Review of HTML

Ch. 1

Review of tags covered

- <html> </html>
- <body></body>
- <title></title>
-
- <table></table> <table border =“1”>

<tr></tr> <td></td><caption></caption>

-
-
-
- <h2> various header tags
- Img tag
- Style, attributes and values
- alt

Digitizing Data

- Chapter 7

Digitizing Discrete Information

- The dictionary definition of digitize is to represent information with digits.
- Digit means the ten Arabic numerals 0 through 9.
- Digitizing uses whole numbers to stand for things.

The PandA Representation

- PandA is the name used for two fundamental patterns of digital information:
  - Presence
  - Absence
- PandA is the mnemonic for “Presence and Absence”
- A key property of PandA is that the phenomenon is either present or not

A Binary System

- The PandA encoding has two patterns: present and absent
- Two patterns make it a binary system
- There is ***no*** law that says ***on*** means "***present***" or ***off*** means "***absent***"

Table 7.1 Possible interpretations of the two PandA patterns

Present	Absent
True	False
1	0
On	Off
Yes	No
+	–
Black	White
For	Against
Yang	Yin
X	Y
. . .	. . .

Bits Form Symbols

- *In the PandA representation, the unit is a specific place (in space and time), where the presence or absence of the phenomenon can be set and detected.*
- The PandA unit is known as a ***bit***
- ***Bit*** is a contraction for "***b***inary dig***it***"
- Bit sequences can be interpreted as binary numbers
- Groups of bits form symbols

Bits in Computer Memory

- Memory is arranged inside a computer in a very long sequence of bits
- Going back to the definition of bits (previous slide), this means that places where the physical phenomenon encoding the information can be ***set*** and ***detected***

Hex Explained

- ***Hex*** digits, short for ***hex***adecimal digits, are base-16 numbers
- A bit sequence might be given in 0's and 1's:
  - 11111111100110001110001010 10
- Writing so many 0's and 1's is tedious and error prone
- There needed to be a better way to write bit sequences…hexadecimal digits

The 16 Hex Digits

- The digits of the hexadecimal numbering system are 0, 1, ... , 9, A, B, C, D, E, F
- Because there are 16 digits (***hexits***), they can be represented perfectly by the 16 symbols of 4-bit sequences:
  - The bit sequence 0000 is hex 0
  - Bit sequence 0001 is hex 1
  - Bit sequence 1111, is hex F

Hex to Bits and Back Again

- Because each hex digit corresponds to a 4-bit sequence, easily translate between hex and binary

  - 0010 1011 1010 1101
    2 B A D

  - F A B 4
    1111 1010 1011 0100

Digitizing Numbers in Binary

- The two earliest uses of PandA were to:
  - Encode numbers
  - Encode keyboard characters
- Representations for sound, images, video, and other types of information are also important

Counting in Binary

- Binary numbers are limited to two digits, 0 and 1
- Digital numbers are ten digits, 0 through 9
- The number of digits is the base of the numbering system

Counting to ten

0	
1	110
10	111
11	1000
100	1001
101	1010

Counting in Binary

- With decimal numbers, we use a place value representation where each "place" represents the next higher power of 10
- With binary numbers, it is the same idea, but with higher powers of 2

Table 7.4 The decimal number 1,010 representing one thousand ten = 1,000 + 10

10^3	10^2	10^1	10^0	Decimal Place Values
1	0	1	0	Digits of Decimal Number
1×10^3	0×10^2	1×10^1	0×10^0	Multiply place digit by place value
1,000	0	10	0	and add to get a decimal 1,010

Place Value in a Decimal Number

- Recall that To find the quantity expressed by a decimal number:
  - The digit in a place is multiplied by the place value and the results are added
- Example, 1010 (base 10) is:
  - Digit in the 1's place is multiplied by its place
  - Digit in the 10's place is multiplied by its place
  - and so on:
    $(0 \times 1) + (1 \times 10) + (0 \times 100) + (1 \times 1000)$

Place Value in a Binary Number

- Binary works the same way
- The base is not 10 but 2
- Instead of the decimal place values:
  1, 10, 100, 1000, . . . ,
  the binary place values are:
  1, 2, 4, 8, 16, . . . ,

Power	Decimal	Binary
0	$1 = 10^0$	$1 = 2^0$
1	$10 = 10^1$	$2 = 2^1$
2	$100 = 10^2$	$4 = 2^2$
3	$1000 = 10^3$	$8 = 2^3$
4	$10{,}000 = 10^4$	$16 = 2^4$
. . .	. . .	. . .

Place Value in a Binary Number

- 1010 in binary:
  - (1 × 8) + (0 × 4) + (1 × 2) + (0 × 1)

Table 7.5 The binary number 1010, representing the decimal number ten = 8 + 2

2^3	2^2	2^1	2^0	Binary Place Values
1	0	1	0	Bits of Binary Number
1×2^3	0×2^2	1×2^1	0×2^0	Multiply place bit by place value
8	0	2	0	and add to get a decimal 10

Table 7.6 Binary representation of the decimal number one thousand ten = 11 1111 0010

2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	Binary Place Values
1	1	1	1	1	1	0	0	1	0	Bits of Binary Number
1×2^9	1×2^8	1×2^7	1×2^6	1×2^5	1×2^4	0×2^3	0×2^2	1×2^1	0×2^0	Multiply place bit by place value
512	256	128	64	32	16	0	0	2	0	and add to get decimal 1,010

Digitizing Text

- The number of bits determines the number of symbols available for representing values:
  - *n bits in sequence yield 2n symbols*
- The more characters you want encoded, the more symbols you need

Digitizing Text

- Roman letters, Arabic numerals, and about a dozen punctuation characters are about the minimum needed to digitize ***English*** text
- What about:
  - Basic arithmetic symbols like +, −, *, /, =?
  - Characters not required for English ö, é, ñ, ø?
  - Punctuation? « », ¿, π, ∀)? What about business symbols: ¢, £, ¥, ©, and ®?
  - And so on.

