CS271

Week 4

Binary Arithmetic

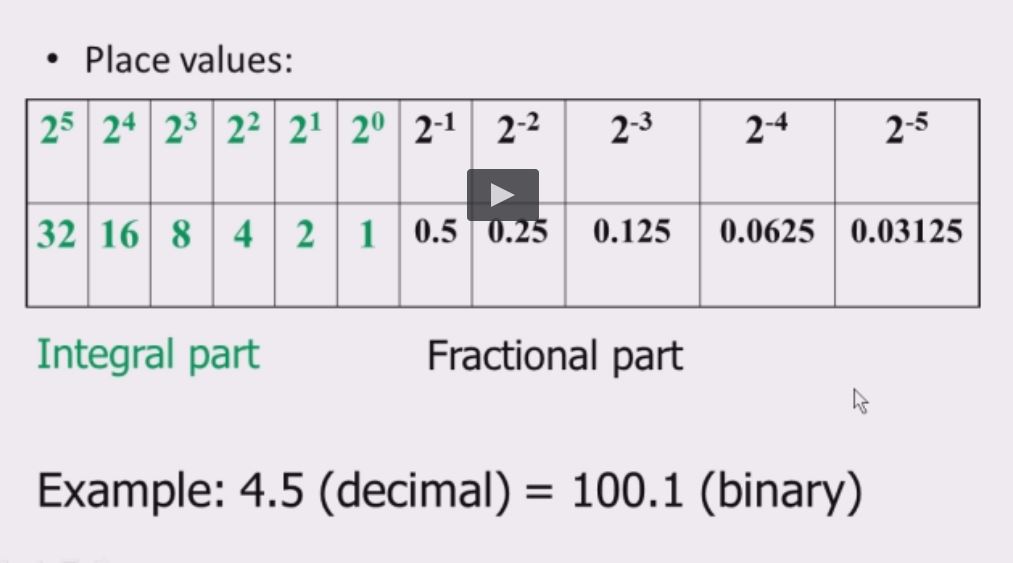
* Following examples use 8-bit twos complement operands
  + Everything extends to 16-bit, 32-bit, n-bit representations.
  + What is range of values for 8-bit operands?
* Usual arithmetic operands can be performed directly in binary form with b-bit repos

