University of Arizona

CSc 340

Foundations of Computer Systems

Christian Collberg

January 16, 2001

Data Representation

Slide 2–1

Representing Integers

- So far we've ignored how the computer represents numbers. I assumed that numbers could be transmitted over wires, but left out the details.
- One possibility is to use the wire's voltage to represent the number, e.g. 5 volts = 5, 17 volts = 17, etc. This is an analog representation.
- Problems:
  1. It's dangerous to compute Bill Gate's worth.
  2. You can increase your salary by scuffing your feet and touching the payroll computer.

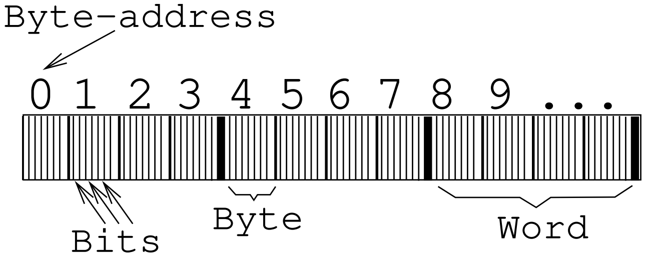
Slide 2–2

Bits, Bytes, and Words

- Instead, computers represent numbers in digital format. A number is a sequence of bits, each of which can have the values 0 or 1. Voltages above a threshold are 1, below are 0.
- A collection of 8 bits is known as a byte. A byte is the smallest amount of memory that can be accessed. Each byte has its own address, and the addresses n and $n+1$ refer to consecutive bytes.

Slide 2–3

Bits, Bytes, and Words ...



- A word is a larger collection of bits, usually 2 or 4 bytes. On a 32-bit computer a word is 4 bytes or 32 bits. The word size is usually determined by the number of bits in a register.

Slide 2–4

Positional Number Systems

- That's great, but how do we use a bunch of 1s and 0s to represent a number? First, let's look at numbering systems in general.
- Most numbering systems are positional number systems, in which a sequence of digits represents a number, such that the position of a digit within the sequence indicates the magnitude of its value.

Slide 2–5

Positional Number Systems . . .

- Example: The value two-hundred and thirty-eight is represented by the sequence "238" (that is, the digit "2", followed by the digit "3", followed by the digit "8"). The position of each digit indicates its value

$$\begin{aligned} \text{"238"} &= 2\cdot 100 + 3\cdot 10 + 8\cdot 1 \\ &= 2\cdot 10^2 + 3\cdot 10^1 + 8\cdot 10^0 \\ &= 238 \end{aligned}$$

- Base 10 numbering is called decimal. It's what humans use, probably because we have ten fingers.

Slide 2–6

Positional Number Systems . . .

- In general, the sequence of digits:

$$d_{n-1}\cdots d_2 d_1 d_0$$

  represents the value:

$$d_{n-1}\cdot r^{n-1} + \cdots + d_0\cdot r^0.$$

  r is called the radix (or the base).
- For decimal numbers the radix is 10.
- The radix for a number is usually indicated as a subscript, e.g. 101_2 is base 2, 101_8 is base 8, etc.

Slide 2–7

Positional Number Systems . . .

- Unlike English text, numbers are read from right-to-left. The least- significant digit is on the right, and represents the radix to the zeroth power, while the most-significant digit is on the left and represents the radix to the highest power.
- Computers use bits to represent numbers, so the numbers are necessarily base 2. These are binary numbers.
- This is a very important point – all computers store numbers in binary format.

Slide 2–8

Converting Binary to Decimal

- Multiply each digit by the radix raised to the appropriate power, and sum:

$$\begin{aligned} 325_7 &= 3 \cdot 7^2 + 2 \cdot 7^1 + 5 \cdot 7^0 \\ &= 3 \cdot 49 + 2 \cdot 7 + 5 \\ &= 147 + 14 + 5 \\ &= 166_{10} \end{aligned}$$

Slide 2–9

Converting Decimal to Binary

- Repeatedly divide by the target radix. Each remainder is the next least- significant digit of the number.
- Example: $489_{10} = X_5$?