Assigning Symbols

- We need to represent:
  - 26 uppercase,
  - 26 lowercase letters,
  - 10 numerals,
  - 20 punctuation characters,
  - 10 useful arithmetic characters,
  - 3 other characters (new line, tab, and backspace)
  - 95 symbols…enough for English

Assigning Symbols

- To represent 95 distinct symbols, we need 7 bits
  - 6 bits gives only $2^6 = 64$ symbols
  - 7 bits give $2^7 = 128$ symbols
- 128 symbols is ample for the 95 different characters needed for English characters
- Some additional characters must also be represented

Assigning Symbols

- ***ASCII*** stands for American Standard Code for Information Interchange
- ASCII is a widely used 7-bit (2^7) code
- The advantages of a “standard” are many:
  - Computer parts built by different manufacturers can be connected
  - Programs can create data and store it so that other programs can process it later, and so forth

Extended ASCII: An 8-Bit Code

- 7-bit ASCII is not enough, it cannot represent text from other languages
- IBM decided to use the next larger set of symbols, the ***8***-bit symbols (2^{8})
- Eight bits produce $2^{8} = 256$ symbols
  - The 7-bit ASCII is the 8-bit ASCII representation with the leftmost bit set to 0
  - Handles many languages that derived from the Latin alphabet

NATO Broadcast Alphabet

- The code for the letters used in radio communication is ***purposely*** inefficient
- The code is distinctive when spoken amid **noise**
- The alphabet encodes letters as words
  - Words are the symbols
  - “Mike” and “November” replace “em” and “en”
- The longer encoding improves the chance that letters will be recognized
- Digits keep their usual names, except nine, which is known as ***niner***

NATO Broadcast Alphabet

Table 7.7 NATO broadcast alphabet designed not to be minimal

A	Alpha	H	Hotel	O	Oscar	V	Victor
B	Bravo	I	India	P	Papa	W	Whiskey
C	Charlie	J	Juliet	Q	Quebec	X	X-ray
D	Delta	K	Kilo	R	Romeo	Y	Yankee
E	Echo	L	Lima	S	Sierra	Z	Zulu
F	Foxtrot	M	Mike	T	Tango		
G	Golf	N	November	U	Uniform		

Bar Codes

- Universal Product Codes (UPC) also use more than the minimum number of bits to encode information
- In the UPC-A encoding, 7 bits are used to encode the digits 0 – 9

Bar Codes

- UPC encodes the manufacturer (left side) and the product (right side)
- Different bit combinations are used for each side
- One side is the complement of the other side
- The bit patterns were chosen to appear as different as possible from each other

Bar Codes

- Different encodings for each side make it possible to recognize whether the code is right side up or upside down

Table 7.8 Bit encoding for bars for the UPC-A encoding; notice that the two sides are complements of each other, that is, the 0's and 1's are switched.

Digit	Left Hand Side	Right Hand Side
0	0001101	1110010
1	0011001	1100110
2	0010011	1101100
3	0111101	1000010
4	0100011	1011100
5	0110001	1001110
6	0101111	1010000
7	0111011	1000100
8	0110111	1001000
9	0001011	1110100

Why “Byte”?

- Computer memory is subject to errors
- An extra bit is added to the memory to help detect errors
  - A ninth bit per byte can detect errors using parity
- ***Parity*** refers to whether a number is even or odd
  - Count the number of 1’s in the byte. If there is an even number of 1’s we set the ninth bit to 0

Why “Byte”?

- All 9-bit groups have even parity:
  - Any single bit error in a group causes its parity to become odd
  - This allows hardware to detect that an error has occurred
  - It cannot detect *which* bit is wrong, however

Why “Byte”?

- IBM was building a supercomputer, called Stretch
- They needed a word for a quantity of memory ***between*** a bit and a word”
  - A word of computer memory is typically the amount required to represent computer instructions (currently a word is 32 bits)

Why “Byte”?

- Then, why not bite?

- The ‘i’ to a ‘y’ was done so that someone couldn’t accidentally change ‘byte’ to ‘bit’ by the dropping the ‘e’ ”
  - bit**e** bit (*the meaning changes)*
  - byt**e** byt (*what’s a byt?)*

Summary

- We began the chapter by learning that digitizing doesn't require digits—any symbols will do
- We explored the following:
  - PandA encoding, which is based on the presence and absence of a physical phenomenon. Their patterns are discrete; they form the basic unit of a bit. Their names (most often 1 and 0) can be any pair of opposite terms.

Summary

- We explored the following:
  - 7-bit ASCII, an early assignment of bit sequences (symbols) to keyboard characters. Extended or 8-bit ASCII is the standard.
  - The need to use more than the minimum number of bits to encode information.
  - The mystery of the *y in byte.*