|  |  |  |
| --- | --- | --- |
| + | 0 | 1 |
| 0 | 0 | 1 |
| 1 | 1 | 10 |

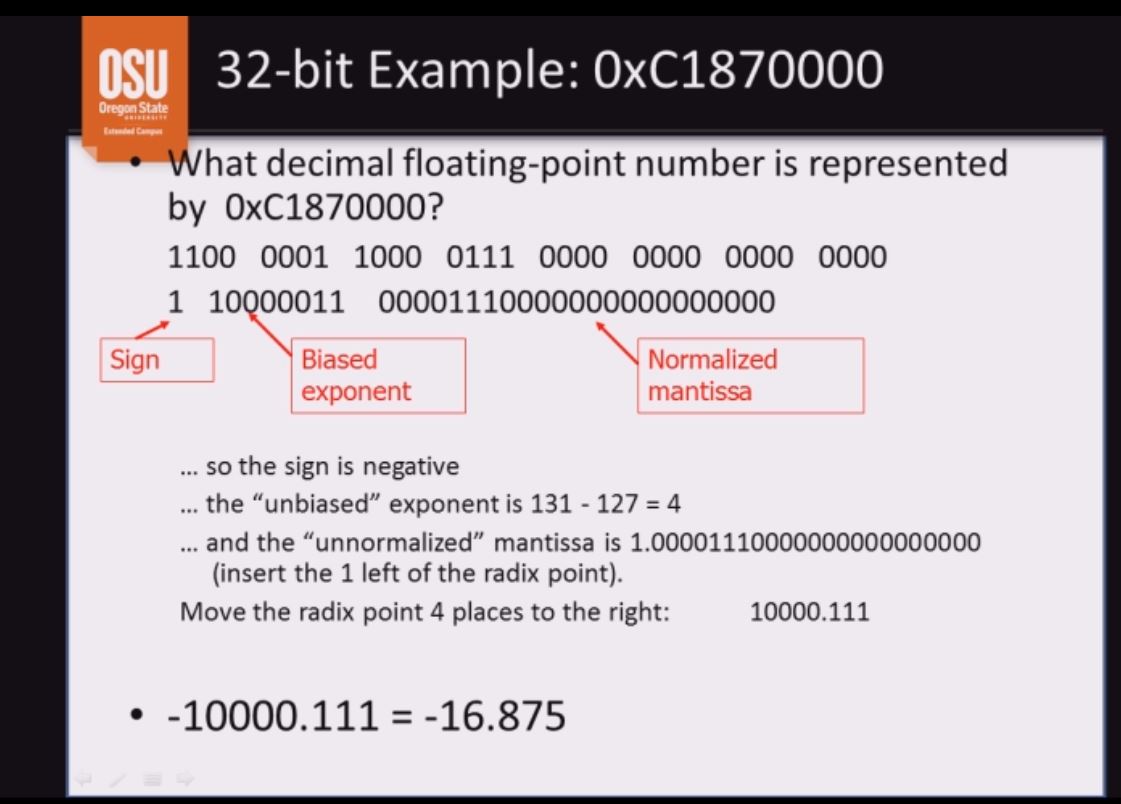
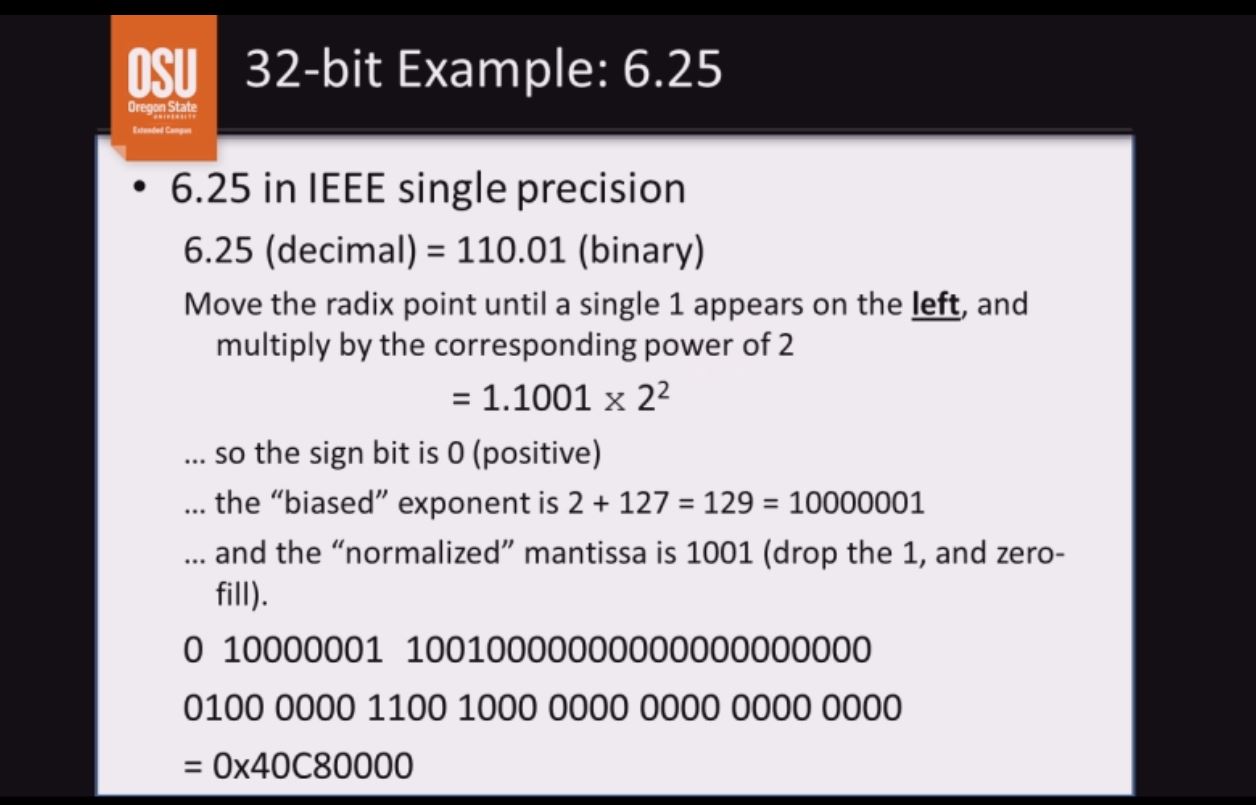
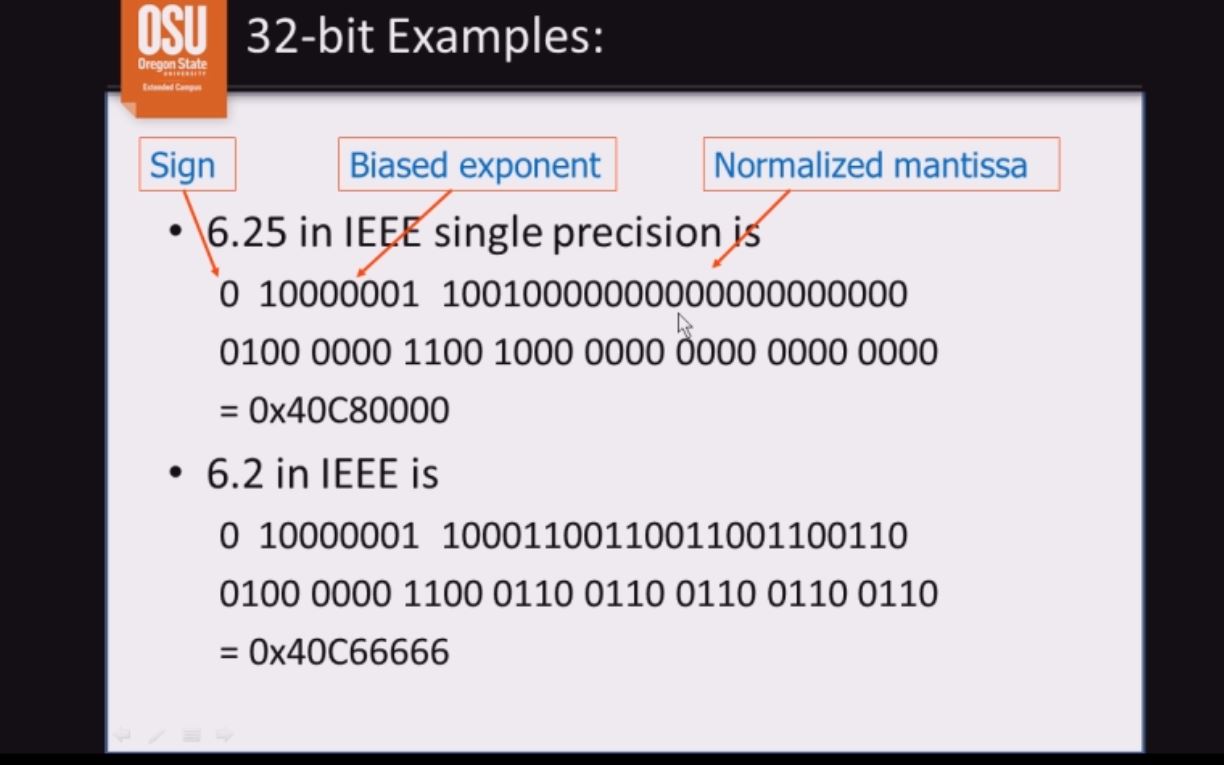
* + Use the usual rules of add and carry
    - With two operands, the carry bit is never greater than 1
    - 0+0+1 = 01, 0+1+1=10, 1+0+1=10, 1+1+1=11
    - Example:
      * 00101101
      * +00011110
      * 01001011
        + This is done by
        + 1111
        + 00101101 = 45
        + 00011110 = 30
        + ----------------
        + 01001011 = 75
  + How does overflow occur
    - Possible results might produce 9-bit result
* Binary subtraction
  + Similar to adding
  + Order of operands, align and borrow from left, like normal
  + Or take twos complement and add instead
* Verification
  + Perform operation on binary operands
  + Convert results to decimal
  + Convert operands to decimal
  + Perform operation on decimal operands
  + Compare results
* Binary Multiplication