Division	Quotient	Remainder	Result
489/5	97	4	4_5
97/5	19	2	24_5
19/5	3	4	424_5
3/5	0	3	3424_5

Check: $3424_5 = 3 \cdot 5^3 + 4 \cdot 5^2 + 2 \cdot 5^1 + 4 \cdot 5^0 = 489_{10}$

Slide 2–10

Converting Between Bases

- Convert to decimal first, then to the target base.
- Example: $634_7 = X_{13}$?

$$634_7 = 6 \cdot 7^2 + 3 \cdot 7 + 4 = 319_{10}$$

Division	Quotient	Remainder	Result
319/13	24	7	7_{13}
24/13	1	11	B_{13}
1/13	0	1	$1B7_{13}$

Check: $1B7_{13} = 1 \cdot 13^2 + 11 \cdot 13^1 + 7 = 319_{10}$

Slide 2–11

Octal and Hexadecimal

- Sometimes the conversion is easy, if both radixes are powers of the same number. This allows numbers to be converted by grouping digits.
- For example, to convert a decimal number to a base 100 number each pair of decimal digits corresponds to a single base 100 digit.

Slide 2–12

Octal and Hexadecimal ...

- Computers use binary numbers, but it's unpleasant for humans to deal with long strings like 1110101_2.
- The larger the base, the fewer digits it takes to represent the number.
- Decimal representations are shorter (1110101_2 is 117_{10}) but it's not easy to convert between binary and decimal.
- Therefore, people often use octal (base 8) and hexadecimal (base 16) because they are powers of two, and can be converted to and from binary easily:

Slide 2–13

Decimal	Binary	Octal	Hexadecimal
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	1 0000	20	10

Slide 2–14

Octal and Hexadecimal ...

- Note that octal uses the digits 0-7, and hexadecimal 0-F. Hexadecimal needs 6 more digits than decimal, and instead of inventing new symbols the letters A-F are used.
- Each octal digit represents three binary digits, and each hex digit represents four. Digits are grouped from the right (least-significant), and leading zeros are added achieve the correct grouping.

Slide 2–15

Converting between powers of 2

- Convert 2537_8 into binary

$$\begin{aligned} 2537_8 &= 010\ 101\ 011\ 111_2 \\ &= 010101011111_2 \end{aligned}$$

- Convert 101101001010_2 into octal

$$\begin{aligned} 101101001010_2 &= 101\ 101\ 001\ 010_2 \\ &= 5512_8 \end{aligned}$$

Slide 2–16

Converting between powers of 2 ...

- Convert $3A9_{16}$ into binary

$$\begin{aligned} 3A9_{16} &= 0011\ 1010\ 1001_2 \\ &= 001110101001_2 \end{aligned}$$

- Convert 101101001010_2 into hexadecimal

$$\begin{aligned} 101101001010_2 &= 1011\ 0100\ 1010_2 \\ &= B4A_{16} \end{aligned}$$

Slide 2–17

Converting between powers of 2 ...

- Convert 4013_8 into hexadecimal

$$\begin{aligned} 4013_8 &= 100\ 000\ 001\ 011_2 \\ &= 1000\ 0000\ 1011_2 \\ &= 80B_{16} \end{aligned}$$

Slide 2–18

Fractions

- Fractions are expressed via digits to the right of radix point. These digits represent decreasing negative powers of the radix.
- The number

$$0.d_1 d_2 \ldots d_n$$

represents the value

$$d_1 \cdot r^{-1} + d_2 \cdot r^{-2} + ... + d_n \cdot^{r-n} .$$

- Example:

$$0.234 = 2 \cdot 10^{-1} + 3 \cdot 10^{-2} + 4 \cdot 10^{-3}$$

Slide 2–19

Convert non-decimal fractions to decimal

- Multiply each digit by the radix to the appropriate power and sum
- Convert 0.117_8 into decimal:

$$\begin{aligned} 0.117_8 &= 1 \cdot 8^{-1} + 1 \cdot 8^{-2} + 7 \cdot 8^{-3} \\ &= \frac{1}{8^1} + \frac{1}{8^2} + \frac{7}{8^3} \\ &= \frac{1}{8} + \frac{1}{64} + \frac{7}{512} \\ &= 0.154296875_{10} \end{aligned}$$

Slide 2–20

Convert decimal fractions to non-decimal

- Multiply the fraction by the target radix.
- Integer portion is next digit, starting with most-significant.
- Repeat with remaining fraction. Note that this may never terminate.
- Convert 0.380859375_{10} into hex:

$$0.380859375_{10} \cdot 16 = \boxed{6}.09375 \qquad 0.6_{16}$$

$$0.09375 \cdot 16 = \boxed{1}.5 \qquad 0.61_{16}$$

$$0.5 \cdot 16 = \boxed{8}.0 \qquad 0.618_{16}$$

Slide 2–21

Binary-Coded Decimal (BCD)

- Another way of encoding decimal numbers in a binary computer is binary-coded decimal. Its (only) advantage is that it's easy to convert between BCD and decimal. Each digit of the decimal number is encoded using a 4-bit binary representation.

