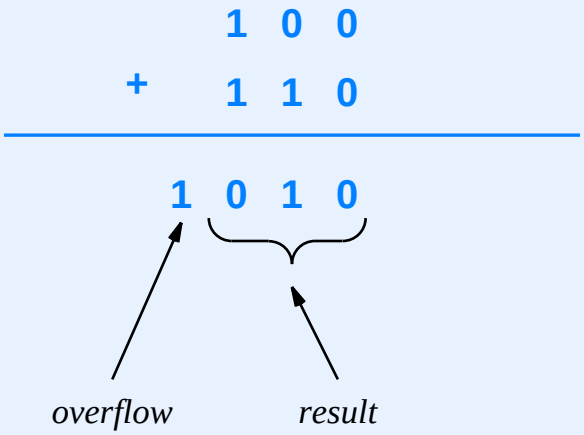
Integer Representation In Binary

- Each binary integer represented in k bits
- Computers have used k = 8, 16, 32, 60, and 64
- Many computers support multiple integer sizes (e.g., 16, 32, and 64 bit integers)
- 2^k possible bit combinations exist for k bits
- Positional interpretation produces *unsigned integers*

Unsigned Integers

- Straightforward positional interpretation
- Each successive bit represents next power of 2
- No provision for negative values
- Arithmetic operations can produce *overflow* or *underflow* (result cannot be represented in *k* bits)
- Handled with *wraparound* and *carry bit*

Illustration Of Overflow



- Values wrap around address space
- Hardware records overflow in separate carry indicator

Signed Values

- Needed by most programs
- Several representations possible
- Each has been used in at least one computer
- Some bit patterns used for negative values (typically half)
- Tradeoff: unsigned representation cannot store negative values, but can store integers that are twice as large as a signed representation

Example Signed Integer Representations

- Sign magnitude
- One’s complement
- Two’s complement
- Note: each has interesting quirks

Sign Magnitude Representation

- Familiar to humans
- First bit represents sign
- Successive bits represent absolute value of integer
- Interesting quirk: can create negative zero

One's Complement Representation

- Positive number uses positional representation
- Negative number formed by inverting all bits of positive value
- Example of 4-bit one's complement
  - 0 0 1 0 represents 2
  - 1 1 0 1 represents –2
- Interesting quirk: two representations for zero (all 0's and all 1's)

Two's Complement Representation

- Positive number uses positional representation
- Negative number formed by subtracting 1 from positive value and inverting all bits of result
- Example of 4-bit two's complement
  - 0 0 1 0 represents 2
  - 1 1 1 0 represents –2
  - High-order bit is set if number is negative
- Interesting quirk: one more negative values than positive values

Example Of Values In Unsigned And Two's Complement Representations

Binary Value	Unsigned Equivalent	Two's Complement Equivalent
1111	15	–1
1110	14	–2
1101	13	–3
1100	12	–4
1011	11	–5
1010	10	–6
1001	9	–7
1000	8	–8
0111	7	7
0110	6	6
0101	5	5
0100	4	4
0011	3	3
0010	2	2
0001	1	1
0000	0	0

Implementation Of Unsigned And Two's Complement

A computer can use a single piece of hardware to provide unsigned or two's complement integer arithmetic; software running on the computer can choose an interpretation for each integer.

- Example (k = 4)
  - Adding 1 to binary 1 0 0 1 produces 1 0 1 0
  - Unsigned interpretation goes from 9 to 10
  - Two's complement interpretation goes from –7 to –6

Sign Extension

- Needed when computer has multiple sizes of integers
- Works for unsigned and two's complement representations
- Extends high-order bit (known as *sign bit*)

Explanation Of Sign Extension

- Assume computer
  - Supports 32-bit and 64-bit integers
  - Uses two's complement representation
- When 32-bit integer assigned to 64-bit integer, correct numeric value requires upper sixteen bits to be filled with zeroes for positive number or ones for negative number
- In essence, sign bit from shorter integer must be *extended* to fill high-order bits of larger integer

Example Of Sign Extension During Assignment

- The 8-bit version of integer –3 is:

1 1 1 1 1 1 0 1

- The 16-bit version of integer –3 is:

1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1

- During assignment to a larger integer, hardware copies all bits of smaller integer and then replicates the high-order (sign) bit in remaining bits

Example Of Sign Extension During Shift

- Right shift of a negative value should produce a negative value
- Example
  - Shifting –4 one bit should produce –2 (divide by 2)
  - Using sixteen-bit representation, –4 is:

    1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0

- After right shift of one bit, value is –2:

    1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0

- Solution: replicate high-order bit during right shift

Summary Of Sign Extension

Sign extension: in two's complement arithmetic, when an integer Q composed of K bits is copied to an integer of more than K bits, the additional high-order bits are made equal to the top bit of Q. Extending the sign bit means the numeric value remains the same.