  + Usual algorithm
  + Ex: 45x5 = 225
    - 00101101
    - X 101
    - 00101101
    - 001011010
    - -------------------
    - 11100001
  + Or repeated addition
    - Just add it 5 times, more
  + Start by mult by 2
    - 00101101
    - 01011010 = 64+16+8+2 = 90
    - Shifting left one space = mult by 2
      * 45x5 = shift two places to left = mult by 4 then add it once 1 for 5th
      * 10110100
      * 00101101
      * Add together!
* Binary Division
  + Usual algorithm
    - 45/7
    - 1111101101
  + Repeated subtraction
  + If divisor power of 2, shift right and keep track of dropped bits
* Arithmetic operations
  + NOTE: all of the integer arithmetic operations can be accomplished only using add complement
    - Addition
    - Complement
  + Addition: check
  + Subtraction: complement and add
  + Mult: repeated add
  + Division repeated subtract
  + Comparison: non-destructive subtract
* Byte Ordering
  + When it takes more than one byte to rep a value
  + Big Endian
    - Bytes are ordered left-> right(most significant to least significant ) in each word
    - Used in motorla archs, (Mac) and others
  + Little-endian
    - Bytes are ordered least significant to most in each word
    - Used in Intel Architectures
  + For both schemes:
    - Within each byte, bit values are stored left->right(as usual)
    - Each char is one byte
    - Strigns are stored in byte order
  + PROBLEM: communication between architectures
* EX: -1234
  + Have to know if you are using big or little endian
  + We are using intel, thus using little endian
  + Big endian (big end first)
    - Memory address
    - 1004 1005 1006 1007
    - 11111111 11111111 11111011 00101110
    - FF FF FB 2E
    - Byte3 Byte2 Byte1 Byte0
  + Little endian
    - Memory address
    - 1004 1005 1006 1007
    - 00101110 11111011 11111111 11111111
    - 2E FB FF FF
    - Byte0 Byte1 Byte2 Byte3
  + Internet communication must be consist across architectures
  + Network order is always big endian