Digit	Binary	Digit	Binary	Digit	Binary
0	0000	4	0100	8	1000
1	0001	5	0101	9	1001
2	0010	6	0110	+	1010
3	0011	7	0111	-	1011

Slide 2–22

Binary-Coded Decimal (BCD) . . .

- The '+' and '-' signs also have 4-bit encodings, allowing positive and negative numbers to be represented:
- Convert 37 to BCD: 0011 0111
- Convert -42 to BCD: 1011 0100 0010
- Convert 1001 0111 from BCD to decimal: 97
- Note that some 4-bit values do not represent BCD digits: 1100, 1101, 1110, and 1111. BCD is therefore less "dense" than binary; a 4-bit binary number can represent 16 values, a 4-bit BCD number only 10.

Slide 2–23

Character Encodings

- Strings of characters are represented by grouping bits together so that each group represents one character.
- ASCII (American Standard Code for Information Interchange) is the most common. Each character is represented by a 7-bit quantity.
- Because computers typically have 8-bits per byte, each ASCII digit is stored in one byte with the MSB=0.
- Convert the following sequence of bytes into characters:

01001000	01100101	01101100	01101100	01101111
'H'	'E'	'L'	'L'	'0'

Character Encodings . . .

- Unicode is gradually replacing ASCII. Unicode contains non-English characters, allowing Greek, Hebrew, etc. alphabets to be encoded. Each Unicode character is 16-bits; basically, the upper byte indicates the alphabet and the lower byte the character, although some alphabets have more than 256 characters and use several values for the upper byte. If the upper byte is 0, the lower byte is an ASCII character, providing some backwards-compatibility.
- EBCDIC was originally used in IBM mainframes. Perhaps it still is.

Slide 2–24

Char	Dec	Hex	Oct	Bin
^@ (NUL)	0	0x00	00	00000000
^A (SOX)	1	0x01	01	00000001
^B (STX)	2	0x02	02	00000010
^C (ETX)	3	0x03	03	00000011
^D (EOT)	4	0x04	04	00000100
^E (ENQ)	5	0x05	05	00000101
^F (ACK)	6	0x06	06	00000110
^G (BEL)	7	0x07	07	00000111
^H (BS)	8	0x08	010	00001000
^I (HT)	9	0x09	011	00001001
^J (NL)	10	0x0A	012	00001010
^K (VT)	11	0x0B	013	00001011
^L (NP)	12	0x0C	014	00001100
^M (CR)	13	0x0D	015	00001101
^N (SO)	14	0x0E	016	00001110
^O (SI)	15	0x0F	017	00001111
^P (DLE)	16	0x10	020	00010000

Slide 2–25

Char	Dec	Hex	Oct	Bin
^Q (DS1)	17	0x11	021	00010001
^R (DS2)	18	0x12	022	00010010
^S (DS3)	19	0x13	023	00010011
^T (DC4)	20	0x14	024	00010100
^U (NAK)	21	0x15	025	00010101
^V (SYN)	22	0x16	026	00010110
^W (ETB)	23	0x17	027	00010111
^X (CAN)	24	0x18	030	00011000
^Y (EM)	25	0x19	031	00011001
^Z (SUB)	26	0x1A	032	00011010
^[(ESC)	27	0x1B	033	00011011
^\ (FS)	28	0x1C	034	00011100
^] (GS)	29	0x1D	035	00011101
^^ (RS)	30	0x1E	036	00011110
^_ (US)	31	0x1F	037	00011111
(SP)	32	0x20	040	00100000
!	33	0x21	041	00100001

Slide 2–26

Char	Dec	Hex	Oct	Bin
"	34	0x22	042	00100010
#	35	0x23	043	00100011
$	36	0x24	044	00100100
%	37	0x25	045	00100101
&	38	0x26	046	00100110
'	39	0x27	047	00100111
(	40	0x28	050	00101000
)	41	0x29	051	00101001
*	42	0x2A	052	00101010
+	43	0x2B	053	00101011
,	44	0x2C	054	00101100
-	45	0x2D	055	00101101
.	46	0x2E	056	00101110
/	47	0x2F	057	00101111
0	48	0x30	060	00110000
1	49	0x31	061	00110001
2	50	0x32	062	00110010