A Consequence For Programmers

Because two's complement hardware performs sign extension, copying an unsigned integer to a larger unsigned integer changes the value; to prevent such errors from occurring, a programmer or a compiler must add code to mask off the extended sign bits.

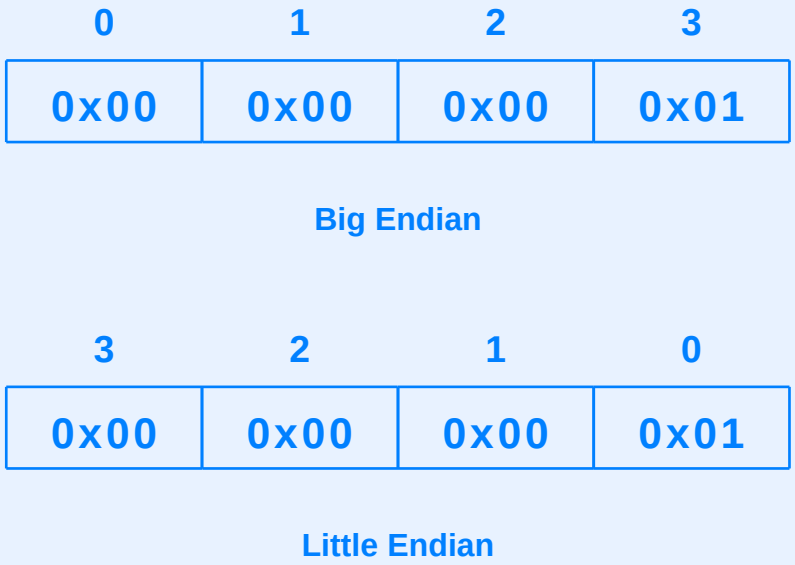
Numbering Bits And Bytes

- Need to choose order for
  - Storage in physical memory system
  - Transmission over serial medium (e.g., a data network)
- Bit order
  - Handled by hardware
  - Usually hidden from programmer
- Byte order
  - Affects multi-byte data items such as integers
  - Visible and important to programmer

Possible Byte Order

- Least significant byte of integer in lowest memory location
  - Known as *little endian*
- Most significant byte of integer in lowest memory location
  - Known as *big endian*
- Other orderings
  - Digital Equipment Corporation once used an ordering with sixteen-bit words in big endian order and bytes within the words in little endian order.
- Note: only big and little endian storage are popular

Illustration Of Big And Little Endian Byte Order



- Note: difference is especially important when transferring data between computers for which the byte ordering differs

Floating Point

- Fundamental idea: follow standard scientific representation
- Store two basic items
- Example: Avogadro's number

$$6.022 \times 10^{23}$$

Floating Point Representation

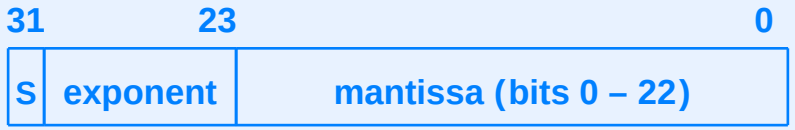
- Use base 2 instead of base 10
- Keep two conceptual items
  - Exponent that specifies the order of magnitude in a base
  - Mantissa that specifies most significant part of value

Optimizing Floating Point

- Value is normalized
- Leading bit is implicit
- Exponent is biased to allow negative values
- Normalization eliminates leading zeroes
- No need to store leading bit (0 is special case)

Example Floating Point Representation: IEEE Standard 754

- Specifies single-precision and double-precision representations
- Widely adopted by computer architects



Special Values In IEEE Floating Point

- Zero
- Positive infinity
- Negative infinity
- Note: infinity values handle cases such as the result of dividing by zero

Range Of Values In IEEE Floating Point

- Single precision range is:

$$2^{-126} \text{ to } 2^{127}$$

- Decimal equivalent is approximately:

$$10^{-38} \text{ to } 10^{38}$$

- Double precision range is:

$$10^{-308} \text{ to } 10^{308}$$

Data Aggregates

- Typically arranged in contiguous memory
- Example: three integers

- More details later in the course

Summary

- Basic output from digital logic is a bit
- Bits grouped into sets to represent
  - Integers
  - Characters
  - Floating point values
- Integers can be represented as
  - Sign magnitude
  - One’s complement
  - Two’s complement

Summary

- One piece of hardware can be used for both
  - Two's complement arithmetic
  - Unsigned arithmetic
- Bytes of integer can be numbered in
  - Big-endian order
  - Little-endian order
- Organizations such as ANSI and IEEE define standards for data representation