Floating point Representation

* Decimal means base ten
* Floating point means a number with an integral part and fractional part
  + Sometimes called real, float
  + Generic term for decimal point is radix point
  + Or called binary point, hexapoint
* Decimal to binary

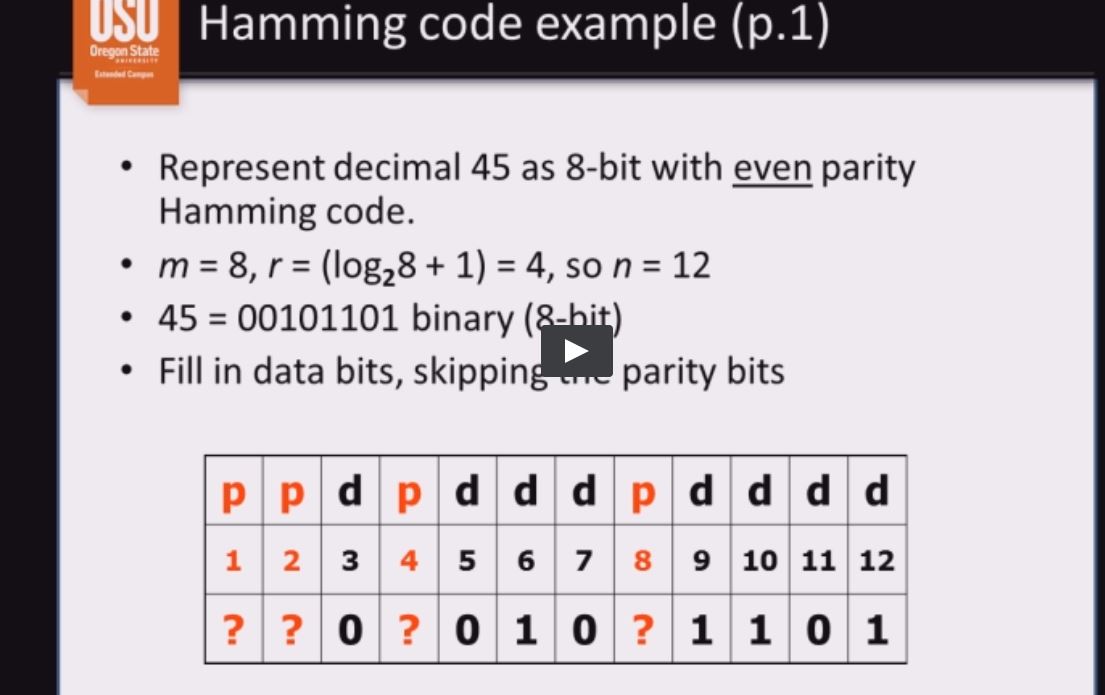


* EX:
  + 6.25 = 110.01
  + Method
    - 6 = 110 Integral part: convert in the usual way
    - .25 x 2 = 0.5 Fractional part: successive multiplication by 2
    - .5 x 2 = 1.0 Stop the when fractional part is 0.
* Could be repeating, thus ignore as its infinite.
* Some architectures handle the integer part and the fraction part separately
* Most use a complelete different representation (IEEE standard) and a separate ALU (floating point unit).
  + Faster operations
* IEEE 754 standard
  + Single precision = 32bit
  + Double precision = 64bit
  + Extended = 80bit
  + 3 parts
    - 1 sign bit
    - Biased exponent
      * Single: 8 bits, double: 11 bits, extended 15 bits.
      * Normalized mantissa
        + Single: 23 bits, double, 52 bits, extended: 64 bits