Slide 2–27

Char	Dec	Hex	Oct	Bin
3	51	0x33	063	00110011
4	52	0x34	064	00110100
5	53	0x35	065	00110101
6	54	0x36	066	00110110
7	55	0x37	067	00110111
8	56	0x38	070	00111000
9	57	0x39	071	00111001
:	58	0x3A	072	00111010
;	59	0x3B	073	00111011
<	60	0x3C	074	00111100
&	61	0x3D	075	00111101
>	62	0x3E	076	00111110
?	63	0x3F	077	00111111
@	64	0x40	0100	01000000
A	65	0x41	0101	01000001
B	66	0x42	0102	01000010
C	67	0x43	0103	01000011

Slide 2–28

Char	Dec	Hex	Oct	Bin
D	68	0x44	0104	01000100
E	69	0x45	0105	01000101
F	70	0x46	0106	01000110
G	71	0x47	0107	01000111
H	72	0x48	0110	01001000
I	73	0x49	0111	01001001
J	74	0x4A	0112	01001010
K	75	0x4B	0113	01001011
L	76	0x4C	0114	01001100
M	77	0x4D	0115	01001101
N	78	0x4E	0116	01001110
O	79	0x4F	0117	01001111
P	80	0x50	0120	01010000
Q	81	0x51	0121	01010001
R	82	0x52	0122	01010010
S	83	0x53	0123	01010011
T	84	0x54	0124	01010100

Slide 2–29

Char	Dec	Hex	Oct	Bin
U	85	0x55	0125	01010101
V	86	0x56	0126	01010110
W	87	0x57	0127	01010111
X	88	0x58	0130	01011000
Y	89	0x59	0131	01011001
Z	90	0x5A	0132	01011010
[	91	0x5B	0133	01011011
\	92	0x5C	0134	01011100
]	93	0x5D	0135	01011101
^	94	0x5E	0136	01011110
_	95	0x5F	0137	01011111
‘	96	0x60	0140	01100000
a	97	0x61	0141	01100001
b	98	0x62	0142	01100010
c	99	0x63	0143	01100011
d	100	0x64	0144	01100100
e	101	0x65	0145	01100101

Slide 2–30

Char	Dec	Hex	Oct	Bin
f	102	0x66	0146	01100110
g	103	0x67	0147	01100111
h	104	0x68	0150	01101000
i	105	0x69	0151	01101001
j	106	0x6A	0152	01101010
k	107	0x6B	0153	01101011
l	108	0x6C	0154	01101100
m	109	0x6D	0155	01101101
n	110	0x6E	0156	01101110
o	111	0x6F	0157	01101111
p	112	0x70	0160	01110000
q	113	0x71	0161	01110001
r	114	0x72	0162	01110010
s	115	0x73	0163	01110011
t	116	0x74	0164	01110100
u	117	0x75	0165	01110101
v	118	0x76	0166	01110110

Slide 2–31

Slide 2–32

Char	Dec	Hex	Oct	Bin
w	119	0x77	0167	01110111
x	120	0x78	0170	01111000
y	121	0x79	0171	01111001
z	122	0x7A	0172	01111010
{	123	0x7B	0173	01111011
|	124	0x7C	0174	01111100
}	125	0x7D	0175	01111101
~	126	0x7E	0176	01111110
^? (DEL)	127	0x7F	0177	01111111

Slide 2–33

Negative numbers

- So far we haven't discussed encoding negative numbers in binary. This is intentional. We'll cover negative numbers when we get to arithmetic, where they are needed.

Slide 2–34

Big Endian vs. Little Endian

- There is an issue of how to store a 4-byte integer in memory. Assume we have a 4-byte integer, where byte 'A' is the most-significant and byte 'D' is the least.

A	B	C	D
MSB			LSB

- The same issue arrises when organizing bits in a byte.

Slide 2–35

Big Endian vs. Little Endian . . .

- When storing this integer there are many possible ways of assigning its four bytes to four bytes of memory. There are two popular ones:

Big-Endian				Little-Endian			
0	1	2	3	0	1	2	3
A	B	C	D	D	C	B	A
MSB			LSB	LSB			MSB

Big Endian vs. Little Endian . . .

- Big-endian: the most-significant byte (A) is stored in the given memory address, and least-significant byte (D) at the address + 3.
- Little-endian: the least-significant byte (D) is stored at the given memory address, and the most-significant at the address + 3.
- Neither one is clearly better than the other.
- You can configure most CPUs to do it either way. The MIPS is usually big-endian, the x86 family little-endian.

Readings and References

- See Table 1.7 in Maccabe for the full ASCII character set.
- Read Maccabe, Chapter 1, section 1.1–1.3.1, pp. 3–19.