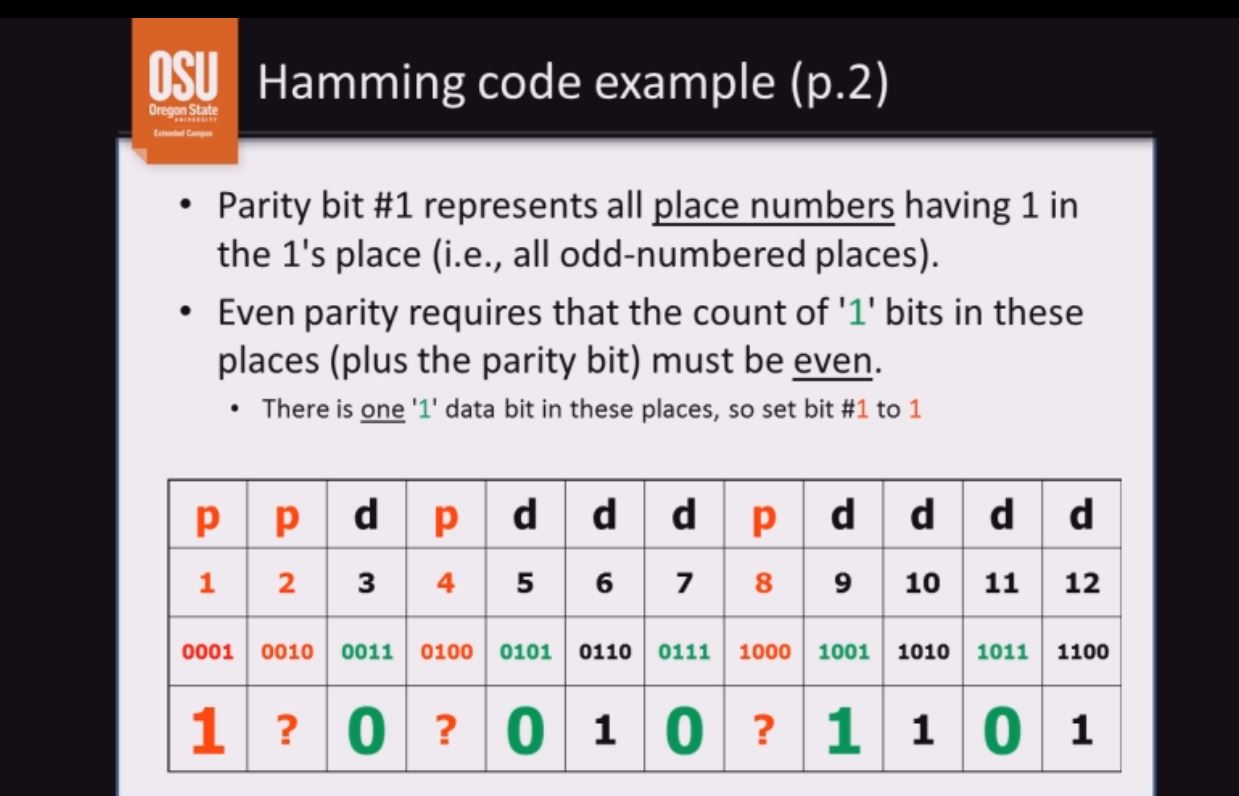


Error-detecting and error correcting codes

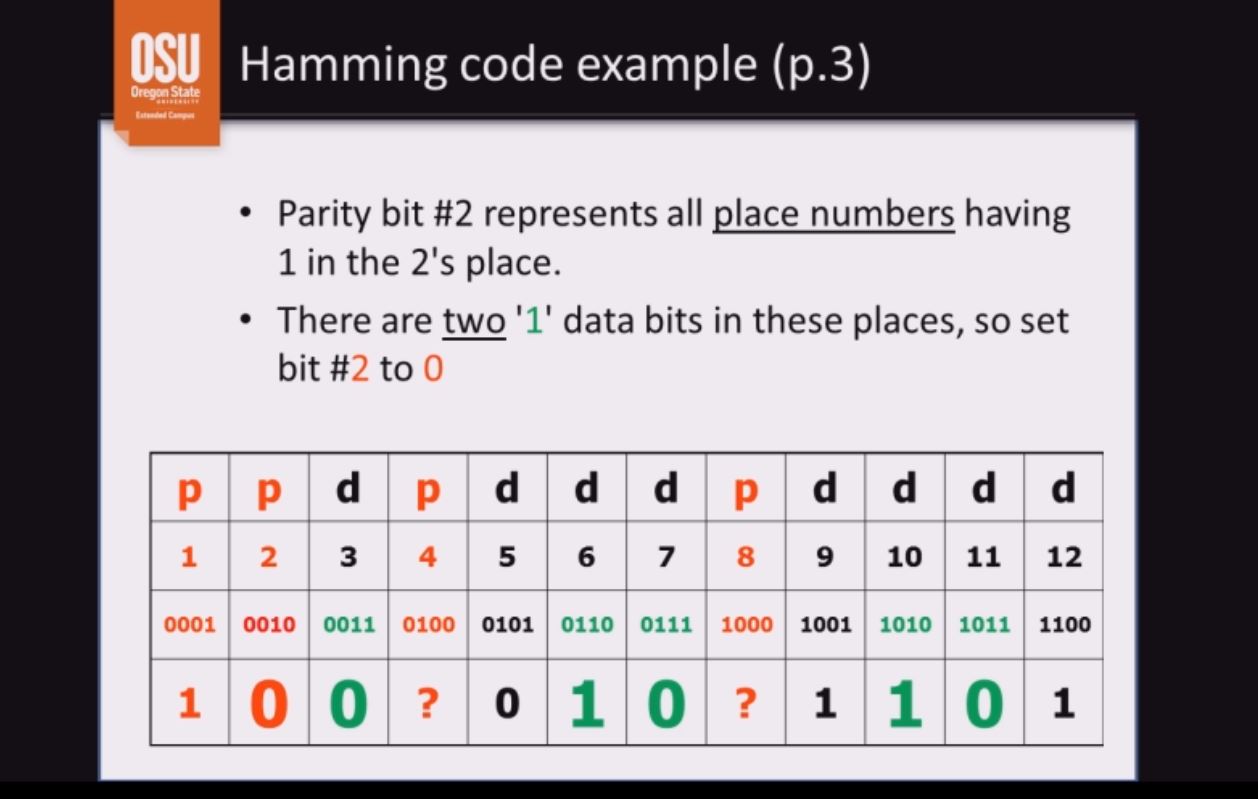
* Parity 0 is the total number if 1 bits in binary code
* Each comp archs is designed to use even parity or odd parity
* EX parity bit for 8-bit code 11010110
  + Even parity system: (1)11010110
    - Sets parity bit to 1 to make a total of 6 one bits
  + Odd parity system: (0)11010110
    - Sets parity bit to 0 to keep 5 one bits
* Code is checked for parity error whever used.
  + Even parity
    - 101010101 (error 5 one bits)
    - 100101010 (OK 4 one bits)
  + Odd parity
    - 101010101 (OK 5 one bits)
    - 100101010 (error 4 one bits)
* Used for network transmissions
  + Error detection
* Not 100% reliable
  + Works only works when error is in odd number of bits
  + Good because most errors are single bit errors
* Error correcting – Hamming codes
  + N-bit code work (n=m+r)
    - M – data bits
    - R check bits (to check parity)
    - There are 2^n possible code works
    - Only 2^m code words are valid
  + Parity is the sum of one check bit and its selected data bits
    - May be even or odd
    - Used for detecting and correcting errors in memory, network transmissions, etc
      * ECC memory, etc.
* Number of parity bits depends on word size
  + Number of required parity bits ® is log2m+1
  + Guarantees hamming distance of 2
  + Invalid codes mean error
* Number the bits left->right, 1->n
* Bits numbered with powers of 2 are parity bits; others are data bits.



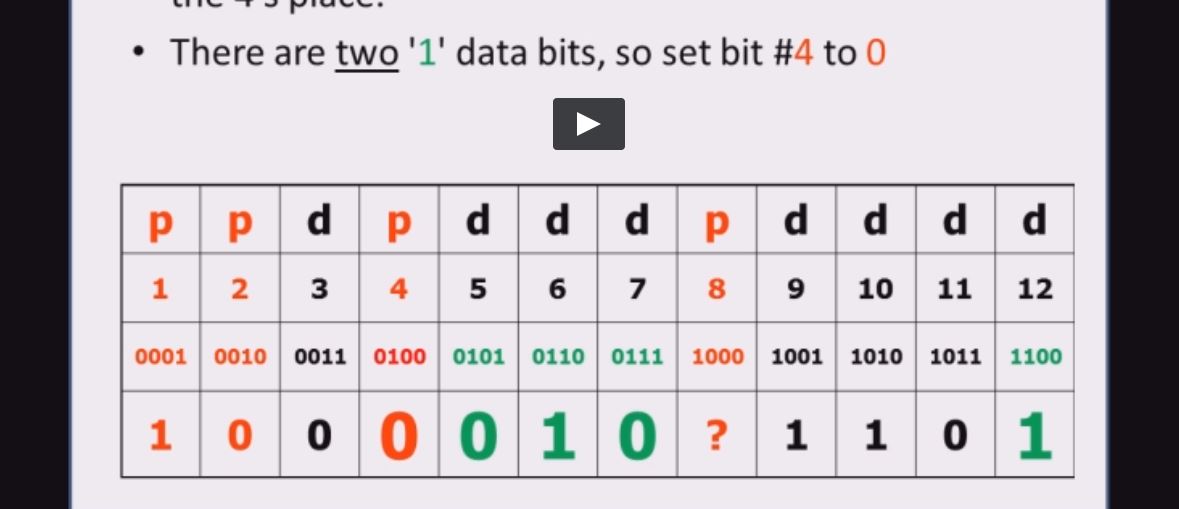
* Parity bit #1 represents all place numbers having 1 in the 1’s place (ie all odd numbered places)
* Even parity requires that the count of ‘1’ bits in these places (plus the parity bit) must be even.
  + There is one ‘1’ data bit in these places, so set bit #1 to 1.



* + Parity bit #2 represents all palce numbers having 1 in the 2’s palce.
  + There are two ‘1’ data bits in these places, so set bet #2 to 0



* + Parity #4 represents all place numbers having 1 in the 4’s place
  + There are two ‘1’ data bits, so set bit #4 to 0



* + Parity #8 represents all place numbers having 1 in the 8’s place.
  + There are three ‘1’ data bits so set bit #8 to 1.



* + 45 = 00101101 (8bit binary)
  + 45 = 100001011101 (12-bit even parity Hamming code)



* + Do SELF CHECK exercise for hamming codes